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

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αυτόματη Κατασκευή Ωρολογίων Προγραμμάτων μέσω Προγραμματισμού με Περιορισμούς

Νικόλαος Ι. Ποθητός

Επιβλέπων: Παναγιώτης Σταματόπουλος, Επίκ. Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ
ΔΕΚΕΜΒΡΙΟΣ 2005

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αυτόματη Κατασκευή Ωρολογίων
Προγραμμάτων μέσω Προγραμματισμού
με Περιορισμούς

Νικόλαος Ι. Ποθητός

Α.Μ.: 1115200100088

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:
Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής ΕΚΠΑ

Περίληψη

Η πρακτική συνεισφορά αυτής της εργασίας είναι διττή. Κατά πρώτο λόγο, σχεδιάστηκε και υλοποιήθηκε σε αντικειμενοστραφές περιβάλλον προγραμματισμού, μία βιβλιοθήκη επίλυσης (γενικών) προβλημάτων ικανοποίησης περιορισμών. Εξάλλου, ο προγραμματισμός με περιορισμούς είναι μια σύγχρονη μέθοδος εύκολης διατύπωσης και αποδοτικής επίλυσης προβλημάτων βελτιστοποίησης. Ένα τέτοιο πρόβλημα —που αντιμετωπίστηκε στο δεύτερο κομμάτι της εργασίας— είναι η κατάρτιση ωρολογίων προγραμμάτων για εκπαιδευτικά ιδρύματα, όπως πανεπιστήμια και σχολεία. Ο στόχος μας σε αυτά τα προβλήματα δεν είναι απλά να βρούμε μία λύση, αλλά και η ποιότητα αυτής της λύσης να είναι ικανοποιητική· π.χ. να μην υπάρχουν πολλές κενές ώρες ανάμεσα στα μαθήματα που παρακολουθεί ένας φοιτητής στη σχολή του. Το «πάντρεμα» της βιβλιοθήκης και του συστήματος κατάρτισης ωρολογίων προγραμμάτων αποτέλεσε μία πραγματική δοκιμασία για την αξιοπιστία αμφότερων των πλευρών.

Θεματική περιοχή: Τεχνητή Νοημοσύνη

Λέξεις-κλειδιά: προγραμματισμός με περιορισμούς, επιλυτής, συνέπεια-ορίων, χρονοπρογραμματισμός, ωρολόγιο πρόγραμμα

Abstract

This work has two main practical contributions. First of all, a library that solves (generic) constraint satisfaction problems (CSPs) was designed and implemented, in an object-oriented programming environment. Besides, constraint programming is a modern method that facilitates the declaration and efficient solving of optimization problems. A problem of this kind—which was coped with in the second part of this work—is automated timetabling for educational institutions, e.g. universities and schools. Our goal in these problems is not to find a solution simply, but to produce qualitative solutions; for example, there should not be many gaps (i.e. idle hours) in a student's personal timetable. The connection between the solver-library and automated timetabling system was a real "crash-test" for both parts' reliability.

Subject Area: Artificial Intelligence

Keywords: constraint programming, solver, bounds-consistency, scheduling, automated timetabling

Στη γιαγιά μου, Βάσω
και στη μνήμη των παππούδων μου,
Μανώλη και Νίκου

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου Παναγιώτη Σταματόπουλο για τις συζητήσεις που κάναμε, τις δεκάδες άμεσες απαντήσεις στα ηλεκτρονικά μηνύματά μου και τη βοήθειά του στο να μάθω LaTeX, έτσι ώστε η εργασία να είναι όσο το δυνατόν καλύτερη, ακόμα και στον τρόπο παρουσίασής της. Ο σημαντικός χρόνος που διέθεσε και όλες οι συμβουλές που έδωσε συνέβαλαν καθοριστικά στο να ξεπεραστούν οι όποιες δυσκολίες και εγγυήθηκαν τη θετική έκβαση της εργασίας.

Επίσης ευχαριστώ τον Κυριάκο Ζερβουδάκη για τις υποδείξεις του σχετικά με το πώς μπορεί να διατυπωθεί αντικειμενοστραφώς και να επιλυθεί ένα πρόβλημα κατάρτισης ωρολογίου προγράμματος. Οι συζητήσεις που κάναμε βοήθησαν στο να εγκλιματιστώ στο περιβάλλον του προγραμματισμού με περιορισμούς, καθώς η εμπειρία του πάνω σε αυτόν τον τομέα είναι μεγάλη.

Τέλος, να ευχαριστήσω τα ξαδέρφια μου Μάνο, Γιάννη και Βαγγέλη Πρεβενιό που μου παραχωρούσαν τον υπολογιστή τους, όποτε τους ζητούσα να κάνω δοκιμές.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

«Θερμοπαρακαλώ τον Σύλλογο Φοιτητών ή όποιον άλλον διαβάζει αυτό το post και είναι υπεύθυνος για το πρόγραμμα των χειμερινών μαθημάτων της σχολής!!!!!
Όταν θα συντάξετε το πρόγραμμα προσπαθήστε όσο το δυνατόν καλύτερα τα μαθήματα του 3ου έτους να είναι κολλητά με τα μαθήματα του 2ου έτους έτσι ώστε όσοι από μας χρωστάμε κάποιο μάθημα από το 3ο εξάμηνο να μην χρειαστεί να είμαστε από το πρωί ως το βράδυ στη σχολή για να παρακολουθήσουμε τα μαθήματα 2ου-3ου έτους που έχουμε δηλώσει.

Την ίδια απαίτηση μπορεί να έχει και οποιοσδήποτε φοιτητής που είναι σε άλλο έτος, πχ 4ο, αλλά η δική μου απάντηση πιστεύω ότι καλύπτει όλους εμάς που τώρα μπαίνουμε στο 3ο έτος.»

Στο παραπάνω μήνυμα[1] στο ανεπίσημο διαδικτυακό φόρουμ του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Αθηνών, ένας φοιτητής εκφράζει την ανησυχία του για το πώς θα καταρτιστεί το ωρολόγιο πρόγραμμα της σχολής, μαζί με τις προσωπικές του επιθυμίες. Και δεν είναι ο μόνος που έχει τέτοιες ανησυχίες· το ωρολόγιο πρόγραμμα πρέπει να ικανοποιεί το σύνολο των φοιτητών, που πιθανότατα έχουν αντικρουόμενες προτιμήσεις. Αυτό το γεγονός από μόνο του κάνει την κατασκευή ενός ωρολογίου προγράμματος μια μη τετριμμένη διαδικασία. Εξάλλου έχει αποδειχθεί ότι ακόμα και η κατάρτιση ενός μικρού μεγέθους προγράμματος, όταν υπάρχουν περιορισμοί για τις ώρες και τις αίθουσες που θα γίνουν κάποια μαθήματα, είναι ένα δύσκολο, NP-πλήρες, πρόβλημα [1, 2].

[1] http://www.forums.gr/showthread.php?postid=223927#post223927

Η κατάρτιση ωρολογίου προγράμματος για εκπαιδευτικά ιδρύματα, είναι λοιπόν ένα πολυσύνθετο πρόβλημα, τόσο από άποψη επίλυσης, όσο και από την πλευρά της διατύπωσης και εύρεσης των προδιαγραφών του. Αυτή η τελευταία πλευρά του προβλήματος μάλιστα, αποτελεί ίσως το σημαντικότερο «αγκάθι» για τη χρησιμότητα των εφαρμογών για δημιουργία ωρολογίων προγραμμάτων: Υπάρχουν πολλές εφαρμογές για αυτή τη δουλειά, βγάζουν ικανοποιητικά αποτελέσματα μέσα σε λογικά χρονικά διαστήματα, όμως σπάνια λαμβάνουν υπόψη όλες τις παραμέτρους των προβλημάτων. Δηλαδή υπάρχει μία δυστοκία στην ανάπτυξη εφαρμογών που θα καλύπτουν τις ανάγκες του ευρύτερου φάσματος των εκπαιδευτικών ιδρυμάτων και είναι σύνηθες φαινόμενο κάθε ίδρυμα να αντιμετωπίζει με διαφορετικά μέσα το δικό του πρόβλημα.

Όσον αφορά τώρα τους τρόπους επίλυσης ενός τέτοιου προβλήματος, εδώ και δεκαετίες, έχουν διατυπωθεί αρκετές μέθοδοι. Έχουν χρησιμοποιηθεί διαδικασίες της Επιχειρησιακής Έρευνας [3] και των δικτύων ροής. Έχουν επίσης επιστρατευτεί μέθοδοι της Τεχνητής Νοημοσύνης, όπως η προσομοιωμένη ανόπτηση [4], η ταμπού αναζήτηση [5], οι γενετικοί αλγόριθμοι [6] και ο προγραμματισμός με περιορισμούς [7, 8, 9, 10].

Στην εργασία αυτή, χρησιμοποιήσαμε τον προγραμματισμό με περιορισμούς, ως μία μοντέρνα μέθοδο σαφούς διατύπωσης και αποτελεσματικής επίλυσης ποικίλων προβλημάτων βελτιστοποίησης. Εξάλλου, βάση αυτής της εργασίας αποτέλεσε το σύστημα κατάρτισης ωρολογίων προγραμμάτων ORPROG [11], το οποίο στηρίζεται σε αυτήν ακριβώς τη μεθοδολογία. Το ORPROG —ένα επιτυχημένο σύστημα, αποτέλεσμα συσσωρευμένης εμπειρίας ετών— χρησιμοποιήθηκε σαν μέτρο σύγκρισης, τόσο για τα αποτελέσματα που παράγονται από την εφαρμογή αυτής της εργασίας, όσο και για τη διεπαφής της, η οποία υλοποιήθηκε σε Visual Basic .NET, μια αξιόλογη γλώσσα για τη δημιουργία εύχρηστων και κομψών διεπαφών. Σημειώνεται ότι προστέθηκαν κάποιες δυνατότητες στην εφαρμογή της εργασίας, οι οποίες δεν υπάρχουν στο ORPROG και είχε ζητηθεί κατά καιρούς από τους χρήστες του, η υλοποίησή τους. Μία τέτοια δυνατότητα, είναι η προσαρμογή της εφαρμογής στις απαιτήσεις των σχολείων, στα οποία η ποιότητα των ωρολογίων προγραμμάτων κρίνεται περισσότερο με βάση το πώς κατανέμονται οι ώρες του κάθε καθηγητή χωριστά, μέσα στην εβδομάδα.

Δεδομένου ότι το ORPROG υλοποιήθηκε πάνω στην εμπορική βιβλιοθήκη επίλυσης προβλημάτων ικανοποίησης περιορισμών ILOG SOLVER [12, 13], υπάρχουν προβλήματα στη μεταφερσιμότητά του, π.χ. στον προσωπικό υπολογιστή ενός χρήστη. Αυτό στάθηκε αφορμή για τη δημιουργία ενός νέου επιλυτή, τον οποίο «βαφτίσαμε» NAXOS SOLVER. Η αντικειμενοστραφής φύση

του προγραμματισμού με περιορισμούς, μας οδήγησε στην απόφαση να τον υλοποιήσουμε στην αντικειμενοστραφή —και συνάμα αποδοτική— γλώσσα C++.[2] Φροντίσαμε έτσι ώστε ο επιλυτής να μπορεί να χρησιμοποιηθεί για την αντιμετώπιση οποιουδήποτε προβλήματος ικανοποίησης περιορισμών και όχι μόνο αυτό της κατάρτισης ωρολογίου προγράμματος.

[2]Αναλυτικότερα, χρησιμοποιήθηκε ο συμβατός με την ANSI/ISO C++, μεταγλωττιστής g++, καθώς και ο αποσφαλματωτής ddd και ο αποσφαλματωτής διαχείρισης μνήμης Valgrind.

Κεφάλαιο 2

Προγραμματισμός με περιορισμούς

Ο προγραμματισμός με περιορισμούς αποτελεί μία από τις καλύτερες προσεγγίσεις που έχει κάνει μέχρι σήμερα η επιστήμη των υπολογιστών στο Ιερό Δισκοπότηρο του προγραμματισμού: ο χρήστης δηλώνει το πρόβλημα και ο υπολογιστής το λύνει.

— Eugene C. Freuder

Ένας πολλά υποσχόμενος τομέας έρευνας της Τεχνητής Νοημοσύνης ασχολείται με τα προβλήματα ικανοποίησης περιορισμών. Τέτοια προβλήματα αποτελούνται από ένα σύνολο *περιορισμένων μεταβλητών* (constraint variables) (τις οποίες για συντομία πολλές φορές θα τις αναφέρουμε απλά σαν *μεταβλητές*), οι οποίες παίρνουν τιμές από ένα συγκεκριμένο πεπερασμένο *πεδίο* (domain) από διακριτές τιμές. Οι περιορισμένες μεταβλητές συσχετίζονται μεταξύ τους μέσω ενός συνόλου *περιορισμών* (constraints), που δεν είναι τίποτα άλλο από σχέσεις μεταξύ των μεταβλητών. Οι σχέσεις αυτές δηλώνονται ρητά και δεν σπάνε σε ενδιάμεσες γραμμικές εξισώσεις και ανισώσεις, όπως συχνά συμβαίνει σε άλλες μεθοδολογίες προγραμματισμού. Αυτό διευκολύνει τη διατύπωση ενός προβλήματος, τόσο θεωρητικά, όσο και πρακτικά μέσω της συγγραφής συντηρήσιμου κώδικα. Η σαφήνεια αυτή στη διατύπωση, την κάνει ανεξάρτητη σε μεγάλο βαθμό από την επίλυση του προβλήματος, γι' αυτό και οι μέθοδοι επίλυσης που υπάρχουν εφαρμόζονται σε όλα τα προβλήματα ικανοποίησης περιορισμών. Στιγμιότυπα τέτοιων προβλημάτων εμφανίζονται στον χρονοπρογραμματισμό, στην ανίχνευση βλαβών σε ψηφιακά κυκλώματα και σε άλλους τομείς. Τα τελευταία χρόνια έχουν αναπτυχθεί πολλά συστήματα και βιβλιοθήκες που υ-

ποστηρίζουν προγραμματισμό με περιορισμούς [14, 15, 16, 17, 18, 19, 20].

2.1 Μαθηματικός ορισμός

Ένα πρόβλημα ικανοποίησης περιορισμών είναι μια δομή $P = (V, D, C)$:

- με $V = \{V_1, V_2, \ldots, V_n\}$ το σύνολο n μεταβλητών
- με $D = \{D_1, D_2, \ldots, D_n\}$ το σύνολο των πεδίων τιμών που αντιστοιχούν στις μεταβλητές, $V_i \in D_i$, $1 \leq i \leq n$. Κάθε πεδίο τιμών είναι πεπερασμένο και αποτελείται από διακριτές τιμές.
- και με $C = \{C_1, C_2, \ldots, C_e\}$ ένα σύνολο περιορισμών, όπου C_i μια σχέση μεταξύ των μεταβλητών ενός συνόλου $S_i \subseteq V$. Ορίζουμε $C_i = (S_i, T_i)$, με $T_i \subseteq D_{i_1} \times D_{i_2} \times \cdots \times D_{i_q}$ το σύνολο με τους δυνατούς συνδυασμούς για τις τιμές των μεταβλητών του συνόλου $S_i = \{V_{i_1}, V_{i_2}, \ldots, V_{i_q}\}$. Ανάλογα με τον αριθμό των μεταβλητών της σχέσης (ο οποίος ισούται με $|S_i|$), κατατάσσουμε τον περιορισμό C_i στις εξής κατηγορίες/τάξεις περιορισμών:
  - αν $|S_i| = 1$, τότε ο C_i είναι *μοναδιαίος* (unary)
  - αν $|S_i| = 2$, τότε ο C_i είναι *δυαδικός* (binary)
  - αν $|S_i| > 2$, τότε ο C_i είναι *ανώτερης τάξης* (higher order)

Η λύση στο πρόβλημα είναι μία ανάθεση τιμών στις μεταβλητές $V_1 = d_1, V_2 = d_2, \ldots, V_n = d_n$, έτσι ώστε να ικανοποιούνται οι περιορισμοί, δηλαδή ένα στοιχείο του συνόλου

$$\{t \mid \ t \in D_1 \times D_2 \times \cdots \times D_n,\ t|_{S_i} \in T_i,\ 1 \leq i \leq e\}.$$

Ένα τυπικό πρόβλημα ικανοποίησης περιορισμών περιλαμβάνει πολλές μεταβλητές. Έτσι ο χώρος αναζήτησης που προκύπτει είναι μεγάλος και 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

Ένας χημικός προτίθεται να κάνει τις αντιδράσεις A, B, Γ και Δ. Είναι γνωστό ότι οι B και Γ χρειάζονται μια χημική ουσία που παράγεται από τη Δ. Επίσης η A χρειάζεται μια χημική ουσία που παράγεται από τη B. Με ποια σειρά μπορούν να γίνουν οι αντιδράσεις;

Το παραπάνω γίνεται να αναπαρασταθεί σαν ένα πρόβλημα ικανοποίησης περιορισμών. Ορίζουμε τις μεταβλητές V_A, V_B, V_Γ και V_Δ οι οποίες δείχνουν τη σειρά με την οποία θα γίνουν οι αντίστοιχες αντιδράσεις. Ως εκ τούτου, τα πεδία τιμών όλων τους ταυτίζονται με το σύνολο $\{1, 2, 3, 4\}$. Θεωρούμε ότι ο χημικός εκτελεί διαδοχικά τις αντιδράσεις και έτσι ισχύει ότι οι μεταβλητές είναι διαφορετικές μεταξύ τους:

$$\begin{array}{lll} V_A \neq V_B, & V_A \neq V_\Gamma, & V_A \neq V_\Delta \\ & V_B \neq V_\Gamma, & V_B \neq V_\Delta \\ & & V_\Gamma \neq V_\Delta \end{array}$$

Βάσει των δεδομένων του προβλήματος ισχύουν επίσης:

$$\begin{array}{l} V_\Delta < V_B \\ V_\Delta < V_\Gamma \\ V_B < V_A \end{array}$$

Εξάλλου, όταν μια αντίδραση X παράγει μια ουσία που χρησιμοποιείται στην Y, η X προηγείται της Y (ή $V_X < V_Y$). Ενώνοντας όλους τους παραπάνω περιορισμούς έχουμε:

$$V_A \neq V_\Gamma \tag{2.1}$$
$$V_B \neq V_\Gamma \tag{2.2}$$
$$V_\Delta < V_B \tag{2.3}$$
$$V_\Delta < V_\Gamma \tag{2.4}$$
$$V_B < V_A \tag{2.5}$$

Οι λύσεις στο πρόβλημα είναι τρεις:

V_A	V_B	V_Γ	V_Δ	Σειρά αντιδράσεων
3	2	4	1	Δ, B, Γ, A
4	2	3	1	Δ, B, A, Γ
4	3	2	1	Δ, Γ, B, A

2.3 Αλγόριθμοι επίλυσης

2.3.1 Γέννα-και-δοκίμαζε

Η πρώτη μέθοδος επίλυσης ονομάζεται *γέννα-και-δοκίμαζε* (generate-and-test). Αυτό που κάνει επαναληπτικά, είναι να *γεννά* έναν συνδυασμό των μεταβλητών του προβλήματος ικανοποίησης περιορισμών και στη συνέχεια να τον *δοκιμάζει* για να διαπιστωθεί αν είναι μια νόμιμη λύση, δηλαδή αν ικανοποιεί τους περιορισμούς.

---

1. **Ανάθεσε στις μεταβλητές τυχαίες τιμές από τα πεδία των τιμών τους.**
2. **Αν για αυτές τις τιμές των μεταβλητών τηρούνται οι περιορισμοί, τότε επίστρεψέ τις ως λύση.**
3. **Πήγαινε στο** Βήμα 1.

Αν αυτός ο αλγόριθμος εφαρμοστεί στο παράδειγμα (§2.2), τότε θα παραχθούν —και θα ελεγχθούν— όλοι οι δυνατοί συνδυασμοί:

V_A	V_B	V_Γ	V_Δ
1	1	1	1
1	1	1	2
		⋮	
4	4	4	4

Συνολικά θα γεννηθούν και θα εξεταστούν $4^4 = 256$ συνδυασμοί, από τους οποίους μόνο 3 θα είναι νόμιμοι. Μια *γεννήτρια* (generator) παραγωγής συνδυασμών θα πρέπει να έχει τις εξής ιδιότητες:

- *πληρότητα*, δηλαδή να παράγει όλους τους δυνατούς συνδυασμούς.
- να είναι *απέριττη*, που σημαίνει να παράγει έναν συνδυασμό μία μόνο φορά, έτσι ώστε να μην μειώνεται η απόδοση της μεθόδου, λόγω επανειλημμένων άσκοπων ελέγχων του ίδιου συνδυασμού.
- *ενημέρωση*, η οποία είναι προαιρετική ιδιότητα και επιτρέπει τη μείωση του χώρου αναζήτησης, μέσω της αξιοποίησης κάποιων πληροφοριών για το πρόβλημα. Στο παραπάνω παράδειγμα είναι εύκολο να συναχθεί το

συμπέρασμα ότι $V_\Delta = 1$, αφού η αντίδραση Δ γίνεται πάντα πρώτη. Με αυτήν την πληροφορία η γεννήτρια θα δημιουργήσει $4^3 = 64$ συνδυασμούς μόνο, αφού δεν θα ασχοληθεί με τη V_Δ και έτσι θα παράγει τους εξής συνδυασμούς:

V_A	V_B	V_Γ	V_Δ
1	1	1	1
1	1	2	1
		⋮	
4	4	4	1

Έχουν προταθεί βελτιώσεις για τον αλγόριθμο, όπως η *αναρρίχηση λόφου* (hill climbing) και το *ευριστικό των ελαχίστων συγκρούσεων* (min conflict heuristic), που αποσκοπούν στο να φτάνουμε πιο σύντομα σε μία λύση, μέσω της αλλαγής της σειράς με την οποία γεννιούνται οι συνδυασμοί. Γενικά πάντως, η απλότητα της μεθοδολογίας γέννα-και-δοκίμαζε δεν συνεπάγεται και υψηλή απόδοση. Αρκεί να σκεφτεί κανείς ότι σε ένα πρόβλημα που δεν έχει λύση, για να το διαπιστώσει αυτό ο αλγόριθμος, θα εξετάσει όλες τις δυνατές περιπτώσεις!

2.3.2 Οπισθοδρόμηση

Η μέθοδος της *οπισθοδρόμησης* (backtracking) αποτελεί μια καλύτερη προσέγγιση στο πρόβλημα. Ενώ στη γέννα-και-δοκίμαζε δίναμε τιμές σε *όλες* τις μεταβλητές και στη συνέχεια ελέγχαμε αν παραβιάζονταν οι περιορισμοί, στην οπισθοδρόμηση δίνουμε τιμή σε *μία* μεταβλητή και έπειτα ελέγχουμε αν ισχύουν οι περιορισμοί για τις ήδη δεσμευμένες μεταβλητές. Μια μεταβλητή ονομάζεται *δεσμευμένη* (bound ή singleton) όταν της έχει ανατεθεί τιμή (δηλαδή όταν το πεδίο τιμών της έχει γίνει μονομελές), αλλιώς είναι *ελεύθερη* (unbound).

Ο αλγόριθμος προβλέπει ότι αν κατά την ανάθεση τιμής σε μία μεταβλητή βρεθεί ότι παραβιάζεται κάποιος περιορισμός (για τις δεσμευμένες μεταβλητές πάντα), τότε επιλέγεται μία άλλη τιμή από το πεδίο τιμών της. Αν έχουμε δοκιμάσει όλες τις τιμές του πεδίου της μεταβλητής, οπισθοδρομούμε, δηλαδή δοκιμάζουμε την επόμενη τιμή της προηγούμενης μεταβλητής που δεσμεύθηκε. Αν αποτύχουν όλες οι αναθέσεις και για αυτήν, τότε οπισθοδρομούμε στην προπροηγούμενη κ.ο.κ. Δέσμευση όλων των μεταβλητών, με ικανοποίηση των περιορισμών, σημαίνει επιτυχία, ενώ αντίθετα η ανάγκη να οπισθοδρομήσουμε πίσω από την πρώτη μεταβλητή ισοδυναμεί με αποτυχία. Στον ψευδοκώδικα που ακολουθεί η οπισθοδρόμηση επιτυγχάνεται μέσω αναδρομής:

```
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
```

Κατά την εφαρμογή του αλγορίθμου στο παράδειγμα (§2.2) εμφανίζονται τα εξής βήματα:

V_A	V_B	V_Γ	V_Δ	Περιορισμός που παραβιάζεται	Οπισθοδρόμηση
1					
1	1			(2.5): $V_B < V_A$	
1	2			(2.5): $V_B < V_A$	
1	3			(2.5): $V_B < V_A$	
1	4			(2.5): $V_B < V_A$	✓
2					
2	1				
2	1	1		(2.2): $V_B \neq V_\Gamma$	
2	1	2		(2.1): $V_A \neq V_\Gamma$	
2	1	3			
2	1	3	1	(2.3): $V_\Delta < V_B$	
2	1	3	2	(2.3): $V_\Delta < V_B$	
2	1	3	3	(2.3): $V_\Delta < V_B$	
2	1	3	4	(2.3): $V_\Delta < V_B$	✓
2	1	4			

⋮

Αν και εκ πρώτης όψεως φαίνεται ότι ο αλγόριθμος έχει μικρότερη πολυπλοκότητα από τον γέννα-και-δοκίμαζε, εντούτοις για τη χειρότερη περίπτωση έχουμε ίδιες πολυπλοκότητες! Έστω το πρόβλημα με $V = \{V_1, V_2, \ldots, V_n\}$, $D = \{D_1, D_2, \ldots, D_n\}$ και μοναδικό περιορισμό $V_1 = V_n$. Δίνεται επίσης $D_1 = \{a\}$ και $D_n = \{b\}$, με $a \neq b$. Κατά την εφαρμογή της μεθόδου της οπισθοδρόμησης, θα χρειαστεί να εξαντληθούν όλοι οι συνδυασμοί των μεταβλητών $V_2, V_3, \ldots, V_{n-1}$, μέχρι να γίνει αντιληπτό ότι δεν υπάρχει λύση, όπως ακριβώς θα συνέβαινε και στον γέννα-και-δοκίμαζε.

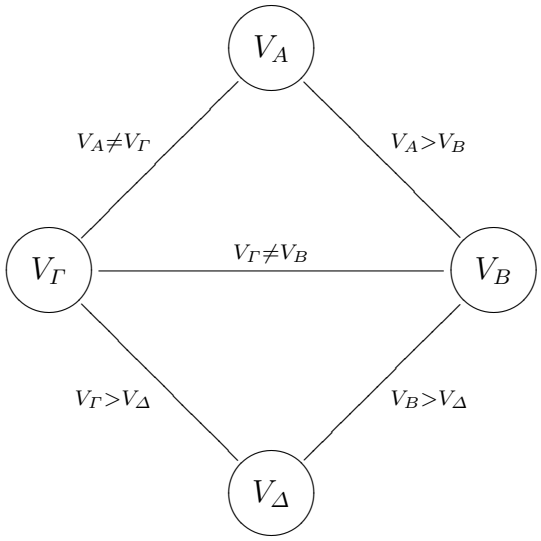
Οι δύο αυτές μέθοδοι πάσχουν λοιπόν, γιατί πολύ απλά είναι *οπισθοθεωρητικές* (retrospective), που σημαίνει ότι εκμεταλλεύονται τους περιορισμούς μόνο για τον έλεγχο των ήδη δεσμευμένων μεταβλητών. Και αυτός ο έλεγχος γίνεται a posteriori, μετά την ανάθεση δηλαδή. Τα πεδία των ελεύθερων μεταβλητών δεν περιορίζονται πριν αυτές δεσμευτούν, ώστε να αποκλειστούν κάποιες τιμές που θα οδηγήσουν ούτως ή άλλως σε αποτυχία και να έχουμε έτσι μια μείωση του χώρου αναζήτησης.

2.3.3 Διάδοση περιορισμών

Οι *εμπροσθοθεωρητικές* (prospective) μέθοδοι —εν αντιθέσει με τις οπισθοθεωρητικές— μετά από κάθε πράξη που γίνεται επί των μεταβλητών, δεν ελέγχουν μόνο τις δεσμευμένες μεταβλητές, αλλά διαγράφουν από τα πεδία τιμών των ελεύθερων μεταβλητών τις τιμές εκείνες που δεν είναι συνεπείς με τους περιορισμούς. Έχουμε δηλαδή ένα «κλάδεμα» του δένδρου αναζήτησης, με στόχο να μην ανατεθούν στο μέλλον λανθασμένες τιμές (a priori pruning). Πριν προχωρήσουμε στην παρουσίαση των αλγορίθμων, θα εισαγάγουμε κάποιες καινούργιες έννοιες.

Το *δίκτυο περιορισμών* (constraint network) είναι ένας γράφος με κόμβους που αναπαριστούν τις μεταβλητές. Τους κόμβους συνδέουν ακμές που αναπαριστούν τους περιορισμούς. Για το παράδειγμα (§2.2) έχουμε το Σχήμα 2.1.

Μια ακμή ενώνει δύο κόμβους-μεταβλητές και επομένως αναπαριστά έναν δυαδικό περιορισμό. Οι μοναδιαίοι περιορισμοί δεν είναι ανάγκη να αναπαρασταθούν, καθώς μπορούν να ικανοποιηθούν σε ένα προεπεξεργαστικό στάδιο, με διαγραφή κάποιων τιμών από τα πεδία των μεταβλητών. Αν είχαμε π.χ. τη μεταβλητή V_i με πεδίο τιμών $D_i = \{1, 2, 3, 4, 5\}$ και τον μοναδιαίο περιορισμό $V_i < 3$, τότε απλά θα θέταμε σαν πεδίο τιμών της V_i το $D_i^a = \{1, 2, \not{3}, \not{4}, \not{5}\} = \{1, 2\}$, από την αρχή. Τι γίνεται όμως με την αναπαράσταση περιορισμών ανώτερης τάξης στο δίκτυο περιορισμών; Δεν υπάρχει λόγος να ασχοληθούμε και με αυτήν, αφού αποδεικνύεται ότι:



Σχήμα 2.1: Δίκτυο περιορισμών

Κάθε πρόβλημα ικανοποίησης περιορισμών μπορεί να μετασχηματιστεί σε ένα ισοδύναμο πρόβλημα, που περιέχει μόνο δυαδικούς περιορισμούς. Ο μετασχηματισμός αυτός λέγεται δυαδικοποίηση *(binarization).*

Απόδειξη: Έστω ότι έχουμε ένα πρόβλημα P, με έναν περιορισμό ανώτερης τάξης C_i, ο οποίος αφορά τις μεταβλητές $V_{i_1}, V_{i_2}, \ldots, V_{i_q}$. Φτιάχνουμε ένα πρόβλημα P' ίδιο με το P, αλλά χωρίς τον C_i. Εισάγουμε στο P' μια νέα μεταβλητή V_0 με πεδίο τιμών τα στοιχεία του συνόλου $D_{i_1} \times D_{i_2} \times \cdots \times D_{i_q}$ που ικανοποιούν τον C_i. Επιπλέον εισάγουμε τους εξής (δυαδικούς) περιορισμούς, με δεδομένο ότι $V_0 = (x_1, x_2, \ldots, x_q)$:

$$\begin{aligned} V_{i_1} &= x_1 \\ V_{i_2} &= x_2 \\ &\vdots \\ V_{i_q} &= x_q \end{aligned}$$

Επαναλαμβάνουμε τη διαδικασία για κάθε περιορισμό ανώτερης τάξης του P. Τελικά το δυαδικό P' είναι ισοδύναμο με το P. □

Με τη δυαδικοποίηση, άσχετα από το γεγονός ότι δεν χρησιμοποιείται συχνά στις υλοποιήσεις των προβλημάτων σε υπολογιστή, έχουμε τη δυνατότητα σε θεωρητικό επίπεδο, να ασχοληθούμε μόνο με τα δυαδικά προβλήματα ικανοποίησης περιορισμών, δηλαδή με το δίκτυο περιορισμών των προβλημάτων.

Το δίκτυο περιορισμών είναι το μέσο στο οποίο θα διαδίδονται οι περιορισμοί (propagation). Οι αλγόριθμοι που θα δούμε για αυτό το δίκτυο-γράφο, επιχειρούν να τον φέρουν σε μια κατάσταση *συνέπειας ακμών* (arc consistency – AC). Μια ακμή (V_i, V_j) είναι συνεπής, αν για κάθε $x \in D_i$, υπάρχει $y \in D_j$, τέτοιο ώστε το ζεύγος τιμών (x, y) να μην παραβιάζει τον περιορισμό για την ακμή αυτή. Όταν μια ακμή (V_i, V_j) είναι συνεπής, τότε *δεν* ισχύει απαραίτητα ότι και η (V_j, V_i) είναι συνεπής, αφού η συνέπεια είναι κατευθυνόμενη. Π.χ. έστω $V_1 \in \{1, 2\}$ και $V_2 \in \{1, 2, 3\}$ με $V_1 \geq V_2$. Ισχύει ότι η (V_1, V_2) συνεπής και η (V_2, V_1) ασυνεπής, αφού δεν υπάρχει $y \in D_1$, με $y \geq 3$.

Με τον παρακάτω αλγόριθμο μπορούμε να εξασφαλίσουμε τη συνέπεια κάθε ακμής (V_i, V_j), αφαιρώντας τις τιμές του D_i που δεν ικανοποιούν τον περιορισμό για τις δύο μεταβλητές. (Π.χ. για την παραπάνω ακμή (V_2, V_1), αφαιρεί το 3 από το D_2.)

```
function REVISE(V_i, V_j)
    del ← false
    for each x in D_i do
        if there is no y ∈ D_j, with (x, y) not violating the constraint then
            delete x from D_i
            del ← true
        end if
    end for
    return del
end function
```

Αυτή η διαδικασία καλείται από έναν *αλγόριθμο συνέπειας ακμών* για να γίνουν όλες οι ακμές ενός γράφου συνεπείς. Τέτοιοι αλγόριθμοι είναι οι AC-1, AC-2, AC-3, AC-4 κ.λπ. οι οποίοι είναι παραλλαγές και βελτιώσεις της βασικής ιδέας:

> Για κάθε ακμή, αφαίρεσε από τα πεδία των μεταβλητών τις τιμές εκείνες που δεν ικανοποιούν τον αντίστοιχο περιορισμό.

Με αυτόν τον τρόπο αφαιρούνται από νωρίς εκείνες οι τιμές που δεν μπορούν να συμμετέχουν στην τελική λύση.

---

procedure AC-1(G)
  $Q \leftarrow \{(V_i, V_j) \mid (V_i, V_j) \in \text{arcs}(G)\}$
  repeat
    changed $\leftarrow$ false
    for each $(V_i, V_j) \in Q$ **do**
      changed $\leftarrow$ *changed* or REVISE(V_i, V_j)
    end for
  until not *changed*
end procedure

Παρατηρούμε ότι ο αλγόριθμος μετά από κάθε αλλαγή στον γράφο, ελέγχει ξανά τις ακμές του για να διαπιστωθεί αν είναι συνεπείς. Και αυτό γιατί μπορεί να συμβεί το φαινόμενο του ντόμινο: να αφαιρεθεί μια τιμή του πεδίου μιας μεταβλητής V_1 για γίνει συνεπής μια ακμή (V_1, V_2) και αυτό να προκαλέσει ασυνέπεια σε ακμές τύπου (V_x, V_1). Αν προσέξουμε τον AC-1 όμως, θα δούμε ότι δεν θα ελέγξει μόνο τις ακμές τύπου (V_x, V_1), αλλά όλες. Ο AC-3 έρχεται να σταματήσει αυτήν τη σπατάλη στους ελέγχους:

---

procedure AC-3(G)
  $Q \leftarrow \{(V_i, V_j) \mid (V_i, V_j) \in \text{arcs}(G)\}$
  while $Q \neq \emptyset$ **do**
    select and delete (V_k, V_m) from Q
    if REVISE(V_k, V_m) **then**
      $Q \leftarrow Q \cup \{(V_i, V_k) \mid (V_i, V_k) \in \text{arcs}(G), i \neq m\}$
    end if
  end while
end procedure

Για να δούμε ποια είναι η *φιλοσοφία* πίσω από αυτούς τους αλγορίθμους, θα προσπαθήσουμε να φέρουμε σε συνέπεια τις ακμές για το δίκτυο περιορισμών του παραδείγματος (§2.2):

	D_A	D_B	D_Γ	D_Δ	Περιορισμός
1	$\{\not{1}, 2, 3, 4\}$	$\{1, 2, 3, \not{4}\}$	$\{1, 2, 3, 4\}$	$\{1, 2, 3, 4\}$	$V_A > V_B$
2	$\{2, 3, 4\}$	$\{\not{1}, 2, 3\}$	$\{1, 2, 3, 4\}$	$\{1, 2, \not{3}, \not{4}\}$	$V_B > V_\Delta$
3	$\{2, 3, 4\}$	$\{2, 3\}$	$\{\not{1}, 2, 3, 4\}$	$\{1, 2\}$	$V_\Gamma > V_\Delta$
4	$\{\not{2}, 3, 4\}$	$\{2, 3\}$	$\{2, 3, 4\}$	$\{1, 2\}$	$V_A > V_B$

Στην $1^η$ γραμμή ασχολούμαστε με τον περιορισμό $V_A > V_B$: Η τιμή 1 αφαιρείται από το D_A, αφού δεν υπάρχει $y \in D_B$ με $1 > y$. Μετά από αυτό η ακμή (V_A, V_B) είναι συνεπής, δεν ισχύει όμως ακόμα το ίδιο και για τη (V_B, V_A), αφού δεν υπάρχει $y \in D_A$,[1] με $y > 4$. Συνεπώς πρέπει να αφαιρεθεί επίσης η τιμή 4 από το D_B. Έχοντας πετύχει τη συνέπεια των ακμών (V_A, V_B) και (V_B, V_A), εξασφαλίσαμε ότι η σχέση που συνδέει τις V_A και V_B, ισχύει.

Στη $2^η$ και στην $3^η$ γραμμή επαναλαμβάνουμε την ίδια διαδικασία για τους περιορισμούς $V_B > V_\Delta$ και $V_\Gamma > V_\Delta$ αντίστοιχα. Με το τέλος της $3^{ης}$ γραμμής ολοκληρώνεται το πρώτο «πέρασμα» από κάθε υπαρκτή ακμή. Βέβαια υπήρξαν κάποιες αλλαγές στα πεδία, οπότε όπως δηλώνεται και στον AC-1,[2] οι ακμές πρέπει να ελεγχθούν ξανά ως προς τη συνέπειά τους. Ό,τι απομένει να γίνει, διεκπεραιώνεται στην $4^η$ γραμμή με το να έρθει σε κατάσταση συνέπειας η ακμή (V_A, V_B).

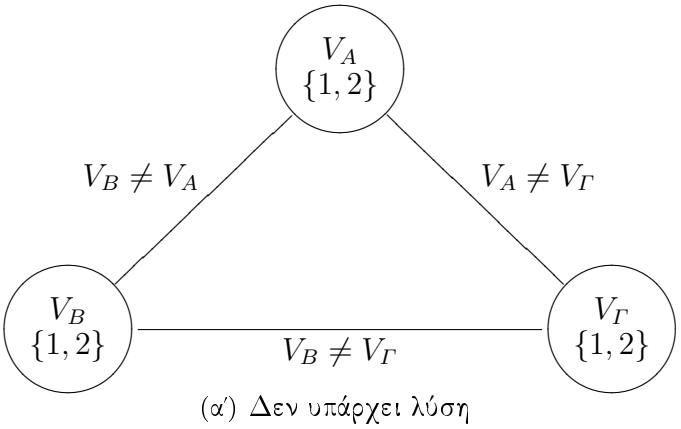
Εύλογα θα αναρωτηθεί κανείς: «Μα είναι λύση αυτό που εμφανίζεται στην $4^η$ γραμμή;» Η απάντηση είναι όχι, οι αλγόριθμοι ελέγχου συνέπειας δεν υπόσχονται ότι θα βρουν λύση! Η χρησιμότητά τους έγκειται στο ότι μειώνουν τον χώρο αναζήτησης. Πράγματι, στο παραπάνω παράδειγμα, το μέγεθος του χώρου αναζήτησης από $4 \times 4 \times 4 \times 4 = 256$ που ήταν, έγινε μόλις $2 \times 2 \times 3 \times 2 = 24$. Τονίζουμε ότι η συνέπεια των ακμών ενός γράφου δεν φανερώνει ότι θα έχουμε μία ή περισσότερες λύσεις, ούτε καν ότι θα έχουμε λύση. Για του λόγου το αληθές παρουσιάζονται οι τρεις γράφοι του Σχήματος 2.2. Κάποιες φορές όμως μπορούμε να αποφανθούμε για τη λύση μετά την εφαρμογή ενός αλγορίθμου AC: 1) όταν κάποια μεταβλητή έχει κενό πεδίο, οπότε δεν υπάρχει λύση και το πρόβλημα ικανοποίησης περιορισμών ονομάζεται *ασυνεπές* (inconsistent) και 2) όταν όλες οι μεταβλητές έχουν μονομελές πεδίο, οπότε οι τιμές των μεταβλητών αποτελούν λύση.

Το άνω φράγμα της πολυπλοκότητας ενός αλγορίθμου συνέπειας ακμών είναι ed^2, όπου e ο αριθμός των περιορισμών και d ο μέγιστος αριθμός των στοιχείων ενός πεδίου τιμών του προβλήματος. Οι αλγόριθμοι της οικογένειας AC διαφέρουν μεταξύ τους στο θέμα της απόδοσης. Οι αποδοτικότεροι αλγόριθμοι πάντως απαιτούν περισσότερη μνήμη για να εκτελεστούν.

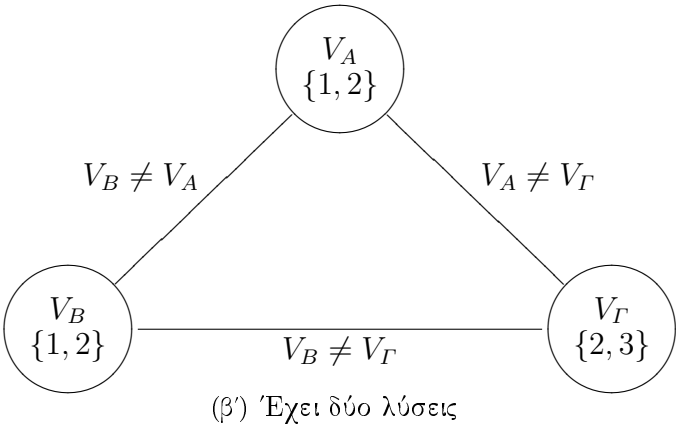
Μετά την εφαρμογή ενός αλγορίθμου AC λοιπόν, για να προχωρήσουμε στην επίλυση του προβλήματος, αναγκαζόμαστε εκ των πραγμάτων να χρησιμοποιήσουμε μία κλασική μέθοδο αναζήτησης όπως η πρώτα-κατά-βάθος (depth-first-search – DFS). Η πρώτα-κατά-πλάτος και ανάλογες μέθοδοι που δεν εκμε-

[1] Το D_A τώρα ισούται με $\{2, 3, 4\}$.

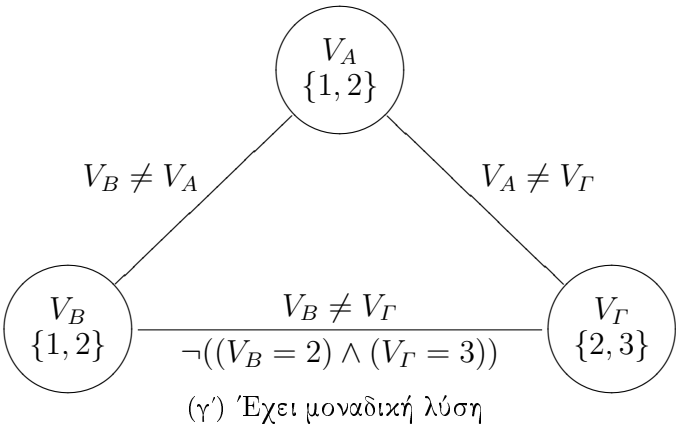
[2] Στον αλγόριθμο AC-1 έχουμε *changed* = true (σελ. 30).



(α′) Δεν υπάρχει λύση



(β′) Έχει δύο λύσεις



(γ′) Έχει μοναδική λύση

Σχήμα 2.2: Τρεις γράφοι με συνέπεια ως προς τις ακμές

ταλλεύονται το πεπερασμένο και σταθερό (ίσο με το πλήθος των μεταβλητών) βάθος του δένδρου αναζήτησης δεν ενδείκνυνται.

Κεντρικό ρόλο στην αποτελεσματικότητα της αναζήτησης παίζουν τα ευριστικά. Η μέθοδος αναζήτησης «συμβουλεύεται» τον ευριστικό κανόνα για να αποφασίσει ποια μεταβλητή θα επιλέξει και ποια τιμή θα της ανατεθεί (με άλλα λόγια ποιος κόμβος του *δένδρου αναζήτησης* θα επιλεγεί). Ένα γενικό ευριστικό είναι *η αρχή της συντομότερης αποτυχίας* (first fail principle). Το ευριστικό αυτό προτείνει να ανατεθεί τιμή στη μεταβλητή με το μικρότερο πεδίο τιμών. Συνηθίζεται να συνδυάζονται περισσότερα του ενός ευριστικά. Π.χ. αν το παραπάνω ευριστικό συναντούσε δύο μεταβλητές με τον ίδιο αριθμό τιμών, δεν θα μπορούσε να αξιολογήσει ποια από τις δύο μεταβλητές είναι καλύτερο να επιλεγεί πρώτη. Σε αυτό το σημείο μπορούμε να ακολουθήσουμε έναν άλλο ευριστικό κανόνα που προτρέπει: διάλεξε τη μεταβλητή εκείνη, που εμπλέκεται σε μεγαλύτερο αριθμό περιορισμών *(most constraint principle)*.

Αν εφαρμόσουμε τα παραπάνω ευριστικά στην 4^{η} γραμμή του προηγούμενου πίνακα έχουμε:

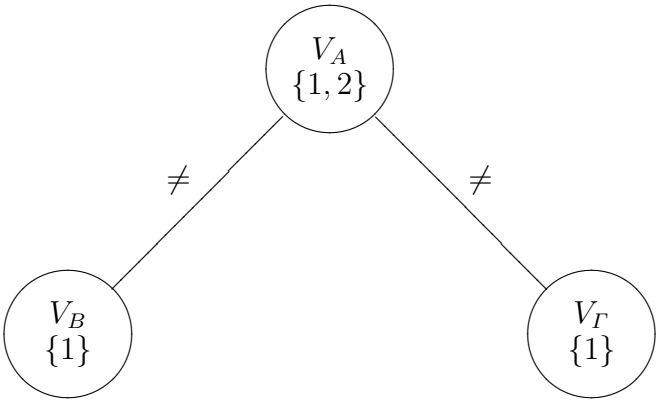
1. *first fail principle:* επιλογή των μεταβλητών V_A, V_B και V_Δ (με τα μικρότερα πεδία τιμών).

2. *most constraint principle:* από τις παραπάνω μεταβλητές, επιλογή της V_B, αφού εμπλέκεται σε περισσότερους (τρεις) περιορισμούς. (Αυτό είναι προφανές στο δίκτυο περιορισμών του Σχήματος 2.1, στο οποίο οι ακμές που συνδέονται με τη V_B είναι τρεις, έναντι δύο ακμών για τις V_A και V_Δ.)

k-συνέπεια

Ειπώθηκε ότι μετά την εφαρμογή ενός αλγορίθμου AC, αυτό που προκύπτει δεν είναι (πάντα) λύση. Για να έχουμε μια πιο συστηματική εικόνα για την κατάσταση ενός δικτύου περιορισμών ανά πάσα στιγμή, θα ορίσουμε την έννοια της K-συνέπειας. Ένας γράφος περιορισμών είναι *k-συνεπής* (k-consistent), αν και μόνο αν για κάθε συνδυασμό τιμών των πεδίων οποιωνδήποτε $k-1$ μεταβλητών, που ικανοποιεί τους μεταξύ τους περιορισμούς, υπάρχει τιμή στο πεδίο κάθε άλλης μεταβλητής, τέτοια ώστε να ικανοποιούνται ταυτόχρονα όλοι οι περιορισμοί που συνδέουν τις k μεταβλητές. Ένας γράφος είναι *ισχυρά k-συνεπής* (strongly k-consistent) αν για κάθε $\ell \leq k$ είναι ℓ-συνεπής.

Σε έναν ισχυρά 1-συνεπή γράφο είναι εξασφαλισμένο ότι θα τηρούνται οι «ατομικοί»-μοναδιαίοι περιορισμοί κάθε μεταβλητής, ενώ οι γράφοι που προκύπτουν από τους αλγορίθμους AC είναι ισχυρά 2-συνεπείς. Αν ένας γράφος n



Σχήμα 2.3: 3-συνεπής αλλά όχι 2-συνεπής γράφος

κόμβων/μεταβλητών είναι ισχυρά n-συνεπής, τότε οποιοσδήποτε συνδυασμός των τιμών των μεταβλητών αποτελεί λύση. Υπάρχουν αλγόριθμοι εξασφάλισης ισχυρής k-συνέπειας, με $k > 2$, όμως το κόστος τους είναι εκθετικό και έτσι προτιμούμε τη χρήση μιας κλασικής μεθόδου αναζήτησης σε συνδυασμό με έναν αλγόριθμο AC.

Στο Σχήμα 2.3, ο γράφος δεν είναι 2-συνεπής. Αυτό μπορούμε να το δείξουμε βάσει του ορισμού, για $k = 2$. Παίρνουμε λοιπόν έναν συνδυασμό τιμών $k - 1$ μεταβλητών, δηλαδή εν προκειμένω μία τιμή μίας μεταβλητής, έστω της V_A. Διαλέγουμε την τιμή 1 της V_A (δεν χρειάζεται να ελέγξουμε αν η τιμή αυτή ικανοποιεί τους υπάρχοντες μοναδιαίους περιορισμούς για την V_A, αφού δεν υπάρχουν τέτοιοι). Όμως δεν υπάρχει τιμή στη V_B που να ικανοποιεί τον περιορισμό $V_A \neq V_B$, για $V_A = 1$. (Το ίδιο ισχύει και για τη V_Γ, αλλά αρκεί να το δείξουμε για μία μεταβλητή.)

Παρόλα αυτά ο γράφος είναι 3-συνεπής. Αν πάρουμε οποιονδήποτε συνδυασμό τιμών $k - 1 = 2$ μεταβλητών, που ικανοποιεί τους περιορισμούς ($1^η$ στήλη του Πίνακα 2.1), τότε υπάρχει τιμή, σε κάθε τρίτη μεταβλητή, που ικανοποιεί τους περιορισμούς ανάμεσα στις $k = 3$ μεταβλητές ($2^η$ στήλη του Πίνακα 2.1). Φυσικά ο γράφος δεν είναι ισχυρά 3-συνεπής, αφού δεν είναι 2-συνεπής.

$V_A = 2, \quad V_B = 1$	$V_\Gamma = 1$
$V_A = 2, \quad V_\Gamma = 1$	$V_B = 1$
$V_B = 1, \quad V_\Gamma = 1$	$V_A = 2$

Πίνακας 2.1: Απόδειξη 3-συνέπειας

Συνέπεια-ορίων

Η *συνέπεια-ορίων* (bounds-consistency) είναι μια πιο χαλαρή έννοια συνέπειας. Αφορά τα άκρα των πεδίων των μεταβλητών που εμπλέκονται σε έναν περιορισμό C_k. Για να εξασφαλιστεί η συνέπεια-ορίων σε έναν περιορισμό C_k, ελέγχεται αν για τα άκρα $\max\{D_i\}$ και $\min\{D_i\}$ κάθε μεταβλητής V_i που εμπλέκεται στον περιορισμό, υπάρχουν τιμές υποστήριξης στα πεδία των υπόλοιπων μεταβλητών, έτσι ώστε να ικανοποιείται ο περιορισμός.

Ας δούμε ένα παράδειγμα. Έστω οι μεταβλητές V_i και V_j, με $D_i = \{0, 3, 9\}$ και $D_j = \{0, 3, 4\}$, που συνδέονται με τον περιορισμό $V_i = 3V_j$. Για να υπάρχει συνέπεια ακμών, θα πρέπει να αφαιρεθεί το 3 από το D_i (αφού $1 \notin D_j$) και το 4 από το D_j (αφού $3 \cdot 4 = 12 \notin D_i$). Ωστόσο για να έχουμε *συνέπεια-ορίων,* θα πρέπει να αφαιρεθεί μόνο το 4 από το D_j, καθώς αποτελεί οριακή (μέγιστη) τιμή του πεδίου αυτού. Το 3 του D_i, δεν υπάρχει πρόβλημα να παραμείνει.

Συνήθως, η συνέπεια-ορίων εφαρμόζεται όταν έχουμε να κάνουμε με περιορισμούς στους οποίους η επιβολή συνέπειας ακμών είναι εξαιρετικά χρονοβόρα — π.χ. στον περιορισμό $V_1 = \sum_{i=2}^{10} V_i$.

Κεφάλαιο 3

Ο επιλυτής Naxos Solver

Μπορεί κανείς να δει τα όρια, αν τα ξεπεράσει.
— Herbert Simon

Στα πλαίσια αυτής της εργασίας σχεδιάστηκε και υλοποιήθηκε ένας επιλυτής προβλημάτων ικανοποίησης περιορισμών, που για συντομία θα τον αναφέρουμε απλά ως *επιλυτή* (solver). Με άλλα λόγια, κατασκευάστηκε μια βιβλιοθήκη που επιτρέπει τη λύση προβλημάτων, μέσω προγραμματισμού με περιορισμούς. Πήρε το όνομα Naxos Solver, από το νησί[1] στο οποίο άρχισε να χτίζεται — και στο οποίο έφθασε σε ένα τελικό, ώριμο στάδιο.

Λόγω της πληθώρας των θεωρητικών εργασιών πάνω στον προγραμματισμό με περιορισμούς, τέθηκαν πολλά διλήμματα και είχαμε πολλές επιλογές να κάνουμε. Ο στόχος μας ήταν να «παντρέψουμε» τη θεωρία με την πράξη, έτσι ώστε να προκύψει ένας επιλυτής με χαμηλές απαιτήσεις σε χρόνο και μνήμη, στον οποίο θα είναι εύκολο να διατυπωθεί κάθε λογής πρόβλημα, το οποίο δύναται να επιλυθεί μέσω προγραμματισμού με περιορισμούς.

Σε αυτό το κεφάλαιο, μετά από μία σύντομη εισαγωγή, αναλύονται οι αλγόριθμοι που χρησιμοποιήθηκαν και αιτιολογούνται οι αποφάσεις που πάρθηκαν. Στο Παράρτημα Α΄ υπάρχει ένα εγχειρίδιο χρήσης του επιλυτή, καθώς και μερικά παραδείγματα χρήσης του, που επιδεικνύουν τη λειτουργικότητά του. Εξάλλου, ένα σοβαρότερο —και σίγουρα πολύ πιο «βαρύ»— παράδειγμα διατύπωσης και επίλυσης ενός προβλήματος με τον Naxos Solver, παρουσιάζεται στο επόμενο κεφάλαιο· αναφερόμαστε στο πρόβλημα κατάρτισης ωρολογίου προγράμματος, που αποτελεί μία σκληρή δοκιμή, σχετικά με τις δυνατότητες και τις ευκολίες που μπορεί να προσφέρει ο επιλυτής.

[1] Πιο συγκεκριμένα, αναφερόμαστε στο χωριό Κορωνίδα της Νάξου.

3.1 Κατηγορίες αλγορίθμων επιβολής συνέπειας

Αρχικά, έπρεπε να μελετήσουμε την οικογένεια αλγορίθμων επιβολής συνέπειας AC και να κάνουμε έναν από αυτόν, την «καρδιά» του επιλυτή.

Στο προηγούμενο κεφάλαιο παρουσιάστηκαν οι AC-1 και AC-3. Μαζί με τον AC-2 —ο οποίος υπάρχει περισσότερο για ιστορικούς λόγους— και τον καινούργιο AC-2001, αποτελούν την κατηγορία αλγορίθμων *προσανατολισμένων στις ακμές* (arc oriented ή coarse grained). Σε αυτούς τους αλγορίθμους, όταν γίνει κάποια αλλαγή στο πεδίο κάποιας μεταβλητής, τότε εξετάζονται οι ακμές, δηλαδή οι υπόλοιπες μεταβλητές, με τις οποίες συνδέεται η μεταβλητή αυτή, μήπως βρεθεί κάποια ασυνέπεια. Αν εξαιρέσουμε τον AC-2001, οι αλγόριθμοι αυτής της κατηγορίας έχουν μεγάλη χρονική πολυπλοκότητα, όμως οι απαιτήσεις τους σε μνήμη είναι οι απολύτως αναμενόμενες (μικρές).

Η δεύτερη μεγάλη κατηγορία αλγορίθμων, είναι *προσανατολισμένη στις τιμές των μεταβλητών* (fine grained). Σε αυτήν περιλαμβάνονται μεταξύ των άλλων, οι αλγόριθμοι AC-4, AC-6 και AC-7 [21]. Ενώ οι αλγόριθμοι αυτοί έχουν βέλτιστη πολυπλοκότητα, οι απαιτήσεις τους σε μνήμη είναι απαγορευτικές για ένα πρακτικό σύστημα που λύνει ποικίλων ειδών προβλήματα, όπως ο επιλυτής. Οι υψηλές απαιτήσεις σε μνήμη οφείλονται στο ότι για κάθε τιμή του πεδίου κάθε μεταβλητής, αποθηκεύονται πληροφορίες που έχουν να κάνουν με τις *τιμές υποστήριξης* (support values), οι οποίες βρίσκονται στα πεδία των συνδεδεμένων με αυτήν μεταβλητών.

Ο AC-2001, ενώ κατατάσσεται στους coarse grained αλγορίθμους, έχει βέλτιστη χρονική πολυπλοκότητα και πολυπλοκότητα μνήμης, ανάλογη με αυτήν των fine grained αλγορίθμων [22].

Τέλος υπάρχει ο AC-5, ο οποίος δεν κατατάσσεται σε καμία από τις παραπάνω κατηγορίες, αν και θα μπορούσε να ανήκει και στις δύο. Ίσως η προηγούμενη πρόταση να φαίνεται διφορούμενη, ωστόσο γίνεται κατανοητή αν σκεφτούμε ότι ο AC-5 είναι παραμετρικός· μπορούμε για κάθε περιορισμό, να δώσουμε τις δικές μας «οδηγίες» για την αντιμετώπισή του, δηλαδή να φτιάξουμε δικές μας υπορουτίνες, οι οποίες θα καλούνται από τον κύριο αλγόριθμο AC-5.

Πράγματι, με τον AC-5 είναι εύκολο να προσομοιώσουμε π.χ. τον AC-3 και τον AC-4. Είναι δυνατόν να φτιάξουμε ακόμη και υβρίδια. Η ευελιξία του αυτή, καθώς και η αποτελεσματική —βέλτιστης πολυπλοκότητας— προσαρμογή του σε απλούς, «καθημερινούς» περιορισμούς, κάνουν τον AC-5 ιδανικό για να χρησιμοποιηθεί σε έναν επιλυτή γενικής χρήσης, όπως ο Naxos.

3.2 Ο αλγόριθμος AC-5

Στον Naxos Solver χρησιμοποιήθηκε μια παραλλαγή του AC-5. Παρακάτω παρουσιάζεται ο αρχικός ορισμός του AC-5 [23] και στη συνέχεια θα αναφερθεί το πώς προσαρμόστηκε για τον επιλυτή.[2]

```
1:  procedure AC-5
2:      Q ← ∅
3:      for each (i, j) ∈ arcs(G) do
4:          ArcCons(i, j, Δ)
5:          Q ← Q ∪ {((k, i), w) | (k, i) ∈ arcs(G), k ≠ j}
6:          D_i ← D_i − Δ
7:      end for
8:      while Q ≠ ∅ do
9:          Pick and remove an ((i, j), w) out of Q
10:         LocalArcCons(i, j, w, Δ)
11:         Q ← Q ∪ {((k, i), w) | (k, i) ∈ arcs(G), k ≠ j}
12:         D_i ← D_i − Δ
13:     end while
14: end procedure
```

Δεν έχουν οριστεί ακόμα οι υπορουτίνες ArcCons και LocalArcCons. Στην πραγματικότητα, δεν θα ορίσουμε τις ίδιες τις υπορουτίνες, αλλά θα δηλώσουμε μόνο τις προδιαγραφές τους.

Η ArcCons(i, j, Δ), θα πρέπει να φέρνει σε συνέπεια την ακμή (i, j). Κατά τη διαδικασία αυτή, όποιες τιμές αφαιρεθούν από το πεδίο D_i, θα πρέπει να μπουν στο σύνολο Δ — το οποίο κατά την έναρξη της εκτέλεσης της υπορουτίνας είναι κενό.

Η LocalArcCons(i, j, w, Δ) θα πρέπει να φέρνει σε συνέπεια την ακμή (i, j) —αφαιρώντας τιμές από το D_i και βάζοντάς τις στο κενό αρχικά σύνολο Δ— *δεδομένης της αφαίρεσης της τιμής* w *από το πεδίο* D_j. Το σημείο που θα πρέπει να προσεχθεί εδώ, είναι ότι η LocalArcCons δεν απαιτείται να εξασφαλίσει τη συνέπεια της ακμής (i, j). Αυτό που πρέπει να κάνει είναι να αφαιρέσει τις τιμές εκείνες του D_i, οι οποίες είχαν μοναδικό στήριγμα στη μεταβλητή j, την τιμή w.

Είναι δυνατόν, να ορίσουμε τη LocalArcCons έτσι:

[2] Ήδη κάναμε μία προσαρμογή, βάζοντας στις γραμμές 5 και 11, τη συνθήκη $k \neq j$. Ανάλογη συνθήκη υπάρχει στον ορισμό του AC-3.

procedure LocalArcCons(i, j, w, Δ)
  ArcCons(i, j, Δ)
end procedure

Αν χρησιμοποιήσουμε τον παραπάνω ορισμό, κατ' ουσίαν αγνοούμε το w και τελικά ο AC-5 ταυτίζεται με τον AC-3.

Πράγματι, αν έχουμε για μία ακμή (i, j) «αφηρημένους» περιορισμούς του τύπου $C_{ij} = \{(0, 33), (35, 33), (9, 999), (3, 5)\}$, αυτό το w δεν θα είναι ιδιαίτερα χρήσιμο. Για τέτοιες περιπτώσεις περιορισμών, ο ιδανικός αλγόριθμος θα ήταν ο AC-2001.

Όμως οι περιορισμοί που τίθενται από έναν προγραμματιστή που χρησιμοποιεί μια βιβλιοθήκη επίλυσης προβλημάτων ικανοποίησης περιορισμών, όπως ο Naxos Solver, είναι «σαφέστεροι». Π.χ. είναι της μορφής $C_{ij} = \{(x, y) \mid x = y, x \in D_i, y \in D_j\}$. Συνηθισμένοι περιορισμοί που διατυπώνονται σε έναν επιλυτή, είναι οι *συναρτησιακοί* (functional), όπως μία σχέση $V_i = 4V_j + 7$, μεταξύ των μεταβλητών V_i και V_j, *αντι-συναρτησιακοί* (anti-functional), όπως η ανισότητα $V_i \neq V_j$ και *μονότονοι* (monotonic) όπως το $V_i < 8V_j$.

Για τις παραπάνω κατηγορίες περιορισμών —καθώς και για αρκετές άλλες— μπορούμε να ορίσουμε την LocalArcCons κατά τέτοιο τρόπο, ώστε να έχει σταθερή πολυπλοκότητα. Π.χ. για τον περιορισμό $V_i = 4V_j + 7$, αν κληθεί η LocalArcCons$(i, j, 3, \Delta)$ —δηλαδή αν έχουμε $w = 3$— μπορούμε να δούμε σε $O(1)$ αν $(4 \cdot 3 + 7) \in D_i$, ενώ η αντίστοιχη κλήση ArcCons(i, j, Δ), θα έπαιρνε χρόνο $O(|D_i|)$. Και ενώ για όλα τα παραπάνω οι απαιτήσεις μνήμης είναι χαμηλές —ανάλογες με αυτές του AC-3—, η πολυπλοκότητα που επιτυγχάνουμε για τις προαναφερθείσες κατηγορίες περιορισμών είναι η βέλτιστη (ίδια με αυτήν των fine grained αλγορίθμων και του AC-2001).

3.3 Προσαρμογή του AC-5

Για τις ανάγκες του Naxos Solver, χρησιμοποιήθηκε μια παραλλαγή του AC-5:

```
1: procedure AC-5-Naxos
2:     Q ← ∅
3:     arcs(G) ← ∅
4:     while read (i, j) do
5:         arcs(G) ← arcs(G) ∪ {(i, j)}
6:         ArcCons(i, j, Δ)
7:         Q ← Q ∪ {(i, (i, j), w) | w ∈ Δ}
8:         D_i ← D_i − Δ
9:     end while
10:    while Q ≠ ∅ do
11:        Pick and remove an (i, (i, j), w) out of Q
12:        for each (k, i) ∈ arcs(G), with k ≠ j do
13:            LocalArcCons(k, i, w, Δ)
14:            Q ← Q ∪ {(k, (k, i), w) | w ∈ Δ}
15:            D_k ← D_k − Δ
16:        end for
17:    end while
18: end procedure
```

Μια πρώτη παρατήρηση είναι ότι ο γράφος G δεν είναι αρχικά σχηματισμένος. Αυτό προσφέρει μία μεγαλύτερη ευελιξία: Ο αλγόριθμος μπορεί να εισχωρήσει στη διαδικασία δήλωσης των μεταβλητών και των περιορισμών — με το που δηλώνεται ένας περιορισμός, ενσωματώνεται αυτόματα στο δίκτυο περιορισμών.[3]

Πάντως, το σημαντικότερο νέο στοιχείο του αλγορίθμου, έγκειται στο ότι το Q περιέχει άλλου τύπου στοιχεία. Ενώ στον AC-5 το Q αποτελούνταν από στοιχεία $((i, j), w)$, που σήμαινε ότι πρέπει να ελεγχθεί η συνέπεια της ακμής (i, j), δεδομένης της απουσίας του w από D_j, στον AC-5-Naxos το Q έχει στοιχεία $(i, (i, j), w)$, που σημαίνει ότι θα πρέπει να ελεγχθεί η συνέπεια όλων των ακμών, που κατευθύνονται προς τη μεταβλητή i, δεδομένης πάλι της απουσίας του w από D_j [24, 25]. Η ακμή (j, i) δεν χρειάζεται να ελεγχθεί. Ο νέος τύπος λοιπόν των στοιχείων του Q, περιέχει περισσότερες πληροφορίες.[4]

[3] Γίνεται δηλαδή αυτόματα "post", σύμφωνα και με την ορολογία του Ilog Solver. Ο τελευταίος επιλυτής απαιτεί από τον προγραμματιστή ο οποίος τον χρησιμοποιεί, να δηλώνει ρητά το πότε θα γίνονται post κάποιοι συγκεκριμένοι περιορισμοί. Στον Naxos Solver τα πράγματα είναι πιο απλά.

[4] Ο παράγοντας εξοικονόμησης μνήμης ισούται με τον μέσο όρο των βαθμών των κόμβων του G.

Περισσότερες τροποποιήσεις

Στον αλγόριθμο που χρησιμοποιεί πραγματικά ο επιλυτής, έχουμε αναθέσει περισσότερες «αρμοδιότητες» στις υπορουτίνες ArcCons και LocalArcCons: σε αυτές ενσωματώνονται οι γραμμές 7–8 και 14–15 αντίστοιχα του AC-5-Naxos. Δηλαδή δεν υπάρχει το ενδιάμεσο σύνολο Δ και αντί αυτού, στις δύο υπορουτίνες περνάμε σαν παράμετρο το ίδιο το Q. Επιπλέον, οι δύο υπορουτίνες είναι υπεύθυνες για την αφαίρεση ασυνεπών τιμών από τις μεταβλητές. Για το σκοπό αυτό χρησιμοποιούν μία μέθοδο remove. Αν η remove κληθεί να αφαιρέσει και την τελευταία τιμή ενός πεδίου μιας μεταβλητής, τότε εγείρεται μια σημαία, που δείχνει ότι το δίκτυο περιορισμών δεν είναι 2-συνεπές.

Μία άλλη παραδοχή που έγινε για απλούστευση, είναι ότι η ArcCons για μία ακμή (i, j), κάνει και τη δουλειά της ArcCons για την ακμή (j, i). Η συμμετρικότητα αυτή *δεν* ισχύει για την LocalArcCons, γιατί το όρισμα w που παίρνει, αφορά τη μία από τις δύο ακμές.

Επιπλέον —χωρίς να χρησιμοποιούμε αυστηρή μαθηματική γλώσσα— επιτρέπονται πολλαπλές ακμές (i, j), αφού το $\text{arcs}(G)$ και το Q, έχουν υλοποιηθεί σαν λίστες και όχι σαν σύνολα. Με αυτήν την παραδοχή διατηρούμε την απλότητα των περιορισμών. Π.χ. οι περιορισμοί $3V_i = V_j + 1$ και $V_i = 9V_j$, δεν χρειάζεται να συγχωνευθούν σε έναν περιορισμό.

Τέλος, επιτράπηκαν οι *καθολικοί περιορισμοί* (global constraints), οι οποίοι έχουν να κάνουν με περισσότερες από δύο μεταβλητές. Η απόφαση αυτή ήταν επιβεβλημένη λόγω της συχνής χρήσης καθολικών περιορισμών, όπως π.χ. του «όλες οι μεταβλητές ενός συνόλου να είναι διαφορετικές μεταξύ τους». Έτσι, στον τελικό αλγόριθμο AC-5-Naxos-Final που παρουσιάζεται παρακάτω, οι ακμές (i, j) έχουν αντικατασταθεί με περιορισμούς C_k. Οι υπορουτίνες ArcCons και LocalArcCons για τους καθολικούς περιορισμούς, επιτρέπεται να βάζουν στο Q στοιχεία του στυλ (i, NIL, w), που υποδεικνύουν ότι θα πρέπει να εξεταστεί, εκτός των άλλων, και ο (καθολικός) περιορισμός που προκάλεσε την αφαίρεση του w από το D_i.

```
1: procedure AC-5-Naxos-Final
2:     Q ← ∅
3:     C ← ∅
4:     while read C_k do
5:         C ← C ∪ {C_k}
6:         C_k.ArcCons(Q)
7:     end while
8:     while Q ≠ ∅ do
9:         Pick and remove an (i, C_s, w) out of Q
10:        for each C_k ∈ C that variable i is involved in, with C_k ≠ C_s do
11:            C_k.LocalArcCons(i, w, Q)
12:        end for
13:    end while
14: end procedure
```

Με C_k.ArcCons, αναφερόμαστε στην ArcCons που έχει σχεδιαστεί για τον περιορισμό C_k. Τα ίδια ισχύουν και για το C_k.LocalArcCons.

Αναφορικά με τη γραμμή 10 του αλγορίθμου, είναι εύκολο να υλοποιηθεί, αν προηγουμένως, για κάθε περιορισμένη μεταβλητή, έχουμε φτιάξει έναν πίνακα με τους περιορισμούς στους οποίους αυτή εμπλέκεται.

3.4 Συναρτήσεις ArcCons και LocalArcCons για γνωστούς περιορισμούς

Στον Naxos Solver υλοποιούνται οι πιο διαδεδομένοι περιορισμοί. Η ενσωμάτωσή τους στον AC-5, γίνεται με συγκεκριμένες για κάθε περιορισμό, συναρτήσεις ArcCons και LocalArcCons. Στη συνέχεια, θα οριστούν αυτές οι συναρτήσεις. Θα χρησιμοποιηθούν συμβολισμοί του παρακάτω πίνακα. Στη δεύτερη στήλη του, υπάρχει η επεξήγηση των συμβολισμών της πρώτης στήλης.[5]

V_i.min	$\min\{D_i\}$
V_i.max	$\max\{D_i\}$
V_i.value	Η μοναδική τιμή του D_i, εφόσον $\|D_i\| = 1$. Αν $\|D_i\| \neq 1$, ο συμβολισμός δεν ορίζεται.

[5] Αντί του συμβολισμού V_i.value, θα μπορούσαμε απλά να χρησιμοποιήσουμε το V_i.min ή V_i.max. Δεν το κάναμε, για να είναι οι αλγόριθμοι που ακολουθούν πιο ευανάγνωστοι.

V_i.next(x)	$\min\{y \mid\ y > x,\ y \in D_i \cup \{+\infty\}\}$
V_i.previous(x)	$\max\{y \mid\ y < x,\ y \in D_i \cup \{-\infty\}\}$
V_i.REMOVE(x)	REMOVE(V_i, x, Q)
VarFired	Το πρώτο όρισμα της LocalArcCons(i, w, Q), όπως χρησιμοποιείται στον παραπάνω αλγόριθμο AC-5-Naxos-Final. Αναφέρεται στη μεταβλητή από την οποία αφαιρέθηκε το w.

Ορίζουμε τη REMOVE ως εξής:

procedure REMOVE(V_i, x, Q)
  $D_i \leftarrow D_i - \{x\}$
end procedure

Επίσης, η REMOVE προσθέτει στο Q το κατάλληλο στοιχείο και σηκώνει μια «σημαία ασυνέπειας», αν επιχειρηθεί να αφαιρεθεί η τελευταία τιμή ενός πεδίου μεταβλητής.

Στο εξής, αντί για $V_i, V_j, \ldots$, θα χρησιμοποιήσουμε τα $X, Y, Z, \ldots$, για να αναφερόμαστε σε περιορισμένες μεταβλητές. Το C, θα συμβολίζει μία ακέραια σταθερά. Ακόμα, όταν δύο συμβολισμοί μεταβλητών (π.χ. οι X και VarFired) αναφέρονται στο ίδιο στιγμιότυπο μεταβλητής, τότε το δείχνουμε με τον τελεστή $\equiv$ (π.χ. $X \equiv$ VarFired). Αντίστοιχα ορίζεται και το $\not\equiv$. Τονίζεται ότι όταν αναφέρουμε ότι $X \equiv Y$, δεν εννοούμε ότι υπάρχουν δύο μεταβλητές X και Y με ίδια πεδία τιμών, αλλά ότι υπάρχει μία μεταβλητή την οποία συμβολίζουμε είτε με X, είτε με Y.

3.4.1 Ο περιορισμός $X < Y$

Έχουμε τους παρακάτω ορισμούς για τις δύο υπορουτίνες:

procedure ArcCons
  while X.max $\geq Y$.max **do**
    X.REMOVE(X.max)
  end while
  while X.min $\geq Y$.min **do**
    Y.REMOVE(Y.min)
  end while
end procedure

Ομοίως, ορίζεται ο περιορισμός $X \leq Y$. Η LocalArcCons ταυτίζεται με την ArcCons· εξάλλου, με οποιονδήποτε άλλον γνωστό τρόπο και να ορίζαμε την LocalArcCons, δεν θα κερδίζαμε κάτι σε πολυπλοκότητα.

Ένα θετικό χαρακτηριστικό της παραπάνω συνάρτησης, είναι ότι εστιάζει στα *άκρα* των πεδίων των μεταβλητών. Συνεπώς ο περιορισμός αυτός συμβαδίζει με το πνεύμα της *συνέπειας-ορίων* (bounds-consistency).

3.4.2 Ο περιορισμός $X = Y$

```
procedure ArcCons
    for each v ∈ D_X do
        if v ∉ D_Y then
            X.remove( v )
        end if
    end for
    for each v ∈ D_Y do
        if v ∉ D_X then
            Y.remove( v )
        end if
    end for
end procedure

procedure LocalArcCons(VarFired, w, Q)
    if X ≡ VarFired then
        if w ∈ D_Y then
            Y.remove( w )
        end if
    else                                          ▷ Y ≡ VarFired
        if w ∈ D_X then
            X.remove( w )
        end if
    end if
end procedure
```

Με τη LocalArcCons, εξασφαλίζουμε ότι μετά την αφαίρεση μίας τιμής w από το πεδίο της μίας μεταβλητής (δηλαδή της VarFired), θα αφαιρεθεί η ίδια τιμή από το πεδίο της άλλης μεταβλητής, εφόσον βέβαια ανήκει σε αυτό.

Σε αυτόν τον περιορισμό φαίνεται ξεκάθαρα η διαφορά στην τάξη πολυπλοκότητας, μεταξύ των ArcCons και LocalArcCons. Ουσιαστικά, δρέπουμε τους καρπούς της απόφασής μας να χρησιμοποιήσουμε τον AC-5.

Ανάλογοι είναι οι ορισμοί για τον γενικότερο περιορισμό $X = \pm Y + C$, με C ακέραια σταθερά.

3.4.3 Ο περιορισμός $X \neq Y$

procedure ArcCons
  if $|D_Y| = 1$ **and** Y.value $\in D_X$ **then**
    X.remove(Y.value)
  end if
  if $|D_X| = 1$ **and** X.value $\in D_Y$ **then**
    Y.remove(X.value)
  end if
end procedure

Η LocalArcCons ταυτίζεται και εδώ με την ArcCons — χωρίς «απώλειες» σε πολυπλοκότητα.

3.4.4 Ο περιορισμός AllDiff

Ο περιορισμός AllDiff εφαρμόζεται πάνω σε ένα σύνολο μεταβλητών — εδώ θα το συμβολίζουμε με Arr. Επιβάλλει όλες οι μεταβλητές που αποτελούν το Arr, να είναι διαφορετικές μεταξύ τους, όταν είναι δεσμευμένες· είναι δηλαδή μια γενίκευση του περιορισμού που περιγράφηκε προηγουμένως. Με $\min\{Arr\}$ και $\max\{Arr\}$, συμβολίζουμε την ελάχιστη και τη μέγιστη τιμή αντίστοιχα, που υπάρχει στην ένωση των πεδίων των μεταβλητών που αποτελούν το Arr.

procedure ArcCons
  if $\max\{Arr\} - \min\{Arr\} + 1 < |Arr|$ **then**
    Raise "inconsistency flag"
  end if
  for each $X \in Arr$ **do**
    for each $Y \in Arr$, with $Y \not\equiv X$ **do**
      if $|D_Y| = 1$ **and** Y.value $\in D_X$ **then**
        X.remove(Y.value)
      end if
    end for
  end for
end procedure

Η συνθήκη του πρώτου **if** παραπάνω, είναι εμπνευσμένη από την αρχή του περιστερώνα: *Αν τοποθετήσουμε n περιστέρια σε $n-1$ φωλιές, θα υπάρχει μια τουλάχιστον φωλιά με 2 τουλάχιστον περιστέρια.* [26] Εδώ τα «περιστέρια» είναι οι μεταβλητές και οι «φωλιές» που μπορούν να μοιραστούν, είναι η ένωση των τιμών τους. Αν οι τιμές είναι λιγότερες από τις μεταβλητές, τότε αναγκαστικά δύο μεταβλητές θα έχουν την ίδια τιμή και συνεπώς ο περιορισμός AllDiff δεν θα ισχύει.

procedure LocalArcCons(VarFired, w, Q)
  if $|D_{\text{VarFired}}| = 1$ **then**
    for each $X \in Arr$, with $X \not\equiv$ VarFired **do**
      if VarFired.value $\in D_X$ **then**
        X.remove(VarFired.value)
      end if
    end for
  end if
end procedure

Και οι δύο υπορουτίνες για το AllDiff, έχουν μία ιδιαιτερότητα σε σχέση με τους υπόλοιπους περιορισμούς: Όταν αφαιρούν μία τιμή από κάποια μεταβλητή του Arr, θα πρέπει να επανεξεταστεί η συνέπεια των ακμών του περιορισμού. Σε αυτήν την περίπτωση τοποθετούνται στο Q στοιχεία του τύπου (i, NIL, w).

Βασιζόμενος στο παραπάνω, θα μπορούσε κανείς να ισχυριστεί εδώ, ότι οι ArcCons και LocalArcCons δεν κάνουν καλά τη «δουλειά» τους. Ωστόσο, σε συνδυασμό με τον AC-5, οι υπορουτίνες αυτές δίνουν τα σωστά αποτελέσματα.

3.4.5 Οι περιορισμοί $X \vee Y$ και $\neg(X \vee Y)$

Η υπορουτίνα που θα παρουσιαστεί στη συνέχεια αφορά και τους δύο περιορισμούς, $X \vee Y$ και $\neg(X \vee Y)$. Για τον πρώτο περιορισμό, η μεταβλητή neg των υπορουτινών θα είναι false, ενώ για τον δεύτερο η neg θα είναι true. Θεωρούμε ότι το "true" ταυτίζεται με τον ακέραιο αριθμό 1 και το "false" με το 0. Επίσης, θεωρούμε ότι οι μεταβλητές X και Y είναι boolean, ήτοι $D_X, D_Y \subseteq \{0, 1\}$. (Αν οι μεταβλητές δεν είναι boolean, η ArcCons αρχικά αφαιρεί τις τιμές $v < 0$ και $v > 1$ από τα πεδία τους.[6])

procedure ArcCons

[6] Εδώ, για χάρη της συμβατότητας με τη συνέπεια-ορίων, οι τιμές αφαιρούνται από τα άκρα του πεδίου, μέχρι το τελευταίο να γίνει υποσύνολο του $\{0, 1\}$.

```
    if |D_Y| = 1 and Y.value = neg and neg ∈ D_X then
        X.REMOVE( neg )
    end if
    if |D_X| = 1 and X.value = neg and neg ∈ D_Y then
        Y.REMOVE( neg )
    end if
end procedure
```

Η LocalArcCons ταυτίζεται με την ArcCons.

3.4.6 Μετα-περιορισμοί

Οι *μετα-περιορισμοί* (meta-constraints) είναι περιορισμοί που ορίζονται με βάση άλλους «περιορισμούς», όπως αυτοί που ορίστηκαν προηγουμένως. Π.χ. ένας μετα-περιορισμός είναι το $X = (Y = Z)$. Όταν το Y είναι ίσιο με το Z, το X είναι true, αλλιώς, αν το Y είναι διάφορο του Z, το X είναι false. Προφανώς, αν τα D_Y και D_Z περιέχουν τιμές για τις οποίες ο «περιορισμός» $Y = Z$, μπορεί να γίνει και αληθής και ψευδής, τότε $D_X = \{0, 1\}$.

Η λέξη «περιορισμός», μπήκε παραπάνω σε εισαγωγικά, για να τονίσουμε ότι στην πραγματικότητα, το $Y = Z$ δεν είναι περιορισμός, αλλά μια *σχέση*, που άλλες φορές ισχύει και άλλες όχι. Ωστόσο, αν $D_X = \{1\}$, τότε η σχέση θα πρέπει να ισχύει και αντιστρόφως, αν $D_X = \{0\}$.

3.4.7 Ο μετα-περιορισμός $X = (Y < Z)$

```
 1: procedure ArcCons
 2:     if 0 ∈ D_X and Y.max < Z.min then
 3:         X.REMOVE( 0 )
 4:     end if
 5:     if 1 ∈ D_X and Y.min ≥ Z.max then
 6:         X.REMOVE( 1 )
 7:     end if
 8:     if |D_X| = 1 then
 9:         if X.value = 0 then
10:             while Z.max > Y.max do
11:                 Z.REMOVE( Z.max )
12:             end while
13:             while Z.min > Y.min do
```

```
14:                 Y.REMOVE( Y.min )
15:             end while
16:         else
17:             while Y.max ≥ Z.max do
18:                 Y.REMOVE( Y.max )
19:             end while
20:             while Y.min ≥ Z.min do
21:                 Z.REMOVE( Z.min )
22:             end while
23:         end if
24:     end if
25: end procedure
```

Τις γραμμές 17–22, τις δανειστήκαμε από τον αντίστοιχο αλγόριθμο για τον περιορισμό $X < Y$ (§3.4.1). Είναι εμφανές ότι, όταν η μεταβλητή X δεσμεύεται, τότε η σχέση $Y < Z$, ή η άρνησή της, γίνεται περιορισμός.

Η LocalArcCons ταυτίζεται με την ArcCons.

3.4.8 Οι μετα-περιορισμοί $X = (Y = Z)$ και $X = (Y \neq Z)$

Η υπορουτίνα που θα παρουσιαστεί στη συνέχεια αφορά και τους δύο μετα-περιορισμούς, $X = (Y = Z)$ και $X = (Y \neq Z)$. Για τον πρώτο μετα-περιορισμό, η μεταβλητή neg θα είναι false, ενώ για τον δεύτερο η neg θα είναι true.

```
 1: procedure ArcCons
 2:     if neg ∈ D_X and Y.min = Z.max and Y.max = Z.min then
 3:                                  ▷ ... if Y and Z are bound to the same value
 4:         X.REMOVE( neg )
 5:     end if
 6:     if ¬neg ∈ D_X and D_Y ∩ D_Z = ∅ then
 7:         X.REMOVE( ¬neg )
 8:     end if
 9:     if |D_X| = 1 then
10:         if X.value = neg then
11:             if |D_Z| = 1 and Z.value ∈ D_Y then
12:                 Y.REMOVE( Z.value )
13:             end if
```

```
14:             if |D_Y| = 1 and Y.value ∈ D_Z then
15:                 Z.REMOVE( Y.value )
16:             end if
17:         else
18:             for each v ∈ D_Y do
19:                 if v ∉ D_Z then
20:                     Y.REMOVE( v )
21:                 end if
22:             end for
23:             for each v ∈ D_Z do
24:                 if v ∉ D_Y then
25:                     Z.REMOVE( v )
26:                 end if
27:             end for
28:         end if
29:     end if
30: end procedure
```

Οι γραμμές 11–16, προέρχονται από τον αντίστοιχο αλγόριθμο του περιορισμού $X \neq Y$ (§3.4.3) και οι γραμμές 18–27 από τον περιορισμό $X = Y$ (§3.4.2).

Η LocalArcCons αυτή τη φορά είναι διαφορετική από την ArcCons και αυτό οφείλεται στην έμμεση ύπαρξη του περιορισμού $X = Y$. Κατά τα άλλα, δεν είναι δύσκολο να κατασκευαστεί, οπότε ο ορισμός της παραλείπεται.

Για να δούμε αν ισχύει $D_Y \cap D_Z = \emptyset$ (γραμμή 6 του παραπάνω αλγορίθμου), καλούμε τη συνάρτηση EmptyIntersection(Y, Z), που ορίζεται ως εξής:

```
function EMPTYINTERSECTION(Y, Z)
    v_y ← −∞
    while true do
        v_z ← Z.next(v_y)
        v_y ← Y.next(v_y)
        if v_y = +∞ or v_z = +∞ then
            break
        end if
        if v_y = v_z then
            return false
        else if v_y > v_z then
            if v_y ∈ D_Z then
```

```
            return false
        end if
    else
        if v_z ∈ D_Y then
            return false
        end if
        v_y ← v_z
    end if
end while
return true
end function
```

3.4.9 Οι μετα-περιορισμοί $X = (Y \wedge Z)$ και $X = \neg(Y \wedge Z)$

Παρακάτω, για τον πρώτο μετα-περιορισμό (από τους $X = (Y \wedge Z)$ και $X = \neg(Y \wedge Z)$), η μεταβλητή neg θα είναι false, ενώ για τον δεύτερο η neg θα είναι true.

```
procedure ArcCons
    if neg ∈ D_X and Y.min ∧ Z.min then
        X.remove( neg )
    end if
    if ¬neg ∈ D_X and ¬(Y.max ∧ Z.max) then
        X.remove( ¬neg )
    end if
    if |D_X| = 1 then
        if X.value ≠ neg then
            if 0 ∈ D_Y then
                Y.remove( 0 )
            end if
            if 0 ∈ D_Z then
                Z.remove( 0 )
            end if
        end if
    end if
end procedure
```

Χωρίς να έχουμε απώλεια στην τάξη πολυπλοκότητας, μπορούμε να ορίσουμε την LocalArcCons βάσει της ArcCons.

Ανάλογους ορισμούς έχουμε και για τους μετα-περιορισμούς $X = (Y \vee Z)$ και $X = \neg(Y \vee Z)$. Σε αυτό το σημείο ολοκληρώσαμε την αναφορά μας στους μετα-περιορισμούς.

3.4.10 Ο περιορισμός $X = CY$

Υπενθυμίζουμε ότι με C συμβολίζουμε μία ακέραια σταθερά. Εδώ, για να έχει νόημα ο περιορισμός, έχουμε $C \neq 0$.

procedure ArcCons
  for each $v \in D_X$ **do**
    if $v \pmod C \neq 0$ **or** $\lfloor \frac{v}{C} \rfloor \notin D_Y$ **then**
      X.remove(v)
    end if
  end for
  for each $v \in D_Y$ **do**
    if $C \cdot v \notin D_X$ **then**
      Y.remove(v)
    end if
  end for
end procedure

procedure LocalArcCons(VarFired, w, Q)
  if $X \equiv$ VarFired **then**
    SuppVal $\leftarrow \lfloor \frac{w}{C} \rfloor$
    if $w \pmod C = 0$ **and** SuppVal $\in D_Y$ **then**
      Y.remove(SuppVal)
    end if
  else ▷ $Y \equiv$ VarFired
    SuppVal $\leftarrow C \cdot w$
    if SuppVal $\in D_X$ **then**
      X.remove(SuppVal)
    end if
  end if
end procedure

Θέματα υλοποίησης

Η ακέραια διαίρεση $\lfloor \frac{a}{b} \rfloor$, υλοποιήθηκε στη C++ με τον τελεστή "/". Ωστόσο, σε κάποια μηχανήματα, αν $a < 0$ ή $b < 0$, το a/b μπορεί να δώσει $\lceil \frac{a}{b} \rceil$ [27].

3.4.11 Ο περιορισμός $X = \lfloor Y/C \rfloor$

Σε αυτόν τον περιορισμό —ο οποίος είναι όμοιος με την ακέραια διαίρεση της C/C++—, κάθε τιμή v της X, αντιστοιχεί στις τιμές $[C \cdot v, C(v+1)]$ της Y, εφόσον βέβαια υπάρχουν, για $C > 0$.

procedure ArcCons
  for each $v \in D_X$ **do**
    if Y.next$(C \cdot v - 1) \geq C(v+1)$ **then**
      X.remove(v)
    end if
  end for
  for each $v \in D_Y$ **do**
    if $\lfloor \frac{v}{C} \rfloor \notin D_X$ **then**
      Y.remove(v)
    end if
  end for
end procedure

procedure LocalArcCons(VarFired, w, Q)
  if $X \equiv$ VarFired **then**
    SuppVal $\leftarrow C \cdot w - 1$
    while (SuppVal $\leftarrow Y$.next(SuppVal)) $< C(w+1)$ **do**
      Y.remove(SuppVal)
    end while
  else ▷ $Y \equiv$ VarFired
    SuppVal $\leftarrow \lfloor \frac{w}{C} \rfloor$
    if SuppVal $\in D_X$ **then**
      if Y.next$(C \cdot$ SuppVal $- 1) \geq C($SuppVal $+ 1)$ **then**
        X.remove(SuppVal)
      end if
    end if
  end if
end procedure

Στην παραπάνω συνθήκη του **while**, πρώτα γίνεται ανάθεση τιμής στο SuppVal και έπειτα αυτό συγκρίνεται με το $C(w+1)$.

3.4.12 Ο περιορισμός $X = \lfloor C/Y \rfloor$

Σε αυτόν τον περιορισμό, κάθε τιμή v της X, με $v \neq -1$ και $v \neq 0$, αντιστοιχεί στις τιμές $(\lfloor \frac{C}{v+1} \rfloor, \lfloor \frac{C}{v} \rfloor]$ της Y, εφόσον βέβαια υπάρχουν, για $C > 0$. Οι αντίστοιχες τιμές για το -1 της X, είναι οι $(-\infty, -C]$. Στο 0 της X, αντιστοιχούν οι $(C, +\infty)$ τιμές της Y.

procedure ArcCons
  if $-1 \in D_X$ **and** $-\infty = Y$.previous$(-C+1)$ **then** ▷ $-C = \lfloor \frac{C}{-1} \rfloor$
    X.remove(-1)
  end if
  if $0 \in D_X$ **and** Y.next$(C) = +\infty$ **then** ▷ $C = \lfloor \frac{C}{1} \rfloor$
    X.remove(0)
  end if
  for each $v \in D_X$, with $v \neq -1$, $v \neq 0$ **do**
    if Y.next$(\lfloor \frac{C}{v+1} \rfloor) > \lfloor \frac{C}{v} \rfloor$ **then**
      X.remove(v)
    end if
  end for
  for each $v \in D_Y$ **do**
    if $\lfloor \frac{C}{v} \rfloor \notin D_X$ **then**
      Y.remove(v)
    end if
  end for
end procedure

Από τον παραπάνω αλγόριθμο, μπορεί κανείς να κατασκευάσει και την LocalArcCons — ανατρέχοντας ίσως και στην αντίστοιχη υπορουτίνα για τον περιορισμό $X = \lfloor Y/C \rfloor$.

3.4.13 Οι περιορισμοί min και max

Ο περιορισμός min εφαρμόζεται πάνω σε ένα σύνολο μεταβλητών, έστω Arr και μία μεταβλητή X, η οποία θα ισούται με την ελάχιστη τιμή των μεταβλητών του Arr. Με την ArcCons εξασφαλίζουμε ότι $X.\min = \min_{Y \in Arr}\{Y.\min\}$ και $X.\max = \min_{Y \in Arr}\{Y.\max\}$, αφαιρώντας τιμές από τα πεδία εκείνα που

παραβιάζουν αρχικά αυτές τις συνθήκες.[7] Επίσης, για τις ενδιάμεσες τιμές των μεταβλητών, επιβάλλονται οι συνθήκες:

$$\forall v \in D_X, \exists Y \in Arr : v \in D_Y \quad \text{και}$$
$$\forall Y \in Arr, \forall v \in D_Y, v \in D_X.$$

Η LocalArcCons, αν αφαιρεθεί μία τιμή w από το πεδίο της X, τότε φροντίζει να ισχύει $\forall Y \in Arr$, $w \notin D_Y$. Αν αφαιρεθεί μία τιμή w από κάποια μεταβλητή $Y \in Arr$ και $w \in D_X$, τότε ελέγχεται αν υπάρχει κάποια άλλη μεταβλητή στο Arr, της οποίας το πεδίο περιέχει το w· αν δεν υπάρχει τέτοια μεταβλητή, τότε το w αφαιρείται από το D_X.

3.4.14 Ο περιορισμός Inverse

Ο περιορισμός *Inverse* (Αντιστροφή), εφαρμόζεται ανάμεσα σε δύο πίνακες από περιορισμένες μεταβλητές. Έστω ότι έχουμε έναν πίνακα Arr, του οποίου οι τιμές των μεταβλητών είναι θετικές και επιθυμούμε ο «αντίστροφός» του να είναι ο $ArrInv$. Τότε θα ισχύει:

$$\forall v \in D_{ArrInv[i]}, \quad D_{Arr[v]} \ni i.$$

Αν δεν υπάρχει v, τέτοιο ώστε $i \in D_{Arr[v]}$, τότε το πεδίο της $ArrInv[i]$ θα περιέχει την ειδική τιμή -1 και μόνο.

Απλοποιώντας τους συμβολισμούς, μπορούμε να γράψουμε ότι πρέπει να ισχύει:

$$Arr[ArrInv[i]] = i \quad \text{και} \quad ArrInv[Arr[i]] = i.$$

Εξ ου και η ονομασία "Inverse". Βέβαια, αυτές οι σχέσεις θα είχαν νόημα, αν όλες μεταβλητές των δύο πινάκων ήταν δεσμευμένες και αν η μοναδική τιμή που θα περιείχε το πεδίο τιμών κάθε μεταβλητής, συμβολιζόταν με το ίδιο το όνομα της μεταβλητής. Ακόμα θα έπρεπε $\forall i$, $ArrInv[i] \neq -1$.

procedure ArcCons
  for $i \leftarrow 0$ **to** $|ArrInv| - 1$ **do**
    for each $v \in D_{ArrInv[i]}$, with $v \neq -1$ **do**
      if $v \notin [0, |Arr|)$ **or** $i \notin D_{Arr[v]}$ **then**
        $ArrInv[i]$.remove(v)

[7] Για τον περιορισμό max ισχύει ό,τι και στον min, με μία διαφορά: Στην ArcCons εξασφαλίζεται ότι $X.\min = \max_{Y \in Arr}\{Y.\min\}$ και $X.\max = \max_{Y \in Arr}\{Y.\max\}$.

```
            else if |D_{Arr[v]}| = 1 and −1 ∈ D_{ArrInv[i]} then
                ArrInv[i].REMOVE( −1 )
            end if
        end for
    end for
    for i ← 0 to |Arr| − 1 do
        for each v ∈ D_{Arr[i]} do
            if v ∉ [0, |ArrInv|) or i ∉ D_{ArrInv[v]} then
                Arr[i].REMOVE( v )
            end if
        end for
    end for
end procedure
```

```
1:  procedure LocalArcCons(VarFired, w, Q)
2:      i ← index of array that VarFired belongs to
3:      if VarFired ∈ ArrInv then
4:          if w ∈ [0, |Arr|) and i ∈ D_{Arr[w]} then
5:              Arr[w].REMOVE( i )
6:          end if
7:      else                                              ▷ VarFired ∈ Arr
8:          if w ∈ [0, |ArrInv|) and i ∈ D_{ArrInv[w]} then
9:              ArrInv[w].REMOVE( i )
10:         end if
11:         if |D_{VarFired}| = 1 and −1 ∈ D_{ArrInv[VarFired.value]} then
12:             ArrInv[VarFired.value].REMOVE( −1 )
13:         end if
14:     end if
15: end procedure
```

Διαβάζοντας την LocalArcCons, ίσως αναρωτηθεί κανείς αν είναι απαραίτητοι οι έλεγχοι $w \in [0, |Arr|)$ και $w \in [0, |ArrInv|)$, στις γραμμές 4 και 8 αντίστοιχα. Εφόσον οι τιμές που περιέχονται σε αυτά τα διαστήματα, αφαιρέθηκαν από την ArcCons και αφού η LocalArcCons δεν καλείται για τις τιμές που αφαίρεσε η ArcCons του ίδιου περιορισμού,[8] ποιος ο λόγος να κάνουμε αυτούς τους ελέγχους; Η απάντηση είναι, γιατί οι τιμές εκτός των διαστημά-

[8] Όπως έχει αναφερθεί, οι μοναδικές υπορουτίνες που δεν υπακούουν σε αυτόν τον κανόνα, είναι αυτές για τον περιορισμό AllDiff.

των, είναι πιθανόν να έχουν αφαιρεθεί από συναρτήσεις ArcCons που αφορούν άλλους περιορισμούς, στο πρώτο **while** του αλγορίθμου AC-5-Naxos-Final.

Θέματα υλοποίησης

Στη γραμμή 2 της LocalArcCons, παίρνουμε τον αύξοντα αριθμό της VarFired, στον πίνακα στον οποίο ανήκει. Κατά το σχεδιασμό του επιλυτή, η υποστήριξη αυτής της γραμμής, δεν ήταν τετριμμένη διαδικασία.

Η παράμετρος VarFired που παίρνει η κάθε LocalArcCons, δεν είναι τίποτα άλλο από έναν δείκτη σε μία μεταβλητή. Έχοντας αυτό μόνο στα χέρια μας για τη VarFired, δεν μπορούμε να γνωρίζουμε σε σταθερό χρόνο, αν ένας από τους πίνακες στους οποίους ανήκει είναι ο Arr ή ο $ArrInv$. Λέμε «ένας από τους πίνακες», γιατί μία μεταβλητή μπορεί να ανήκει σε πολλούς πίνακες. Οι πίνακες στον επιλυτή, έχουν υλοποιηθεί σαν πίνακες από *δείκτες* σε μεταβλητές και όχι σαν πίνακες μεταβλητών. Το γεγονός αυτό κάνει τα πράγματα δύσκολα.

Η λύση που υιοθετήθηκε, είναι ότι για κάθε περιορισμό Inverse, δημιουργούμε έναν πίνακα κατακερματισμού, με κλειδί τους δείκτες μνήμης των μεταβλητών. Αναζητώντας τον δείκτη μνήμης μιας μεταβλητής στον πίνακα κατακερματισμού, βρίσκουμε σε σταθερό χρόνο τον πίνακα στον οποίο ανήκει —δηλαδή τον Arr ή τον $ArrInv$— και τη θέση της σε αυτόν.

3.4.15 Ο περιορισμός του αθροίσματος

Στον περιορισμό του αθροίσματος, καθώς και σε αυτόν του γινομένου που θα περιγραφεί στην επόμενη παράγραφο, η πλήρης επιβολή συνέπειας κρίθηκε ασύμφορη. Αντί αυτής, θα εφαρμόσουμε συνέπεια-ορίων.

Στη συνέχεια, θεωρούμε ότι η μεταβλητή X ισούται με το άθροισμα του συνόλου μεταβλητών Arr.

procedure ArcCons
    summin $\leftarrow \sum_{Y \in Arr} Y$.min
    summax $\leftarrow \sum_{Y \in Arr} Y$.max
    changedsum $\leftarrow$ true

    while true **do**
        repeat
            while X.min $<$ summin **do**
                X.remove(X.min)
            end while

```
            for each Y ∈ Arr do
                while Y.min + summax − Y.max < X.min do
                    summin ← summin − Y.min
                    Y.REMOVE( Y.min )
                    summin ← summin + Y.min          ▷ summin is updated
                    changedsum ← true
                end while
            end for
        until ¬(X.min < summin)

        if ¬changedsum then
            break
            ▷ The first time, it will not break, because changedsum = true
        end if
        changedsum ← false

        repeat
            while X.max > summax do
                X.REMOVE( X.max )
            end while
            for each Y ∈ Arr do
                while Y.max + summin − Y.min > X.max do
                    summax ← summax − Y.max
                    Y.REMOVE( Y.max )
                    summax ← summax + Y.max          ▷ summax is updated
                    changedsum ← true
                end while
            end for
        until ¬(X.max > summax)

        if ¬changedsum then
            break
        end if
        changedsum ← false
    end while
end procedure
```

Για να δούμε αν η ελάχιστη τιμή $Y.\min$ του πεδίου μιας μεταβλητής $Y \in Arr$

είναι συνεπής, προσθέτουμε σε αυτήν το άθροισμα των μεγίστων των υπόλοιπων μεταβλητών του Arr (δηλαδή το summax $- Y$.max). Αν το αποτέλεσμα είναι μικρότερο του X.min, τότε δεν υπάρχει καμία περίπτωση η Y.min να είναι συνεπής, οπότε αφαιρείται. Με ανάλογο τρόπο, ελέγχουμε την τιμή Y.max.

procedure LocalArcCons(VarFired, w, Q)
  if $w <$ VarFired.min **or** $w >$ VarFired.max **then**
    ArcCons() ▷ Only boundary values are taken into account
  end if
end procedure

Οι μέθοδοι LocalArcCons που αφορούν συνέπεια-ορίων, μπορούν να υλοποιηθούν πιο αποδοτικά, αν αντί για ένα απλό όρισμα w, παίρνουν ένας εύρος Δ με τιμές που αφαιρέθηκαν από τη μεταβλητή VarFired. Και αυτό γιατί, μόνο το ελάχιστο ή το μέγιστο του Δ χρειάζεται να ελεγχθεί για να έχουμε συνέπεια-ορίων.

3.4.16 Ο περιορισμός $X = YZ$

Και εδώ, θα εφαρμόσουμε συνέπεια-ορίων.

procedure ArcCons
  repeat
    changed $\leftarrow$ false
    bounds $\leftarrow \{Y.\min \cdot Z.\min, Y.\min \cdot Z.\max, Y.\max \cdot Z.\min,$
                $Y.\max \cdot Z.\max\}$
    pmin $\leftarrow$ min{bounds}
    pmax $\leftarrow$ max{bounds}
    while $X.\min <$ pmin **do**
      X.remove($X.\min$)
    end while
    while $X.\max >$ pmax **do**
      X.remove($X.\max$)
    end while
    ProductPrune(X, Y, Z, changed)
    ProductPrune(X, Z, Y, changed)
  until ¬changed
end procedure

procedure ProductPrune(X, Y, Z, changed)
  while $Y.\min \cdot Z.\min \notin [X.\min, X.\max]$ **and**
      $Y.\min \cdot Z.\max \notin [X.\min, X.\max]$ **do**
    X.remove($Y.\min$)
    changed $\leftarrow$ true
  end while
  while $Y.\max \cdot Z.\min \notin [X.\min, X.\max]$ **and**
      $Y.\max \cdot Z.\max \notin [X.\min, X.\max]$ **do**
    X.remove($Y.\max$)
    changed $\leftarrow$ true
  end while
end procedure

Η ProductPrune αφαιρεί ασυνεπείς τιμές από το πεδίο της μεταβλητής που βρίσκεται στο δεύτερο όρισμά της. Στις παραπάνω συναρτήσεις, έχει ληφθεί υπόψη το ενδεχόμενο οι Y και Z να περιέχουν αρνητικές τιμές.

procedure LocalArcCons(VarFired, w, Q)
  if $w <$ VarFired.min **or** $w >$ VarFired.max **then**
    ArcCons() ▷ Only boundary values are taken into account
  end if
end procedure

Στους αλγορίθμους επιβολής συνέπειας-ορίων, δεν χρησιμοποιούμε καθόλου τα ίδια τα πεδία των μεταβλητών, παρά μόνο τα άκρα τους. Π.χ. στην ProductPrune αντί του D_X, χρησιμοποιήθηκε το $[X.\min, X.\max]$, το οποίο είναι υπερσύνολο του D_X.

3.5 Προτεινόμενη βελτίωση του AC-5

Κατά την εφαρμογή του AC-5, παρατηρήσαμε οι LocalArcCons καλούνταν πολλές φορές άσκοπα, για τους περιορισμούς που δεν λάμβαναν υπόψη το όρισμα w της υπορουτίνας. Πρόκειται για τους περιορισμούς που αφορούν συνέπεια-ορίων (τέτοιοι είναι οι μονοτονικοί και πολλοί άλλοι), καθώς και για τους αντισυναρτησιακούς, όπως το $\neq$ (οι οποίοι εμπεριέχουν τη συνέπεια-ορίων, καθώς προκαλούν αφαίρεση τιμών, όταν το κάτω και το άνω όριο του πεδίου μίας μεταβλητής ταυτίζονται, δηλαδή όταν αυτή είναι δεσμευμένη).

Έτσι, σκεφτήκαμε να συγκεντρώνουμε σε κάθε στοιχείο της ουράς Q του

AC-5, όλες τις τιμές w που αφαιρέθηκαν από μία συγκεκριμένη μεταβλητή. Για να εντοπίζουμε σε σταθερό χρόνο το στοιχείο της λίστας εκείνο, που αφορά μία συγκεκριμένη μεταβλητή, θα χρησιμοποιήσουμε έναν πίνακα κατακερματισμού, με κλειδί το όνομα της μεταβλητής. Τελικά, τα στοιχεία της Q θα είναι της μορφής $(\text{VarFired}, W)$, με $W = \{w_1, \ldots, w_n\}$. Μάλιστα, κάθε μεταβλητή VarFired, θα μπορεί να εμφανίζεται σε ένα το πολύ στοιχείο της Q.

Τροποποιούμε ελαφρώς και τη LocalArcCons και την κάνουμε συνάρτηση που επιστρέφει μία τιμή αληθείας: true αν δεν λαμβάνει υπόψη το όρισμα w που παίρνει, αλλιώς false. Ο αλγόριθμος που προκύπτει είναι ο εξής:

```
procedure AC-5-Hashed
    Q ← ∅
    for each (i, j) ∈ arcs(G) do
        ArcCons(i, j, Δ)
        Enqueue(i, Δ, Q)
        D_i ← D_i − Δ
    end for
    while Q ≠ ∅ do
        Pick and remove an (i, W) out of Q
        for each (k, i) ∈ arcs(G) do
            for each w ∈ W do
                IgnoreW ← LocalArcCons(k, i, w, Δ)
                Enqueue(k, Δ, Q)
                D_k ← D_k − Δ
                if IgnoreW then
                    break
                end if
            end for
        end for
    end while
end procedure

procedure Enqueue(i, Δ, Q)
    Get the position of the element (i, W) of Q     ▷ ... using a hash table
    if it does not exist then
        Q ← Q ∪ {(i, Δ)}
    else
        Replace it by (i, W ∪ Δ)
```

end if
end procedure

Μία παρατήρηση που θα μπορούσε να γίνει στον σχεδιασμό του παραπάνω αλγορίθμου, είναι γιατί δεν καλούμε ξεχωριστά τις LocalArcCons των περιορισμών που αφορούν συνέπεια-ορίων, αλλά περιμένουμε να δούμε τι θα επιστρέψει η κάθε LocalArcCons. Ο λόγος για τον οποίον προχωρήσαμε σε αυτόν τον κάπως πιο δυσανάγνωστο σχεδιασμό, είναι ότι σε πρακτικά συστήματα δίνονται συνήθως αρκετές «αρμοδιότητες» στις LocalArcCons.

Οι έλεγχοι των περιορισμών μέσω της LocalArcCons, μειώνονται αισθητά, καθώς για κάθε σύνολο W και ανάλογα με τον περιορισμό, μπορεί να απαιτηθεί μόνο ένας έλεγχός του.[9]

Μία βελτίωση στον παραπάνω αλγόριθμο, η οποία έχει ενσωματωθεί εξ ορισμού και στον AC-3, είναι η τοποθέτηση στο σύνολο W, «δίπλα» από κάθε w_i, του ονόματος του περιορισμού από τον οποίο προήλθε, έτσι ώστε να μην χρειάζεται να καλέσουμε την LocalArcCons του αντίστοιχου συμμετρικού περιορισμού, για το w_i αυτό.

Μία ακόμα βελτίωση που θα μπορούσε να γίνει, είναι η τοποθέτηση σε κάθε στοιχείο $(\text{VarFired}, W)$ της Q, μίας τιμής αληθείας με όνομα ChangedBound, η οποία θα είναι true αν κάποιο στοιχεία του W, ήταν πρώην *άκρο* της μεταβλητής VarFired, ή αλλιώς false. Στη συνέχεια, θα μπορούμε να περνάμε την ChangedBound σαν ένα επιπλέον όρισμα της LocalArcCons, έτσι ώστε, όταν ο περιορισμός για τον οποίον καλείται αυτή η υπορουτίνα αφορά συνέπεια-ορίων και η ChangedBound είναι false, η υπορουτίνα να επιστρέφει αμέσως true, χωρίς να χρειάζεται να μπει στο κύριο μέρος της.[10]

«Δίπλα» στην ChangedBound, αν αυτή είναι true, μπορεί να υπάρχει ο τελευταίος περιορισμός ο οποίος την έκανε true (έστω και αν κάποιος άλλος νωρίτερα την έκανε επίσης true), *εφόσον* αυτός αφορά καθαρά επιβολή συνέπειας-ορίων. Διαφορετικά, θα υπάρχει η ειδική τιμή NIL. Η σκοπιμότητα της ύπαρξης αυτού του «δείκτη» στον περιορισμό, είναι για να τον βλέπει ο αλγόριθμος και να μην καλέσει καθόλου την LocalArcCons για αυτόν (όσον αφορά τη μεταβλητή VarFired).

[9] Η διαφορά γίνεται πιο αισθητή στους μη δυαδικούς περιορισμούς, στους οποίους εφαρμόζεται συνέπεια-ορίων, όπως π.χ. στον $X_1 = \sum_{i=2}^{10} X_i$.

[10] Αυτό έχει πρακτική σημασία μόνο για πολύπλοκους περιορισμούς που αφορούν συνέπεια-ορίων. Παράδειγμα ενός τέτοιου —μη δυαδικού όμως— περιορισμού, είναι ο $X_1 = \sum_{i=2}^{10} X_i$ που προαναφέρθηκε. Υπάρχουν βέβαια και πιο πολύπλοκοι *δυαδικοί* περιορισμοί για συνέπεια-ορίων, οι οποίοι όμως είναι πιο πολύ θεωρητικής, παρά πρακτικής σημασίας.

Τέλος, μία αλλαγή από την οποία θα προέκυπτε ελάττωση των πράξεων πάνω στην Q, θα ήταν να επιτρέπονται να έχουμε *εύρη* αφαιρεμένων τιμών από κάποιο πεδίο και όχι μόνο απλές τιμές w.

3.6 Αναζήτηση λύσης μέσω στόχων

3.6.1 Εισαγωγή

Όπως είναι γνωστό, στα περισσότερα προβλήματα δεν αρκεί μόνο η επιβολή 2-συνέπειας, για να βρεθεί μία λύση (βλ. και Σχήμα 2.2). Από ένα σημείο και μετά, πρέπει να αρχίσουμε την αναζήτηση, επαναλαμβάνοντας την ανάθεση τιμών σε μεταβλητές και ελέγχοντας κάθε φορά —π.χ. μετά από κάθε ανάθεση— αν το δίκτυο περιορισμών είναι 2-συνεπές, σύμφωνα και με την πρακτική της *διατήρησης συνέπειας ακμών* (maintaining arc consistency – MAC).[11] [29, 28] Αν μία ανάθεση τιμής προκαλέσει ασυνέπεια, τότε θα πρέπει να ακυρωθεί και να επιλεγεί μία άλλη τιμή.

Για να διευκολυνθεί, ή, καλύτερα, για να καθοδηγηθεί η αναζήτηση, έχει υλοποιηθεί στον επιλυτή ένας *μηχανισμός στόχων*. Ο προγραμματιστής που χρησιμοποιεί τον επιλυτή μπορεί να ορίσει τους δικούς του στόχους, ή να εκμεταλλευτεί τους ενσωματωμένους. Συνήθως, ένας στόχος προκαλεί την ανάθεση τιμής σε μία περιορισμένη μεταβλητή, ή την αφαίρεση τιμής από το πεδίο της. Αν στην πορεία της αναζήτησης καταλήξουμε σε αδιέξοδο, οι στόχοι που οδήγησαν σε αυτό ακυρώνονται αυτόματα από τον επιλυτή και το δίκτυο περιορισμών με τις μεταβλητές που το συνθέτουν επανέρχεται στην κατάσταση που βρισκόταν πριν εκτελεστούν οι στόχοι.

Γενικά, ένας στόχος μπορεί να αναθέτει ή να αφαιρεί τιμές σε μία ή περισσότερες μεταβλητές, ή να χρησιμοποιείται για την επιλογή μεταβλητής για να της ανατεθεί τιμή και, κατά αυτόν τον τρόπο, να καθορίζει τη μέθοδο αναζήτησης. Ένας στόχος, κατά τον τερματισμό της εκτέλεσής του, μπορεί προαιρετικά να «γεννήσει» έναν άλλον στόχο· αυτή η δυνατότητα μπορεί να προσδώσει χαρακτηριστικά αναδρομής στη διατύπωση των στόχων. Τέλος, δεν θα πρέπει να παραλείψουμε τους *μετα-στόχους* AND και OR. Τους χαρακτηρίσαμε σαν

[11] Στο [28] περιγράφεται ένας βελτιωμένος αλγόριθμος διατήρησης συνέπειας ακμών *MAC Extended (MACE)*. Ο προκαθορισμένος αλγόριθμος που χρησιμοποιεί ο επιλυτής είναι πιο κοντά στον γενικό αλγόριθμο MAC· η επιλογή του MAC, έναντι του MACE, είχε να κάνει με το ότι ο MACE παίρνει αποφάσεις, π.χ. για το ποιες μεταβλητές θα επιλεγούν για να τους ανατεθεί τιμή, χωρίς να ερωτάται ο προγραμματιστής που φτιάχνει τους στόχους. Με λίγα λόγια, ο MACE ακυρώνει τον μηχανισμό στόχων.

«μετα-στόχους», γιατί είναι στόχοι που απλά υπάρχουν για να εξυπηρετούν, ο καθένας τους, δύο άλλους στόχους, τους οποίους λέμε *υπο-στόχους*. Για να πετύχει ο στόχος-AND, θα πρέπει να ικανοποιηθούν και οι δύο υπο-στόχοι του, ενώ για να πετύχει ο OR, αρκεί να ικανοποιηθεί ένας από τους υπο-στόχους του.

Αξίζει να σημειωθεί ότι οι στόχοι-OR, είναι γνωστοί και σαν *σημεία επιλογής* (choice points). Πράγματι, είναι σημεία στα οποία έχουμε δύο εναλλακτικές επιλογές, δηλαδή σημεία στα οποία διακλαδώνεται το δένδρο αναζήτησης. Ο στόχος-OR υπαγορεύει να επιλέξουμε ένα υπο-στόχο του και αν τελικά δεν πετύχει αυτός, να αναιρεθούν οι όποιες (αλυσιδωτές) αλλαγές που αυτός επέφερε στα πεδία των μεταβλητών και να δοκιμάσουμε τον άλλον υπο-στόχο. Αν φυσικά και αυτός αποτύχει, τότε και ο στόχος-OR αποτυγχάνει.

3.6.2 Αντικειμενοστραφής μοντελοποίηση

Η αφηρημένη βασική κλάση για έναν στόχο στον Naxos Solver, ορίζεται ως εξής:

```
class  NsGoal  {
    public:
        virtual bool  isGoalAND (void)  const;
        virtual bool  isGoalOR  (void)  const;
        virtual NsGoal*  getFirstSubGoal  (void)  const;
        virtual NsGoal*  getSecondSubGoal (void)  const;

        virtual NsGoal*  GOAL (void) = 0;
};
```

Από την κλάση αυτή, παράγονται οι κλάσεις των μετα-στόχων `NsgAND` και `NsgOR`, των οποίων η μέθοδος-κατασκευής παίρνει δύο ορίσματα (τύπου "`NsGoal*`"), που δεν είναι τίποτα άλλο παρά οι δύο υπο-στόχοι τους. Όλες οι μέθοδοι της `NsGoal`, εκτός από την `GOAL()`, αφορούν τους μετα-στόχους. Ο προγραμματιστής που θα ορίσει δικούς του στόχους, χρειάζεται να ασχοληθεί μόνο με την `GOAL()`.

Κάθε στόχος που ορίζεται από έναν προγραμματιστή που χρησιμοποιεί τον επιλυτή, θα είναι μία κλάση η οποία θα παράγεται, έστω και έμμεσα, από τη βασική κλάση `NsGoal`. Συνεπώς θα πρέπει να ορίζεται στον κάθε στόχο, η

μέθοδος GOAL(). Η κλάση του στόχου μπορεί ασφαλώς να περιέχει και οποιεσδήποτε άλλες μεθόδους — πλην αυτών της βασική κλάσης, προς αποφυγή συγχύσεων με τους μετα-στόχους.

Η GOAL() είναι μία κρίσιμη μέθοδος, καθώς εκτελείται κάθε φορά που ο επιλυτής προσπαθεί να ικανοποιήσει έναν στόχο. Η μέθοδος αυτή επιστρέφει έναν δείκτη σε NsGoal, δηλαδή επιστρέφει τον επόμενο στόχο προς ικανοποίηση. Αν ο δείκτης είναι το 0, αυτό σημαίνει ότι ο τρέχων στόχος πέτυχε — και έτσι δεν απαιτείται η δημιουργία κάποιου άλλου στόχου.

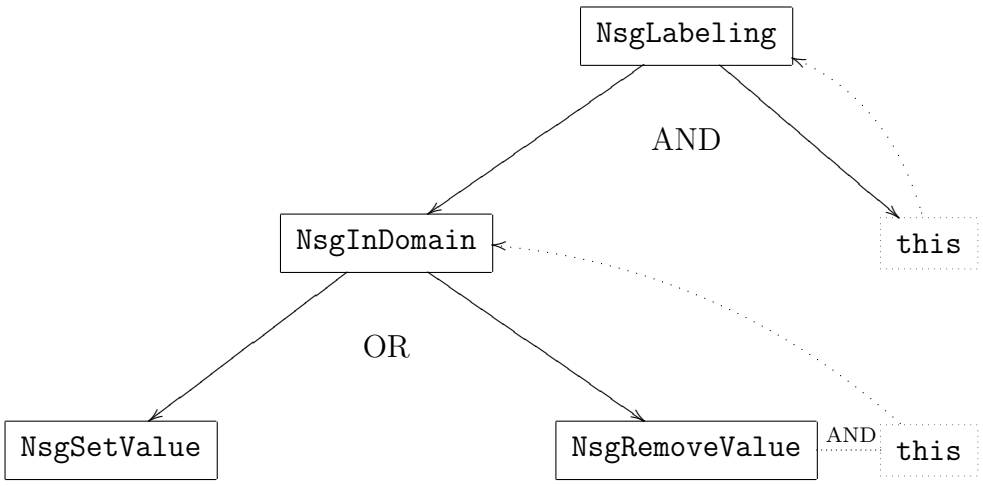
Ας δούμε ένα παράδειγμα στόχων, οι οποίοι είναι μάλιστα ενσωματωμένοι στον επιλυτή, καθώς χρησιμοποιούνται ευρέως. Πρόκειται για τους στόχους της πρώτα-κατά-βάθος αναζήτησης (depth-first-search – DFS).

```
class NsgInDomain : public NsGoal  {
  private:
    NsIntVar&  Var;

  public:
    NsgInDomain (NsIntVar& Var_init)
      : Var(Var_init)    {    }

    NsGoal*  GOAL (void)
    {
        if (Var.isBound())
            return  0;
        NsInt  value = Var.min();
        return ( new NsgOR( new NsgSetValue(Var,value),
                        new NsgAND( new NsgRemoveValue(Var,value),
                                    new NsgInDomain(*this) ) ) );
    }
};


class NsgLabeling : public NsGoal  {
  private:
    NsIntVarArray&  VarArr;

  public:
    NsgLabeling (NsIntVarArray& VarArr_init)
      : VarArr(VarArr_init)    {    }
```

```
    NsGoal*  GOAL (void)
    {
        int  index = -1;
        NsUInt  minDom = NsUPLUS_INF;
        for (NsIndex i = 0;   i < VarArr.size();   ++i)   {
            if ( !VarArr[i].isBound()
                     &&  VarArr[i].size() < minDom )
            {
                minDom = VarArr[i].size();
                index = i;
            }
        }
        if (index == -1)
            return  0;
        return  ( new NsgAND( new NsgInDomain(VarArr[index]),
                              new NsgLabeling(*this) ) );
    }
};
```

Παρατηρούμε ότι στην επιστρεφόμενη τιμή της `GOAL()` (όταν αυτή δεν είναι `0`) και στις μεθόδους κατασκευής των μετα-στόχων `NsgAND` και `NsgOR`, χρησιμοποιούμε τον τελεστή `new`. Αυτό είναι απαραίτητο να γίνεται κάθε φορά, για να φτιάξουμε έναν δείκτη σε στόχο. Ο επιλυτής αναλαμβάνει στην πρώτη ευκαιρία να σβήσει έναν άχρηστο στόχο με τον τελεστή `delete`· για αυτό, *όλοι οι στόχοι που φτιάχνουμε, θα πρέπει να έχουν δημιουργηθεί με* `new` *και να μην γίνονται* `delete` *από εμάς*.

Όσον αφορά την πρακτική σημασία του παραδείγματος, όταν ζητήσουμε από τον επιλυτή να ικανοποιηθεί ο στόχος `NsgLabeling(VarArr)`, τότε αναμένουμε να ανατεθούν τιμές σε όλες τις μεταβλητές του πίνακα `VarArr`. Έτσι, ο στόχος `NsgLabeling`, στη μέθοδο `GOAL()` πάντα, διαλέγει μία μεταβλητή (συγκεκριμένα εκείνη με το μικρότερο πεδίο, σύμφωνα με το ευριστικό first-fail). Έπειτα προστάζει να δοθεί τιμή στη μεταβλητή (μέσω του στόχου `NsgInDomain` ο οποίος αναθέτει σε μία μεταβλητή την ελάχιστη τιμή του πεδίου της) *και* να ικανοποιηθεί ο στόχος `this`. Αυτός ο στόχος —που παραπέμπει και σε ένα είδος «αναδρομής»— δημιουργεί ένα άλλο στιγμιότυπο της `NsgLabeling`, πανομοιότυπο με το τρέχον στιγμιότυπο. Με το `this`, κατ΄ ουσίαν προτρέπουμε τον επιλυτή να αναθέσει τιμή και στις υπόλοιπες μεταβλητές του πίνακα `VarArr`. Όταν η `GOAL()` επιστρέψει `0`, θα έχουμε τελειώσει επιτυχώς (Σχήμα 3.1).



Σχήμα 3.1: Συνδυασμός στόχων που αποτελούν τον `NsgLabeling`

Ενώ ο `NsgLabeling` επιλέγει μία μεταβλητή για να της ανατεθεί τιμή, ο `NsgInDomain` επιλέγει την τιμή που θα ανατεθεί. Πιο συγκεκριμένα, επιλέγει πάντα την ελάχιστη τιμή του πεδίου της μεταβλητής. Έπειτα καλεί τον ενσωματωμένο στόχο `NsgSetValue` που απλά αναθέτει την τιμή στη μεταβλητή. Αν στη συνέχεια βρεθεί ότι η τιμή αυτή δεν ανήκει σε κάποια λύση, τότε αφαιρείται από το πεδίο της μεταβλητής με τον στόχο `NsgRemoveValue` και στη συνέχεια θα ανατεθεί μία άλλη τιμή (στόχος "`NsgInDomain(*this)`").

Συνήθως, για την αντιμετώπιση δύσκολων και μεγάλων προβλημάτων, επιβάλλεται να ορίσουμε τους δικούς μας στόχους, σαν τους `NsgLabeling` και `NsgInDomain`. Ο σκοπός είναι να γίνει η αναζήτηση πιο αποδοτική, μέσω πιο εύστοχων επιλογών, που προκύπτουν από ευριστικές συναρτήσεις.

3.6.3 Ο αλγόριθμος ικανοποίησης στόχων

Σε αυτήν την παράγραφο παρουσιάζεται ο αλγόριθμος ικανοποίησης στόχων. Αρχικά, ο προγραμματιστής που χρησιμοποιεί τη βιβλιοθήκη, έχει θέσει κάποιους στόχους. Όλοι αυτοί οι στόχοι μπαίνουν σε μία στοίβα με όνομα "stackAnd"· όλοι οι στόχοι αυτής της στοίβας θα πρέπει να ικανοποιηθούν.

Ο αλγόριθμος χειρίζεται, όπως είναι φυσικό, διαφορετικά τους μετα-στόχους, από ό,τι τους κοινούς στόχους. Κάθε φορά που συναντά έναν στόχο-AND, βάζει τους δύο υπο-στόχους του, στη στοίβα stackAnd.

Τα αποτελέσματα της εκτέλεσης ενός στόχου-OR είναι λίγο πιο πολύπλοκα, αφού αν αποτύχει ο ένας υπο-στόχος του, ο επιλυτής θα πρέπει να επαναφέρει τα πάντα στην κατάσταση που ήταν πριν την εκτέλεσή του και στη συνέχεια να εκτελέσει τον εναλλακτικό υπο-στόχο. Για να το καταφέρουμε αυτό, έχουμε φτιάξει μία στοίβα stackOr. Κάθε φορά που εκτελείται ένας στόχος-OR, στη στοίβα αυτή προστίθεται ένα στοιχείο, στο οποίο υπάρχουν τα τρέχοντα πεδία όλων των μεταβλητών, η τρέχουσα στοίβα stackAnd με τους στόχους που δεν έχουν ακόμα ικανοποιηθεί, καθώς και ένας δείκτης "delayedGoal" που δείχνει στον πρώτο στόχο, από το προηγούμενο στοιχείο της stackOr που δεν έχει ικανοποιηθεί. Δηλαδή, ο delayedGoal δείχνει σε έναν στόχο που ανήκει σε μία stackAnd κάποιου προηγούμενου στοιχείου της stackOr.[12] Επίσης στο στοιχείο της stackOr μπαίνει και ένας δείκτης στον εναλλακτικό υπο-στόχο του στόχου-OR, έτσι ώστε, αν χρειαστεί να οπισθοδρομήσουμε, να τον εκτελέσουμε και να συνεχίσουμε κανονικά την αναζήτηση.

Παραπάνω, αναφέρθηκε ότι σε κάθε στοιχείο της στοίβας stackOr υπάρχουν τα πεδία όλων των μεταβλητών, όπως ήταν τη στιγμή που μπήκε στη stackOr το συγκεκριμένο στοιχείο. Στην πράξη, δεν αποθηκεύουμε κάθε φορά τα πεδία όλων των μεταβλητών, αλλά αποθηκεύουμε μόνο τα πεδία που θα αλλάξουν. Δηλαδή σε κάθε νέο στοιχείο της stackOr, υπάρχει ένας αρχικά άδειος πίνακας κατακερματισμού. Κάθε φορά που πάμε να αφαιρέσουμε κάποια τιμή από το πεδίο μιας μεταβλητής, ελέγχεται σε σταθερό χρόνο αν στον πίνακα κατακερματισμού υπάρχει το πεδίο αυτής της μεταβλητής. Αν δεν υπάρχει, τότε αποθηκεύεται σε αυτόν και *ύστερα* αφαιρείται η τιμή. Έτσι, γλιτώνουμε αρκετό χρόνο και μνήμη, αφού αποθηκεύουμε τα απαραίτητα.

```
function SOLVE
    while time limit has not been reached do
        if stackOr.top.stackAnd is not empty then
            CurrGoal ← stackOr.top.stackAnd.pop()
        else
            CurrGoal ← stackOr.top.delayedGoal
            stackOr.top.delayedGoal ← get the next goal of
                    stackAnd of the stackOr frame that 'delayedGoal'
```

[12] Όταν τελειώσουμε με την ικανοποίηση των στόχων της stackAnd στην οποία δείχνει ο delayedGoal, τότε στον δείκτη αυτό ανατίθεται η τιμή του delayedGoal του στοιχείου της στοίβας stackOr στην οποία έδειχνε ο delayedGoal. Δηλαδή, καθώς αυξάνουμε τον δείκτη delayedGoal, αυτός μπορεί να οριστεί αναδρομικά από «προηγούμενους» δείκτες delayedGoal στη στοίβα stackOr.

```
                points to or (if there is not such a goal) get the
                next delayed goal of that stackOr frame
        end if

        if CurrGoal is an AND-goal then
            stackOr.top.stackAnd.push( 2nd sub-goal of CurrGoal )
            stackOr.top.stackAnd.push( 1st sub-goal of CurrGoal )
        else if CurrGoal is an OR-goal then
            stackOr.push( 2nd sub-goal of CurrGoal )
            Initialize stackOr.top.delayedGoal
            stackOr.top.stackAnd.push( 1st sub-goal of CurrGoal )
        else
            NewGoal ← result of CurrGoal's execution

            if TwoConsistent() = false then
                if BackTrack() = false then
                    return false
                end if
            end if

            if NewGoal ≠ NIL then
                stackOr.top.stackAnd.push( NewGoal )
            else if stackOr.top.stackAnd is empty
                                and stackOr.top.delayedGoal = NIL then
                return true                                    ▷ Success
            end if
        end if
    end while
    return false
end function
```

Με τη BackTrack οπισθοδρομούμε στο προηγούμενο «σημείο επιλογής». Η TwoConsistent αφορά *μόνο το δεύτερο κομμάτι* (γραμμές 8–13) του αλγορίθμου AC-5-Naxos-Final (σελ. 43). Δηλαδή τρέχουμε το «ελαφρύ» κομμάτι του AC-5, εκείνο που χρησιμοποιεί μόνο τις LocalArcCons και όχι τις ArcCons. Αυτό είναι δυνατόν να γίνει, αφού, καθώς κατεβαίνουμε το δένδρο αναζήτησης, έχουμε τη δυνατότητα να αφαιρούμε μόνο τιμές από τα πεδία. Συνεπώς, με τις κατάλληλες κλήσεις της LocalArcCons είναι δυνατόν να

φέρουμε το δίκτυο περιορισμών σε 2-συνέπεια.

Με τη SOLVE βρίσκουμε μία λύση του προβλήματος. Αν επιθυμούμε να βρούμε την επόμενη λύση, κάνουμε BACKTRACK και ξανακαλούμε τη SOLVE.

3.7 Περιορισμένες μεταβλητές

Η κλάση που υλοποιεί μία περιορισμένη μεταβλητή συνθέτεται, εκτός των άλλων, από την κλάση για το πεδίο τιμών της. Η τελευταία κλάση περιλαμβάνει δύο ακέραιους που παριστάνουν τα άκρα του πεδίου και έναν πίνακα από bit. Αν το i-οστό bit του πίνακα είναι ενεργοποιημένο, αυτό θα πει ότι το πεδίο περιέχει την τιμή `min_dom` $+ i$, όπου `min_dom` η ελάχιστη τιμή του πεδίου.

Αρχικά ο πίνακας έχει μηδενικό μήκος, αφού για να περιγράψουμε το πεδίο μιας μεταβλητής, χρειαζόμαστε μόνο την ελάχιστη και τη μέγιστη τιμή του. Ο πίνακας από bit θα δημιουργηθεί αν αφαιρέσουμε κάποια *ενδιάμεση* τιμή από το πεδίο. Συνεπώς, είναι καλή πρακτική να προσπαθούμε να αφαιρούμε τιμές από τα άκρα των πεδίων τιμών προς τα «μέσα» και όχι αντιστρόφως.

Ο πίνακας υλοποιήθηκε με τη βοήθεια του τύπου `deque`[13] της Standard Template Library (STL) της C++. Ο τύπος αυτός υλοποιεί ένα υβρίδιο πίνακα και διπλά συνδεδεμένης λίστας: είναι ένας πίνακας «κομματιασμένος» στη μνήμη. Δηλαδή είναι ένας πίνακας στον οποίο μπορούμε να προσθαφαιρούμε στοιχεία στα άκρα του, σε σταθερό χρόνο, πλεονέκτημα που δεν έχει ο απλός πίνακας, αλλά χαρακτηρίζει τη λίστα. Ο τύπος αυτός χρησιμοποιήθηκε κατά κόρον στις υλοποιήσεις μας, όπου χρειάστηκε να φτιάξουμε πίνακες.

[13] Προφέρεται «ντεκ».

Κεφάλαιο 4

Κατάρτιση ωρολογίου προγράμματος

Έχω ακούσει ότι ένας από τους λόγους της μακροζωίας των ρωμαϊκών γεφυρών είναι ότι οι σχεδιαστές τους έπρεπε να στέκονται κάτω από αυτές την πρώτη φορά που χρησιμοποιούνταν. Ίσως να έφτασε η ώρα για να θέσουμε έναν παρόμοιο κανόνα και στον τομέα του λογισμικού.

— Henry Baker

Για να προσεγγίσουμε το πρόβλημα της κατάρτισης ωρολογίου προγράμματος στη γενική του εκδοχή, είναι ανάγκη να συγκεντρώσουμε τις προδιαγραφές που έχουν τεθεί για αυτό και έπειτα να προχωρήσουμε σε μια υλοποίηση που θα τις καλύπτει. Ανάμεσα σε αυτά τα δύο βήματα, είναι απαραίτητο να δούμε πιο συστηματικά το πρόβλημα, ορίζοντάς το μαθηματικά.

4.1 Γενικευμένο πρόβλημα ανάθεσης τιμών

Η κατηγορία προβλημάτων χρονοπρογραμματισμού αφορά προβλήματα αναζήτησης που απαντώνται ακόμα και στην καθημερινότητά μας. Π.χ. ένας αθλητής σχεδιάζει για όλη τη χρονιά το πότε θα κάνει προπονήσεις, την ημερήσια διάρκειά τους, σε ποιους τομείς θα δώσει βάρος κάθε φορά και σε ποιες διοργανώσεις θα συμμετάσχει. Στο ξεκίνημα της μέρας κάθε νοικοκυρά κατασκευάζει, ίσως και ασυνείδητα, ένα πρόγραμμα για να φέρει έγκαιρα σε πέρας όλες τις δουλειές του σπιτιού, π.χ. καθαριότητα, μαγείρεμα κ.λπ. Γενικότερα, κάθε πρόβλημα που έχει σαν στόχο την αντιστοίχιση ενός συνόλου από χρονικές περιόδους

(με χαρακτηριστικά την ώρα έναρξης ή τέλους τους και τη διάρκειά τους) σε ένα σύνολο από δραστηριότητες ονομάζεται *πρόβλημα χρονοπρογραμματισμού* (scheduling problem) [9].

Μια άλλη κατηγορία είναι τα *προβλήματα ανάθεσης* (assignment-type problem – ATP) που διατυπώνονται ως εξής: Έστω ότι έχουμε n οντότητες, m πόρους και ένα σύνολο από K δευτερεύοντες περιορισμούς. Κάθε πόρος είναι μοναδιαίος, δηλαδή μπορεί να ανατεθεί σε μία αποκλειστικά οντότητα. Το ζητούμενο είναι σε καθεμία οντότητα να εκχωρηθεί *ένας* ξεχωριστός πόρος, έτσι ώστε να ικανοποιούνται οι K δευτερεύοντες περιορισμοί. Αν υπάρχουν πολλές λύσεις, επιλέγεται αυτή που ελαχιστοποιεί τη συνάρτηση κόστους που θα έχει οριστεί. Το *γενικευμένο πρόβλημα ανάθεσης* (generalized ATP – GATP) είναι όμοιο με το ATP με τη διαφορά ότι κάθε οντότητα μπορεί να απαιτεί *και παραπάνω από έναν* πόρους [30].

Π.χ. ένα ATP είναι το σερβίρισμα m πιάτων με φαγητό (πόροι) σε n πελάτες ενός εστιατορίου (οντότητες). Σε κάθε πελάτη-οντότητα μπορεί να εκχωρηθεί μόνο ένας πόρος, επομένως αυτοί που επιθυμούν και ένα δεύτερο πιάτο θα δυσαρεστηθούν! Αν το πρόβλημα γίνει GATP, τότε το εμπόδιο αυτό δεν υφίσταται, εφόσον βέβαια επαρκούν τα πιάτα-πόροι.

Θα ορίσουμε ένα συγκεκριμένο τύπο προβλήματος χρονοπρογραμματισμού για να τον απεικονίσουμε σε ένα GATP. Έστω ότι έχουμε n δραστηριότητες, καθεμία από τις οποίες πρέπει να συνδεθεί με μία χρονική περίοδο p_i του συνόλου $P = \{p_i \mid 1 \leq i \leq m_p\}$. Οι χρονικές περίοδοι του P δεν αλληλεπικαλύπτονται. Κάθε δραστηριότητα απαιτεί a πόρους, έναν από κάθε πεπερασμένο σύνολο Q_i, $1 \leq i \leq a$. Συνεπώς το GATP πρόβλημα που προκύπτει αφορά n δραστηριότητες-«οντότητες», που πρέπει να συνδεθούν με a πόρους η καθεμία, από το σύνολο $Q \times P$, με $Q = \bigcup_i Q_i$. Μέσα στους δευτερεύοντες περιορισμούς θα υπάρχουν και οι εξής δύο:

1. Οι πόροι οι οποίοι θα εκχωρούνται σε μία δραστηριότητα θα αφορούν την ίδια χρονική περίοδο. Δηλαδή αν (q_1, p_1) και (q_2, p_2) δύο πόροι για την ίδια δραστηριότητα, τότε $p_1 = p_2$.

2. Κάθε δραστηριότητα θα πρέπει να αντιστοιχίζεται με κάποιο στοιχείο κάθε συνόλου Q_i, $1 \leq i \leq a$. Με άλλα λόγια, για μια δεδομένη δραστηριότητα και για $1 \leq i \leq a$, θα υπάρχει $q \in Q_i$ έτσι ώστε ο πόρος (q, p), $p \in P$ να εκχωρείται σε αυτήν.

Π.χ. έστω ότι έχουμε στη διάθεσή μας για μια ημέρα ένα λεωφορείο και 3 οδηγούς. Θέλουμε να προγραμματίσουμε n δρομολόγια διάρκειας μιας ώρας. Αν εν-

διαφερόμαστε να μάθουμε πώς αυτά θα κατανεμηθούν στους οδηγούς και στον χρόνο, δημιουργούμε ένα GATP όπως το παραπάνω με $P = \{p_1, p_2, \ldots, p_{24}\}$ (όπου p_i η i-οστή ώρα της ημέρας), $a = 1$ και $Q = Q_1 = \{d_1, d_2, d_3\}$ (όπου d_i ο i-οστός οδηγός).[1] Συνεπώς καθεμία δραστηριότητα θα συνδεθεί με έναν οδηγό και μία χρονική περίοδο (σύνολο $Q \times P$).

Τι γίνεται όμως αν σε κάθε δρομολόγιο πρέπει να βάλουμε και έναν εισπράκτορα από 4 διαθέσιμους; Το παραπάνω πρόβλημα θα αλλάξει και έτσι θα έχουμε το σύνολο $Q_2 = \{e_1, e_2, e_3, e_4\}$ (όπου e_i ο i-οστός εισπράκτορας), για να γίνει το $Q = Q_1 \cup Q_2$ και το $a = 2$. Τελικά, κάθε δραστηριότητα θα αντιστοιχιστεί με δύο στοιχεία του $Q \times P$ (που θα αφορούν την ίδια χρονική περίοδο): το (d_i, p_k) και το (e_j, p_k).

Αν επιθυμούμε για κάποιο δρομολόγιο x να έχουμε για παράδειγμα 2 εισπράκτορες, δηλώνουμε ρητά ότι για το x χρειαζόμαστε $3 = 1 + 2$ πόρους και προσθέτουμε έναν δευτερεύοντα περιορισμό που αναφέρει ότι το x απαιτεί 2 εισπράκτορες. Ακόμα, στους δευτερεύοντες περιορισμούς μπορούμε να εντάξουμε κανόνες που υπάρχουν π.χ. στις συμβάσεις των εργαζόμενων, στη νομοθεσία, τις οδηγίες του κατασκευαστή του λεωφορείου κ.λπ. Ένας δευτερεύων περιορισμός θα μπορούσε να είναι ότι κανένας οδηγός και εισπράκτορας δεν μπορεί να δουλεύει πάνω από 8 ώρες. Ένας άλλος δευτερεύων περιορισμός πιθανόν να αφορά το λεωφορείο και να ορίζει ότι μετά από 15 ώρες λειτουργίας της μηχανής του, θα σταματά για να γίνει ανεφοδιασμός.

Σε κάθε (G)ATP θα πρέπει να ορίζεται μια συνάρτηση κόστους, για να ελαχιστοποιηθεί κατά την επίλυση του προβλήματος. Σε αυτό το σημείο έχουμε τη μεγαλύτερη διαφοροποίηση ανάμεσα στα GATP προβλήματα χρονοπρογραμματισμού. Ας δούμε μερικά παραδείγματα: Όταν έχουμε να κάνουμε με βάρδιες, ο στόχος συνήθως είναι η ισοκατανομή στις ημέρες των ωρών εργασίας κάθε εργαζόμενου. Μια άλλη περίπτωση είναι όταν μια κατασκευαστική εταιρεία αποσκοπεί στο να περατώσει ένα έργο το συντομότερο δυνατόν, και μια άλλη όταν δίνει περισσότερο βάρος στο χρηματικό κόστος και έτσι επιζητά τη μείωση των υπερωριών και την ελάττωση της συνεχόμενης χρήσης των μηχανημάτων.

4.2 Προδιαγραφές ενός προβλήματος

Πριν μοντελοποιήσουμε το πρόβλημα κατάρτισης ωρολογίου προγράμματος είναι σωστό να προσπαθήσουμε να συγκεντρώσουμε όλες τις απαιτήσεις που έχουν τα περισσότερα ανώτατα εκπαιδευτικά ιδρύματα από αυτό. Αν καταγράψουμε τις

[1] Επειδή $a = 1$, το πρόβλημα εκτός από GATP είναι και ATP.

ανάγκες μιας σχολής και μόνο, κινδυνεύουμε να έχουμε μια μονόπλευρη άποψη του προβλήματος και οι όποιες λύσεις προτείνουμε πιθανόν να είναι ανεφάρμοστες σε άλλα ιδρύματα [31, 9, 32, 14].

4.2.1 Τα δεδομένα

Ο καμβάς πάνω στον οποίο σχεδιάζεται το ωρολόγιο πρόγραμμα δεν είναι άλλος από τον χρόνο. Ένα πρόγραμμα αφορά συνήθως την κατανομή των μαθημάτων μέσα σε μία εβδομάδα· σπανιότερα αφορά ένα δεκαπενθήμερο. Η γενικότερη περίπτωση με την οποία θα ασχοληθούμε περιλαμβάνει D τον αριθμό ημέρες. Κάθε ημέρα κατατέμνεται σε H διδακτικές περιόδους, που συχνά αποκαλούνται και «ώρες». Η ακριβής χρονική διάρκεια μιας διδακτικής περιόδου δεν μας ενδιαφέρει, καθώς αφορά κάθε εκπαιδευτικό ίδρυμα χωριστά — σε κάποια ιδρύματα είναι 1 ώρα και σε άλλα 45 λεπτά.

Εκτός από τα D και H θα πρέπει να είναι γνωστά και τα σύνολα των αιθουσών και των καθηγητών. Κάθε αίθουσα χαρακτηρίζεται από τη χωρητικότητά της, ενώ για κάποιον καθηγητή μπορεί να υπάρξει η απαίτηση κάποιων ορίων στις συνεχόμενες ώρες, στις ώρες ανά ημέρα, ή στις ημέρες ανά εβδομάδα, που μπορεί το πολύ να διδάξει.

Πολλές φορές για λόγους απλότητας, στη βιβλιογραφία για τα προβλήματα ωρολογίων προγραμμάτων, ένα μάθημα, αναφορικά με τη διάρκειά του, έχει σαν μοναδικό χαρακτηριστικό τον αριθμό των διδακτικών περιόδων που καταλαμβάνει στο πρόγραμμα. Και έτσι αν για παράδειγμα το μάθημα «Ανάλυση» έχει αριθμό διδακτικών περιόδων ίσο με 5, οιαδήποτε κατανομή αυτών είναι νόμιμη. Συνεπώς την 1^{η} ημέρα του προγράμματος θα μπορούσαμε να έχουμε 3 ώρες Ανάλυση και τη 2^{η} ημέρα 2 ώρες. Όμως θα υπήρχε και η περίπτωση την 1^{η} ημέρα να γίνουν 4 ώρες Ανάλυση και τη 2^{η} ημέρα 1 ώρα, ή ακόμα να γίνουν και οι 5 ώρες σε μία ημέρα μόνο! Ωστόσο, στα εκπαιδευτικά ιδρύματα τέτοια ζητήματα σχεδόν ποτέ δεν ρυθμίζονται τυχαία. Για κάθε μάθημα δεν γνωστοποιείται απλά ο αριθμός των διδακτικών περιόδων του, αλλά ο αριθμός των διαλέξεών του στο πρόγραμμα και η χρονική διάρκεια καθεμίας από αυτές. Π.χ. για την Ανάλυση θα πρέπει να μπορούμε να δηλώσουμε ότι έχουμε 2 διαλέξεις, η μία των 3 και η άλλη των 2 διδακτικών περιόδων. Δηλαδή τα μαθήματα σπάνε σε διαλέξεις.

Για κάθε μάθημα δηλώνονται οι καθηγητές οι οποίοι θα το διδάξουν, καθώς και οι αίθουσες στις οποίες μπορεί να γίνει, με σειρά προτίμησης. Όσον αφορά τις αίθουσες δηλώνονται και οι προϋποθέσεις που πρέπει να πληρούν για να διδαχθεί σε αυτές το μάθημα, όπως χωρητικότητα και εξοπλισμός. Θα πρέπει

επίσης να δίδεται η δυνατότητα προκαθορισμού των διδακτικών περιόδων και των αιθουσών στις οποίες θα λάβουν χώρα οι διαλέξεις ενός συγκεκριμένου μαθήματος.

Επιπλέον θα πρέπει να υποστηρίζεται η έννοια του ζεύγους ταυτόχρονων μαθημάτων (με ισάριθμες διαλέξεις), που χρησιμεύει σε περιπτώσεις διδασκαλίας με τηλεσυνδιάσκεψη. Οι διαλέξεις των μαθημάτων ενός τέτοιου ζεύγους ξεκινούν την ίδια στιγμή. Π.χ. αν έχουμε τα μαθήματα «Ανάλυση» στην αίθουσα «Α» και «Ανάλυση-κλώνος» στην αίθουσα «Β» και τα εντάξουμε σε ένα ζεύγος, θα πετύχουμε την παράλληλη διδασκαλία τους. Ο καθηγητής του μαθήματος θα μπορεί να βρίσκεται στην αίθουσα «Α» ενώ οι φοιτητές στη «Β» θα μπορούν και αυτοί να τον παρακολουθούν εξ αποστάσεως, μέσα από μια οθόνη.

Είναι σημαντικό επίσης να παρέχεται η δυνατότητα ομαδοποίησης των μαθημάτων, ανάλογα με τις ανάγκες των φοιτητών και των κανονισμών ενός ιδρύματος. Π.χ. τα μαθήματα ομαδοποιούνται πολλές φορές ανά εξάμηνα, καθώς οι φοιτητές συχνά παρακολουθούν όλα τα μαθήματα του εξαμήνου τους. Καλό είναι επίσης να υπάρχει τρόπος διάκρισης των ομάδων ανάλογα με τη βαρύτητά τους.

Τέλος, είναι απαραίτητη η καταγραφή των διδακτικών περιόδων στις οποίες δεν θα είναι διαθέσιμη μια συγκεκριμένη αίθουσα, ένας καθηγητής, ή ένα μάθημα.

4.2.2 Απαιτήσεις

Τα παραπάνω δεδομένα είναι αρκετά για να φωτίσουν τις περισσότερες πλευρές προβλημάτων ποικίλων ιδιαιτεροτήτων. Αυτή η γενικότητα του προβλήματος, κάθε άλλο παρά μας εμποδίζει να θέσουμε κάποιες αυστηρές απαιτήσεις που ισχύουν σε όλες τις περιπτώσεις.

Ας δούμε λοιπόν κάθε διάλεξη σαν μια δραστηριότητα που απαιτεί κάποιους πόρους για να γίνει:

- Καταρχήν απαιτεί *μία* αίθουσα από αυτές που έχουν προταθεί για το μάθημα, σε χρονική περίοδο που είναι διαθέσιμη. Η χωρητικότητα της αίθουσας θα πρέπει να επαρκεί για το ακροατήριο και ο εξοπλισμός της θα πρέπει να καλύπτει τις ανάγκες του μαθήματος.

- Χρειάζεται *όλους* τους καθηγητές που το διδάσκουν διαθέσιμους. Όπως και να έχει, θα πρέπει πάντοτε να τηρούνται τα όρια για *όλους* τους καθηγητές σχετικά με τις συνεχόμενες ώρες, τις ώρες ανά ημέρα και τις ημέρες ανά εβδομάδα, που μπορεί καθένας από αυτούς το πολύ να διδάξει.

- Αν η διάλεξη αφορά μάθημα που ανήκει σε ομάδα, τότε δεν μπορεί να γίνεται ταυτόχρονα με διαλέξεις της ίδιας ομάδας. Ως εκ τούτου μια ομάδα μπορεί να θεωρηθεί και αυτή σαν πόρος. Όπως και οι παραπάνω πόροι, είναι αποκλειστικός, δηλαδή σε δεδομένη χρονική στιγμή μπορεί να εκχωρείται σε μία το πολύ δραστηριότητα-διάλεξη.

Κάποιες άλλες απαιτήσεις για κάθε μάθημα, έχουν να κάνουν με το ότι οι διαλέξεις του δεν μπορούν να γίνονται σε περιόδους που δεν είναι διαθέσιμο· ακόμα, δύο διαλέξεις του δεν μπορούν να λάβουν χώρα την ίδια ημέρα. Τέλος, οι διαλέξεις των μαθημάτων ενός ζεύγους ταυτόχρονων μαθημάτων θα ξεκινούν ταυτόχρονα.

Στα δεδομένα και τις απαιτήσεις του προβλήματος απουσιάζει η οντότητα φοιτητής που σε πολλές εργασίες θεωρείται και αυτή σαν ένας πόρος. Η αλήθεια είναι ότι δεν χρειάζεται να έχουμε έναν ακόμα πόρο-φοιτητή για τη διάλεξη, καθώς αυτός «αντιπροσωπεύεται» από τους πόρους αίθουσα και ομάδα μαθημάτων:

- ✓ Ένας φοιτητής μπορεί να βρίσκεται σε μία αίθουσα και σε αυτήν μπορεί να γίνει μία το πολύ διάλεξη. Δηλαδή ο περιορισμός ότι ένας φοιτητής μπορεί να παρακολουθεί το πολύ μία διάλεξη, ισχύει ήδη.

- ✓ Αν θέλουμε τα μαθήματα που επιθυμεί να παρακολουθήσει ένας φοιτητής να μη γίνονται τις ίδιες ώρες, μπορούμε να τα εντάξουμε σε μία ομάδα. Επομένως και αυτή η ανάγκη του φοιτητή καλύπτεται από ήδη υπάρχοντα πόρο. Στην πράξη βέβαια οι ομάδες μαθημάτων φτιάχνονται βάσει των προτιμήσεων πολλών (και όχι μόνο ενός) φοιτητών.

4.2.3 Κριτήρια ποιότητας

Είναι κοινά διαπιστωμένο ότι ένα πρόβλημα κατάρτισης ωρολογίου προγράμματος, αν μπορεί να λυθεί, έχει συνήθως πολλές λύσεις. Παρόλα αυτά κάποιες από τις λύσεις που πληρούν τις προϋποθέσεις της προηγούμενης παραγράφου μπορεί να κριθούν μη αποδεκτές. Ας φανταστούμε για παράδειγμα ένα 5ήμερο πρόγραμμα στο οποίο όλες οι διαλέξεις έχουν στοιβαχτεί μέσα σε 3 ημέρες!

Για να αποφύγουμε τέτοιες περιπτώσεις και, ακόμα καλύτερα, για να βρεθεί το βέλτιστο πρόγραμμα μπορούμε να λάβουμε υπόψη κάποια από τα παρακάτω κριτήρια ποιότητας:

1. Ισοκατανομή των διδακτικών περιόδων μέσα στην εβδομάδα, για τα μαθήματα που παρακολουθεί ένας φοιτητής, ή που διδάσκει ένας καθηγητής.

2. Ελαχιστοποίηση των κενών στο πρόγραμμα ενός φοιτητή ή ενός καθηγητή. Αν έχουμε ένα μεγάλο διάλειμμα, π.χ. το μεσημέρι για γεύμα, το οποίο μπορεί να θεωρηθεί σαν ψευδομάθημα, τότε αυτό δεν θα πρέπει να προστίθεται στα κενά.

3. Κάποια μαθήματα για εκπαιδευτικούς λόγους ίσως να πρέπει να γίνονται μετά από κάποια άλλα, που θα λειτουργούν με εισαγωγικό τρόπο. Ας πούμε, το μάθημα «Ηλεκτρονική-Εργαστήριο», καλό θα ήταν να γίνεται μετά το μάθημα «Ηλεκτρονική».

4. Όταν τα κτήρια με τις αίθουσες μιας σχολής, βρίσκονται σε αρκετή απόσταση μεταξύ τους, τότε θα μπορούσε να τεθεί ο στόχος της ελαχιστοποίησης των μετακινήσεων ενός φοιτητή ή καθηγητή.

5. Πιθανόν να προταθούν και άλλα κριτήρια, όπως η παιδαγωγική ευστοχία του προγράμματος (π.χ. τα δύσκολα μαθήματα καλό είναι να εναλλάσσονται με τα εύκολα κ.ά.), η αραιή κατανομή των διαλέξεων του ίδιου μαθήματος στην εβδομάδα, ή η ευκολία προσαρμογής του προγράμματος σε απρόβλεπτες αλλαγές (π.χ. πρόσληψη ενός καθηγητή στη θέση ενός άλλου).

Τα κριτήρια 2, 3 και 4 αφορούν τη διαδοχή των μαθημάτων. Όπως φαίνεται είναι δύσκολο, όχι μόνο να συγκεντρώσουμε, αλλά και να μοντελοποιήσουμε μαθηματικά όλα τα κριτήρια, αφού έχουν να κάνουν με πολλούς παράγοντες. Επιπλέον κάθε εκπαιδευτικό ίδρυμα δίνει και μία διαφορετική βαρύτητα σε κάθε κριτήριο. Σε αυτό έγκεινται και οι μεγάλες διαφοροποιήσεις κατά την επίλυση του προβλήματος, που συναντάμε από ίδρυμα σε ίδρυμα.

Στην εργασία αυτή θα ασχοληθούμε με τα κριτήρια 1 και 2 τα οποία θα προσπαθήσουμε να εφαρμόσουμε για τις *ομάδες μαθημάτων*. Και αυτό γιατί πίσω από μία ομάδα μαθημάτων κρύβεται μια μεγάλη μερίδα των φοιτητών. Επομένως τα μαθήματα μίας ομάδας (π.χ. «Μαθήματα Α΄ Εξαμήνου») θα πρέπει να ισοκατανέμονται μέσα στην εβδομάδα και να έχουν το δυνατόν λιγότερα κενά μεταξύ τους.

Όταν βελτιώνουμε ένα πρόγραμμα ως προς ένα κριτήριο, είναι πιθανόν να το κάνουμε χειρότερο ως προς ένα άλλο. Για να κρατήσουμε τις ισορροπίες πρέπει σε κάθε κριτήριο που χρησιμοποιούμε να δώσουμε και από έναν συντελεστή κρισιμότητας. Γνωρίζοντας τη σημασία του καθενός, μπορούμε να τα συγχωνεύσουμε όλα σε ένα κριτήριο, δηλαδή σε *μία* συνάρτηση προς ελαχιστοποίηση. Εξάλλου το γενικευμένο πρόβλημα ανάθεσης (§4.1), δέχεται μόνο *μία* συνάρτηση για ελαχιστοποίηση.

4.3 Μαθηματική μοντελοποίηση

Οι προδιαγραφές του προβλήματος που επιθυμούμε να καλύψουμε, πρέπει να γίνουν πιο σαφείς. Ο δρόμος για να επιτευχθεί αυτό, είναι η μαθηματική τους μοντελοποίηση.

4.3.1 Παράμετροι του προβλήματος

Έστω ότι στο ωρολόγιο πρόγραμμα έχουμε D ημέρες και H ώρες. Ορίζουμε το σύνολο P των διδακτικών περιόδων (ωρών) του ωρολογίου προγράμματος

$$P = \{0, 1, \ldots, D \cdot H - 1\}$$

Ακόμα μας δίνονται το σύνολο των αιθουσών $C \neq \emptyset$, των καθηγητών $T \neq \emptyset$, των μαθημάτων $S \neq \emptyset$ (το "S" προέρχεται από τη λέξη "subject"), των διαλέξεων L και των ομάδων μαθημάτων G, με $g \subseteq S$, $\forall g \in G$. Οι μορφές των παραπάνω συνόλων είναι οι εξής:

$$\begin{aligned} C &= \{c_1,\ c_2,\ \ldots,\ c_{|C|}\} \\ T &= \{t_1,\ t_2,\ \ldots,\ t_{|T|}\} \\ S &= \{s_1,\ s_2,\ \ldots,\ s_{|S|}\} \\ L &= \{\ell_1,\ \ell_2,\ \ldots,\ \ell_{|L|}\} \\ G &= \{g_1,\ g_2,\ \ldots,\ g_{|G|}\} \end{aligned}$$

Κάθε στοιχείο, ανάλογα με το σύνολο στο οποίο ανήκει, έχει τις δικές του ιδιότητες. Για τα $c_i \in C$, $t_i \in T$ και $s_i \in S$ έχουμε αντίστοιχα τα

$$\begin{aligned} \mathrm{nac}(c_i) &\subseteq P \\ \mathrm{nat}(t_i) &\subseteq P \\ \mathrm{nas}(s_i) &\subseteq P \end{aligned}$$

Το σύμβολο "na" προκύπτει από το "not available" και η σχέση αυτή συμβολίζει ότι το αντικείμενο στο οποίο εφαρμόζεται δεν θα είναι διαθέσιμο κάποιες ώρες. Π.χ. το $\mathrm{nat}(t_2) = \{0, 5\}$ συμβολίζει ότι ο καθηγητής t_2 δεν θα είναι διαθέσιμος τις ώρες 0 και 5. Υπάρχουν περιπτώσεις που ένας καθηγητής απαιτεί να υπάρχει κάποιο όριο στις συνεχόμενες ώρες, στις ώρες ανά ημέρα, ή στις ημέρες ανά εβδομάδα, που μπορεί το πολύ να διδάξει. Έτσι ορίζονται οι

αντίστοιχες συναρτήσεις:[2]

$$\begin{aligned} \text{mconthours} &: T \to \{0\} \cup \mathbb{N} \\ \text{mhoursday} &: T \to \{0\} \cup \mathbb{N} \\ \text{mdaysweek} &: T \to \{0\} \cup \mathbb{N} \end{aligned}$$

Αν η τιμή μιας συνάρτησης από τις παραπάνω, για έναν καθηγητή $t_i \in T$, είναι 0, αυτό σημαίνει πως το αντίστοιχο όριο για τον συγκεκριμένο καθηγητή δεν υπάρχει. Π.χ. αν $\text{mdaysweek}(t_2) = 0$, τότε ο t_2 μπορεί να διδάξει ακόμα και όλες τις ημέρες της εβδομάδας.

Όσον αφορά τα μαθήματα, ορίζονται οι παρακάτω σχέσεις με τα άλλα σύνολα

$$\begin{aligned} \emptyset \neq \text{class}(s_i) &\subseteq C \\ \emptyset \neq \text{teach}(s_i) &\subseteq T \end{aligned}$$

για $s_i \in S$. Η πρώτη συμβολίζει τις προτιμώμενες αίθουσες στις οποίες θα λάβουν χώρα οι διαλέξεις του μαθήματος και η δεύτερη τους καθηγητές οι οποίοι το διδάσκουν. Το μάθημα το οποίο αφορά μία διάλεξη, ο αύξων αριθμός της διάλεξης για το μάθημα και η διάρκειά της προκύπτουν αντίστοιχα από τις συναρτήσεις:

$$\begin{aligned} \text{subj} &: L \mapsto S \\ \text{order} &: L \to \mathbb{N} \\ \text{dur} &: L \to \{1, 2, \ldots, H\} \end{aligned}$$

Αν όπου K_i έχουμε το σύνολο των διαλέξεων του μαθήματος s_i:
$K_i = \{\ell_j \mid \ \ell_j \in L \text{ και } \text{subj}(\ell_j) = s_i\}$, τότε για την order πρέπει να ισχύει:
$\forall s_i \in S : \quad \{\text{order}(\ell_j) \mid \ell_j \in L \text{ και } \text{subj}(\ell_j) = s_i\} \ = \ \{1, 2, \ldots, |K_i|\}$. Με την τελευταία σχέση απαιτούμε οι αύξοντες αριθμοί των διαλέξεων να είναι έγκυροι. Αναφορικά με τις ομάδες μαθημάτων, ορίζεται η βαρύτητά τους μέσω της

$$\text{imp} : G \to \mathbb{N}$$

Τέλος, δίδεται το σύνολο των ζευγών ταυτόχρονων μαθημάτων E. Τα στοιχεία του είναι σύνολα δύο μαθημάτων:

$$\{s_i, s_j\} \in E, \quad i \neq j, \ s_i, s_j \in S, \ |K_i| = |K_j|$$

Με την τελευταία ισότητα απαιτούμε τα μαθήματα που θα γίνονται ταυτόχρονα να έχουν τον ίδιο αριθμό διαλέξεων.

[2] Θεωρούμε ότι $\mathbb{N} = \{1, 2, 3, \ldots\}$

Το ζητούμενο είναι να βρούμε τις ώρες στις οποίες θα γίνεται κάθε διάλεξη $\ell_i \in L$, τις οποίες αναπαριστούμε με $x_{i,1}, x_{i,2}, \ldots, x_{i,\,\mathrm{dur}(\ell_i)} \in P$. Στην πραγματικότητα, αρκεί να ξέρουμε το $x_i \equiv x_{i,1}$, αφού οι ώρες στις οποίες γίνεται μια συγκεκριμένη διάλεξη είναι διαδοχικές και η διάρκειά της γνωστή. Κάθε διάλεξη $\ell_i \in L$ θα γίνεται σε μία συγκεκριμένη αίθουσα $cl_i \in C$. Αυτό που τελικά θεωρείται λύση του προβλήματος είναι η εύρεση μιας συνάρτησης

$$\mathrm{sol} : L \to P \times C$$

η οποία απεικονίζει κάθε διάλεξη στην ώρα και στην αίθουσα στις οποίες θα πραγματοποιείται. Θα έχουμε δηλαδή $\forall \ell_i \in L,\ (x_i, cl_i) = \mathrm{sol}(\ell_i)$.

4.3.2 Περιορισμοί

Για να προκύψει η λύση sol θα πρέπει να ικανοποιεί κάποιους περιορισμούς. Έστω ότι έχουμε τα

$$\begin{array}{ll} x_{i,k} & \text{με } (x_i,\, cl_i) = \mathrm{sol}(\ell_i) \text{ και } 1 \le k \le \mathrm{dur}(\ell_i) \\ x_{j,l} & \text{με } (x_j,\, cl_j) = \mathrm{sol}(\ell_j) \text{ και } 1 \le l \le \mathrm{dur}(\ell_j) \end{array}$$

Ορίζουμε ένα βοηθητικό σύνολο F_i που θα περιέχει τις ομάδες μαθημάτων στις οποίες ανήκει το μάθημα s_i:

$$F_i = \{g \mid g \in G,\ s_i \in S \text{ και } s_i \in G\}$$

Ισχύουν τα παρακάτω:

- Καταρχήν οι διδακτικές περίοδοι (ώρες) των διαλέξεων πρέπει να είναι διαδοχικές:
$$x_{i,p+1} = x_{i,p} + 1, \quad 1 \le p \le \mathrm{dur}(\ell_i) - 1 \tag{4.1}$$
- Μία διάλεξη αρχίζει και τελειώνει την ίδια ημέρα!
$$\left\lfloor \frac{x_{i,1}}{H} \right\rfloor = \left\lfloor \frac{x_{i,\,\mathrm{dur}(\ell_i)}}{H} \right\rfloor \tag{4.2}$$
Εξάλλου από την παράσταση $\left\lfloor \frac{x}{H} \right\rfloor$ παίρνουμε την ημέρα στην οποία υπάρχει η διδακτική περίοδος x.
- Οι διαλέξεις ενός μαθήματος γίνονται σε διαφορετικές ημέρες:
$$\mathrm{subj}(\ell_i) = \mathrm{subj}(\ell_j) \overset{i \neq j}{\Longrightarrow} \left\lfloor \frac{x_i}{H} \right\rfloor \neq \left\lfloor \frac{x_j}{H} \right\rfloor \tag{4.3}$$

– Η αίθουσα στην οποία θα γίνεται μια διάλεξη ℓ_i πρέπει να ανήκει στον κατάλογο με τις προτιμώμενες αίθουσες του αντίστοιχου μαθήματος:

$$cl_i \in \text{class}(\text{subj}(\ell_i)) \tag{4.4}$$

– Δύο ταυτόχρονες διαλέξεις δεν μπορούν:

1. να γίνονται στην ίδια αίθουσα:

$$x_{i,k} = x_{j,l} \overset{i \neq j}{\Longrightarrow} cl_i \neq cl_j \tag{4.5}$$

2. να μοιράζονται κάποιον από τους καθηγητές που τις παραδίδουν:

$$x_{i,k} = x_{j,l} \overset{i \neq j}{\Longrightarrow} \text{teach}(\text{subj}(\ell_i)) \cap \text{teach}(\text{subj}(\ell_j)) = \emptyset \tag{4.6}$$

3. να ανήκουν στην ίδια ομάδα μαθημάτων:

$$x_{i,k} = x_{j,l} \overset{i \neq j}{\Longrightarrow} F_{\text{subj}(\ell_i)} \cap F_{\text{subj}(\ell_j)} = \emptyset \tag{4.7}$$

– Μία διάλεξη δεν μπορεί να γίνει μία ώρα στην οποία δεν είναι διαθέσιμο ένα (τουλάχιστον) από τα παρακάτω:

1. η αίθουσά της:

$$x_{i,k} \notin \text{nac}(cl_i) \tag{4.8}$$

2. ένας από τους καθηγητές οι οποίοι διδάσκουν το αντίστοιχο μάθημα:

$$x_{i,k} \notin \text{nat}(t), \quad \forall t \in \text{teach}(\text{subj}(\ell_i)) \tag{4.9}$$

3. το ίδιο το μάθημα:

$$x_{i,k} \notin \text{nas}(\text{subj}(\ell_i)) \tag{4.10}$$

– Για τα ζεύγη ταυτόχρονων μαθημάτων έχουμε ότι τα οι διαλέξεις τους θα αρχίζουν την ίδια στιγμή:

$$\left.\begin{array}{l} \{s_i, s_j\} \in E \\ \text{subj}(\ell_i) = s_p \\ \text{subj}(\ell_j) = s_q \\ \text{order}(\ell_i) = \text{order}(\ell_j) \end{array}\right\} \Longrightarrow x_i = x_j \tag{4.11}$$

– Για κάθε καθηγητή $t \in T$ πρέπει να γίνουν σεβαστά τα όρια που έχουν τεθεί όσον αφορά:

1. τις συνεχόμενες ώρες που μπορεί το πολύ να διδάξει: Ορίζουμε το σύνολο T_t^{ch} με όλους τους αριθμούς των συνεχόμενων ωρών που διδάσκει ο t μέσα στην εβδομάδα. Να σημειωθεί ότι αν π.χ. $5 \in T_t^{\text{ch}}$, τότε σίγουρα και $0, 1, \ldots, 4 \in T_t^{\text{ch}}$.
   Έχουμε $m \in T_t^{\text{ch}}$ αν και μόνο αν ισχύει ένα από τα εξής:
   (i) $m \geq 1$ και $\exists\, x_{i_1,k_1}, x_{i_2,k_2}, \ldots, x_{i_m,k_m} \in P$:
   $$\begin{cases} \ell_{i_p} \in L, \quad 1 \leq p \leq m \\ x_{i_1,k_1} = x_{i_2,k_2} + 1 = \cdots = x_{i_m,k_m} + m - 1 \\ \left\lfloor \dfrac{x_{i_1,k_1}}{H} \right\rfloor = \left\lfloor \dfrac{x_{i_2,k_2}}{H} \right\rfloor = \cdots = \left\lfloor \dfrac{x_{i_m,k_m}}{H} \right\rfloor \\ t \in \displaystyle\bigcap_{1 \leq p \leq m} \text{teach}(\text{subj}(\ell_{i_p})) \end{cases}$$
   (ii) $m = 1$ και $\exists x_{i,k} \in P$: $\ell_i \in L$ και $t \in \text{teach}(\text{subj}(\ell_i))$
   (iii) $m = 0$
   Ο περιορισμός λοιπόν που ισχύει για το T_t^{ch} είναι:
   $$\text{mconthours}(t) \neq 0 \Longrightarrow \max\{T_t^{\text{ch}}\} \leq \text{mconthours}(t) \qquad (4.12)$$

2. τις ώρες ανά ημέρα που μπορεί το πολύ να διδάξει: Ορίζουμε το σύνολο $T_{t,d}^{\text{h}}$ με τις διαφορετικές διδακτικές περιόδους στις οποίες ο t παραδίδει διαλέξεις μέσα στην ημέρα d:
   $$T_{t,d}^{\text{h}} = \{x_{i,j} \mid \ x_{i,j} \in P,\ \left\lfloor \frac{x_{i,j}}{H} \right\rfloor = d,\ \ell_i \in L \text{ και } t \in \text{teach}(\text{subj}(\ell_i))\}$$
   Τότε πρέπει να ισχύει:
   $$\text{mhoursday}(t) \neq 0 \Longrightarrow \max_{0 \leq d \leq D-1} \{|T_{t,d}^{\text{h}}|\} \leq \text{mhoursday}(t) \qquad (4.13)$$

3. τις ημέρες ανά εβδομάδα που μπορεί το πολύ να διδάξει: Ορίζουμε το σύνολο T_t^{w} με τις ημέρες μέσα στην εβδομάδα στις οποίες ο t παραδίδει κάποια διάλεξη:
   $$T_t^{\text{w}} = \{d \mid \ x_{i,j} \in P,\ \left\lfloor \frac{x_{i,j}}{H} \right\rfloor = d,\ \ell_i \in L \text{ και } t \in \text{teach}(\text{subj}(\ell_i))\}$$

Τότε υπάρχει ο περιορισμός:

$$\text{mdaysweek}(t) \neq 0 \Longrightarrow |T_t^{\text{w}}| \leq \text{mdaysweek}(t) \tag{4.14}$$

4.3.3 Βελτιστοποίηση λύσης

Όλοι οι προηγούμενοι περιορισμοί είναι αυστηροί και πρέπει να ισχύουν, για να είναι το ωρολόγιο πρόγραμμα σωστό και συνεπές με τις προδιαγραφές που έχουν διατυπωθεί. Θα πρέπει όμως να εξασφαλιστεί με κάποιον τρόπο και η ποιότητα του προγράμματος, με άλλα λόγια να βρεθεί το καλύτερο πρόγραμμα από τα συνεπή με τους παραπάνω ορισμούς προγράμματα που μπορούν να καταρτιστούν. Έτσι κατασκευάζουμε μια συνάρτηση εκτίμησης η οποία αποτελεί τη σύνδεση δύο κριτηρίων/παραστάσεων. Το βέλτιστο πρόγραμμα παράγεται όταν η συνάρτηση αυτή ελαχιστοποιηθεί.

Ορίζουμε το σύνολο $G_{g,d}$ που περιέχει τις διδακτικές περιόδους των μαθημάτων της ομάδας μαθημάτων $g \in G$, για την ημέρα d:

$$G_{g,d} = \{x_{i,j} \mid \ x_{i,j} \in P,\ \left\lfloor \frac{x_{i,j}}{H} \right\rfloor = d,\ \ell_i \in L \text{ και } \text{subj}(\ell_i) \in g\}$$

Το σύνολο $G_{g,d}^p$ περιέχει τα στοιχεία του $G_{g,d}$ αυξημένα κατά 1:

$$G_{g,d}^p = \{x + 1 \mid \ x \in G_{g,d}\}$$

Έχουμε λοιπόν δύο κριτήρια τα οποία έχουν συγκεκριμένα βάρη όταν συνδυαστούν γραμμικά στη συνάρτηση εκτίμησης:

1. *Ισοκατανομή των διδακτικών περιόδων μέσα στην εβδομάδα, για τις διαλέξεις μιας ομάδας μαθημάτων.* Για κάθε ομάδα $g \in G$ παίρνουμε τη διαφορά του μέγιστου ημερήσιου αριθμού διδακτικών περιόδων των διαλέξεων της ομάδας, από τον αντίστοιχο ελάχιστο. Π.χ. αν έχουμε τρεις διδακτικές ημέρες την εβδομάδα και για μία ομάδα μαθημάτων έχουμε διαλέξεις που διαρκούν 4 διδακτικές περιόδους την $1^η$ ημέρα, 7 τη $2^η$ και 5 την $3^η$ τότε $\max\{4,7,5\} - \min\{4,7,5\} = 3$. Αθροίζουμε όλες αυτές τις διαφορές, αφού έχουν πρώτα πολλαπλασιαστεί με τη βαρύτητα της κάθε ομάδας, για $g \in G$:

$$\sum_{g \in G} \text{imp}(g) \cdot \left(\max_{0 \leq d \leq D-1} \{|G_{g,d}|\} - \min_{0 \leq d \leq D-1} \{|G_{g,d}|\} - \textsc{Greedy}(g) \right) \tag{4.15}$$

Στην παραπάνω παράσταση, υπάρχει ένας καινούργιος όρος, το "GREEDY(g)". Ουσιαστικά, πρόκειται για την τιμή $\max_{0\le d\le D-1}\{|G_{g,d}|\} - \min_{0\le d\le D-1}\{|G_{g,d}|\}$, που θα παίρναμε εφόσον κατασκευάζαμε —με έναν παντελώς πρόχειρο τρόπο— το ωρολόγιο πρόγραμμα, με τη βοήθεια *άπληστου* (greedy) αλγορίθμου *εξισορρόπησης φορτίου* (load balancing) [33]. Ο αλγόριθμος είναι ο εξής:[3]

---

function GREEDY(g)
  $i \leftarrow 0$
  for each $\ell \in L$, with subj(ℓ) $\in g$ **do**
    Durs[i] $\leftarrow$ dur(ℓ)
    $i \leftarrow i + 1$
  end for
  SORTDESCENDING(Durs)
  for $d \leftarrow 0$ **to** $D - 1$ **do**
    DayLoad[d] $\leftarrow 0$
  end for
  for $i \leftarrow 0$ **to** |Durs| $- 1$ **do**
    $d \leftarrow$ argmin{DayLoad}
    DayLoad[d] $\leftarrow$ DayLoad[d] + Durs[i]
  end for
  return $\left(\max\limits_{0\le d\le D-1}\{\text{DayLoad}\} - \min\limits_{0\le d\le D-1}\{\text{DayLoad}\}\right)$
end function

Αποδεικνύεται ότι ο αλγόριθμος αυτός προσεγγίζει στη χειρότερη περίπτωση το ήμισυ της «ποιότητας» της βέλτιστης λύσης, όσον αφορά το πρόβλημα εξισορρόπησης φορτίου πάντα· εξάλλου το τελευταίο είναι NP-πλήρες. Όσον αφορά την εκτίμηση που δίνει ο αλγόριθμος, για το πρόβλημα κατάρτισης ωρολογίου προγράμματος που εξετάζουμε, είναι προφανές ότι δεν μπορεί να είναι αξιόπιστη, καθώς δεν λαμβάνεται υπόψη κανένας περιορισμός από αυτούς που έχουν τεθεί — π.χ. ότι διαλέξεις του ίδιου μαθήματος δεν μπορούν να γίνονται την ίδια ημέρα. Τότε ποιος ο λόγος που χρησιμοποιούμε αυτήν τη μέθοδο;

Η σκοπιμότητα αυτής της μεθόδου έγκειται στη δίκαιη μεταχείριση όλων των ομάδων. Π.χ. έστω η ομάδα μαθημάτων A, με τέσσερις δίωρες

[3] Με argmin$\{a_i\}$, συμβολίζουμε το μικρότερο i για το οποίο παίρνουμε το ελάχιστο a_i. Με τη SORTDESCENDING, ταξινομούμε τον πίνακα Durs σε φθίνουσα σειρά.

διαλέξεις και η ομάδα μαθημάτων B, με μία εξάωρη και μία δίωρη διάλεξη. Θα δούμε την περίπτωση στην οποία υπάρχουν δύο ημέρες στο ωρολόγιο πρόγραμμα ($D = 2$).

Εκ των πραγμάτων, για την ομάδα B μπορούμε να πετύχουμε στην καλύτερη περίπτωση

$$\max_{0 \le d \le D-1} \{|G_{\mathsf{B},d}|\} - \min_{0 \le d \le D-1} \{|G_{\mathsf{B},d}|\} = 6 - 2 = 4,$$

βάζοντας π.χ. την εξάωρη διάλεξη την 1^{η} ημέρα και τη δίωρη διάλεξη τη 2^{η} ημέρα.

Όμως, αυτό το "4" δεν είναι σωστό να «μπει στο ίδιο τσουβάλι» με ένα $\max_{0 \le d \le D-1}\{|G_{\mathsf{A},d}|\} - \min_{0 \le d \le D-1}\{|G_{\mathsf{A},d}|\} = 4$, για την ομάδα A. Αυτό θα σήμαινε ότι θα έχουμε τρεις δίωρες διαλέξεις (της A) τη μία ημέρα και μία δίωρη διάλεξη την άλλη ημέρα. Ωστόσο θα μπορούσαμε να πετύχουμε $\max_{0 \le d \le D-1}\{|G_{\mathsf{A},d}|\} - \min_{0 \le d \le D-1}\{|G_{\mathsf{A},d}|\} = 4 - 4 = 0$, μοιράζοντας καλύτερα τις διαλέξεις στις ημέρες.

Συνοψίζοντας, η παράσταση "GREEDY(g)" εισήχθη με στόχο να λαμβάνονται υπόψη πιθανές εγγενείς ανισοκατανομές των διαλέξεων των ομάδων μαθημάτων — όπως π.χ. της ομάδας B.

2. *Κενά (εντός μιας ημέρας) μεταξύ των διαλέξεων της ίδιας ομάδας μαθημάτων.* Οι διαλέξεις μιας ομάδας, ανάλογα και με τη βαρύτητά της, παρακολουθούνται όλες από μεγάλο αριθμό φοιτητών. Π.χ. οι φοιτητές του Α′ εξαμήνου παρακολουθούν συνήθως όλα τα βασικά μαθήματα του εξαμήνου τους. Έτσι η τιμή της παρακάτω παράστασης θα πρέπει να γίνει όσο το δυνατόν μικρότερη:

$$\sum_{g \in G} \mathrm{imp}(g) \cdot \left(\sum_{\substack{0 \le d \le D-1, \\ G_{g,d} \neq \emptyset}} \big(\max\{G^{p}_{g,d}\} - \min\{G_{g,d}\} \big) - \sum_{\substack{\ell_i \in L, \\ \mathrm{subj}(\ell_i) \in g}} \mathrm{dur}(\ell_i) \right) \tag{4.16}$$

Η διαφορά $\max\{G^{p}_{g,d}\} - \min\{G_{g,d}\}$ ισούται με τον αριθμό των διδακτικών ωρών που χωρίζουν την πρώτη διάλεξη της ομάδας g, από το τέλος της τελευταίας διάλεξης της ίδιας ομάδας, για την ημέρα d. Π.χ. στον Πίνακα 4.1 η διαφορά είναι 7. Αν από αυτή τη διαφορά αφαιρέσουμε τη συνολική διάρκεια των διαλέξεων της g την ημέρα d, παίρνουμε τον αριθμό των κενών. Για τον Πίνακα 4.1 παίρνουμε $7 - (2 + 3) = 2$.

	d
1	
2	
3	$x_{i_1,1}$
4	$x_{i_1,2}$
5	
6	
7	$x_{i_2,1}$
8	$x_{i_2,2}$
9	$x_{i_2,3}$

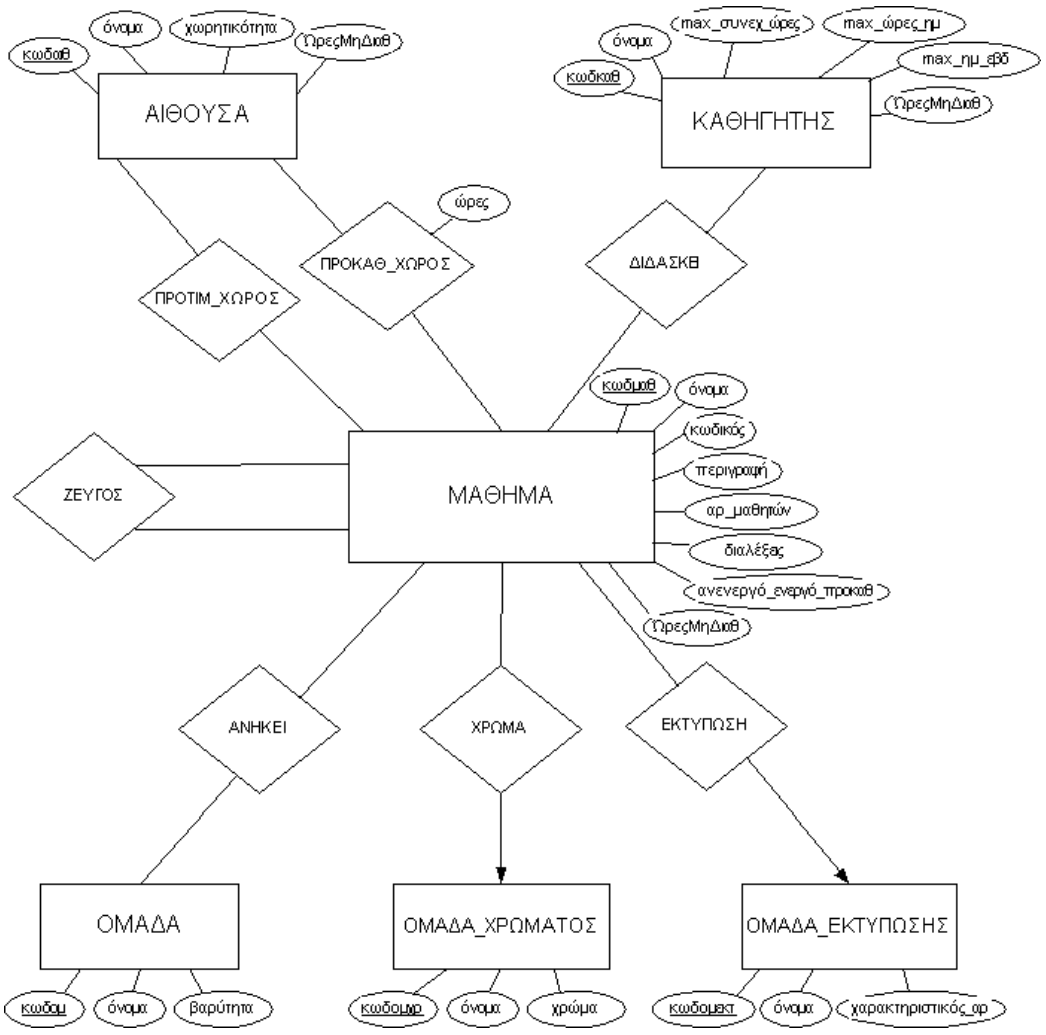
Πίνακας 4.1: Οι διαλέξεις μιας ομάδας μαθημάτων την ημέρα d

4.4 Διάγραμμα οντοτήτων-συσχετίσεων

Κοιτώντας από τη σκοπιά των βάσεων δεδομένων, ο τρόπος που προσεγγίζουμε το πρόβλημα ωρολογίου προγράμματος φαίνεται στο διάγραμμα οντοτήτων-συσχετίσεων (Σχήμα 4.1). Ένας σημαντικός περιορισμός που υπάρχει στις συσχετίσεις αφορά στην αυτοσυσχέτιση ΖΕΥΓΟΣ: δύο μαθήματα που αποτελούν ένα ζεύγος ταυτόχρονων μαθημάτων πρέπει να είναι διαφορετικά μεταξύ τους και κάθε ζεύγος πρέπει να είναι μοναδικό — τα ζεύγη (ΜΑΘ_Α, ΜΑΘ_Β) και (ΜΑΘ_Β, ΜΑΘ_Α) δεν μπορούν να υπάρχουν ταυτόχρονα.

4.5 Υλοποίηση με προγραμματισμό με περιορισμούς

Σε ένα πρόβλημα χρονοπρογραμματισμού-GATP (§4.1), εκχωρούμε σε κάθε δραστηριότητα κάποιους αναγκαίους πόρους, για μία συγκεκριμένη χρονική περίοδο $p \in P$. Μιλώντας τώρα με όρους προβλημάτων ικανοποίησης περιορισμών, μία δραστηριότητα μπορεί να θεωρηθεί σαν μία μεταβλητή $V_i \in P$. Για λόγους απλότητας στην αναπαράσταση των δεδομένων στον υπολογιστή, θεωρούμε ότι το P είναι της μορφής $\{0, 1, \ldots, |P| - 1\}$.



Σχήμα 4.1: Διάγραμμα οντοτήτων-συσχετίσεων

4.5.1 Μοναδιαίος πόρος

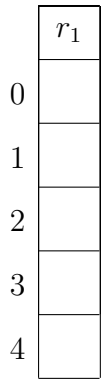
Μπορούμε να φανταστούμε κάθε *μοναδιαίο πόρο* σαν έναν πίνακα με $|P|$ στοιχεία (Σχήμα 4.2(α′)). Τα στοιχεία αυτού του πίνακα, είναι στην ουσία υποδοχές, καθεμία από τις οποίες θα συνδέεται με μία το πολύ δραστηριότητα (Σχήμα 4.2(β′)). Με τη σύνδεση αυτή καθορίζεται αυτόματα και η χρονική περίοδος κατά την οποία θα γίνεται η δραστηριότητα.

Μετά την παραπάνω θεωρητική παράγραφο, θα περάσουμε σε θέματα υλοποίησης με προγραμματισμό με περιορισμούς που είναι συμβατός με τον επιλυτή. Καταρχήν αν έχουμε n δραστηριότητες, δημιουργούμε ισάριθμες μεταβλητές με πεδίο τιμών το P:

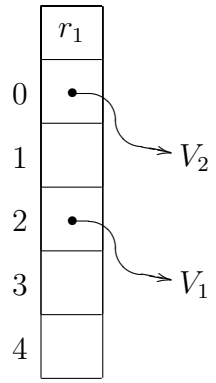
```
NsIntVarArray  var;
for (i=0;  i<N;  ++i)
    var.push_back( NsIntVar(pm, 0, np-1) );
```

Όπου `pm` ο διαχειριστής του προβλήματος (`NsProblemManager`) και `np` $= |P|$. Ο στόχος μας είναι να βρούμε μία νόμιμη ανάθεση τιμών για τις `var`, δηλαδή να μάθουμε σε ποια χρονική περίοδο θα γίνεται κάθε δραστηριότητα. Ας δούμε τώρα το πώς υλοποιείται ένας πόρος. Έχουμε την εξής κλάση:

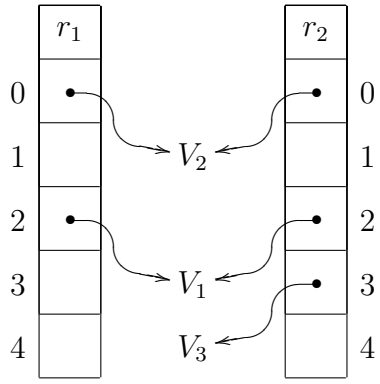
```
class  CUnarySimple  {
  protected:
    NsProblemManager&  pm;
    NsIntVarArray  vActivities;
    int  nperiods;

  public:
    CUnarySimple (NsProblemManager& pm_init, int nperiods_init)
      :  pm(pm_init), nperiods(nperiods_init)   {  }

    void  add (NsIntVar& vStart)
    {
        vActivities.push_back( vStart );
    }

    void  close (void)
    {
            pm.add( NsAllDiff(vActivities) );
```



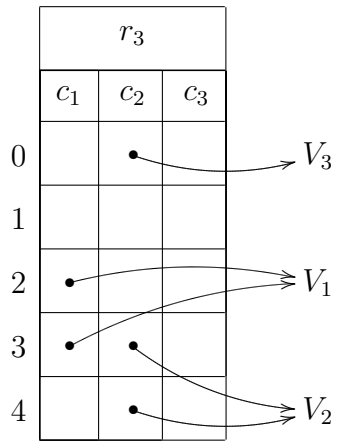
(α′) Ένας πόρος r_1 για $|P| = 5$



(β′) Διασύνδεση των δραστηριοτήτων V_1 και V_2 με τον πόρο r_1



(γ′) Οι δραστηριότητες των V_1 και V_2 απαιτούν και τον r_1 και τον r_2



(δ′) Ένας πολλαπλός πόρος

Σχήμα 4.2: Πόροι και αναθέσεις τους

```
    }

    void  exclude (int p)
    {
        for (NsIndex i = 0;  i < vActivities.size();  ++i)
            vActivities[i].remove(p);
    }
};
```

Η CUnarySimple έχει στενή σχέση με τις δραστηριότητες (μεταβλητές var) οι οποίες τη χρειάζονται σαν πόρο. Με τη μέθοδο add() αντιγράφουμε όλες αυτές τις μεταβλητές στον πίνακα vActivities. Όταν τελειώσουμε με τις add(), καλούμε την close() για να εξασφαλίσουμε ότι όλες οι μεταβλητές του vActivities θα έχουν διαφορετικές τιμές· δηλαδή οι δραστηριότητες που ζητούν τον ίδιο πόρο δεν θα γίνουν την ίδια στιγμή, έτσι ώστε να μπορεί να εκχωρηθεί σε αυτές ο πόρος. Με την exclude(int p) δηλώνουμε ότι την περίοδο p ο πόρος δεν είναι διαθέσιμος. Για να παρουσιάσουμε με κώδικα το πρόβλημα που υπάρχει στο Σχήμα 4.2(β′), δηλαδή το ότι δύο μεταβλητές απαιτούν τον πόρο r_1, μπορούμε να γράψουμε τις εξής εντολές:

```
int  nperiods = 5;

// Αρχικοποίηση των μεταβλητών-δραστηριοτήτων //
NsIntVarArray  var;
for (i=0;  i<2;  ++i)
     var.push_back( NsIntVar(pm, 0, nperiods-1) );

// Σύνδεση των 2 μεταβλητών με τον πόρο r1 //
CUnarySimple  r1(pm, nperiods);
r1.add(var[0]);
r1.add(var[1]);
r1.close();
```

Στην πράξη το παραπάνω ισοδυναμεί με την απευθείας κλήση της
`pm.add( NsAllDiff(var) );`
Και αυτό επειδή πρόκειται για μια απλοϊκή περίπτωση που αφορά έναν μόνο πόρο.

Όπως αναφέρθηκε και στην §4.1, όταν μία δραστηριότητα απαιτεί περισσότερους του ενός πόρους, οι πόροι αυτοί εκχωρούνται για *την ίδια* χρονική

περίοδο. Στο Σχήμα 4.2(γ′) παρουσιάζεται μία λύση σε ένα τέτοιο πρόβλημα ανάθεσης πόρων, το οποίο διατυπώνεται με κώδικα ως εξής:

```
NsIntVarArray  var;
for (i=0;  i<3;  ++i)
     var.push_back( NsIntVar(pm, 0, nperiods-1) );

CUnarySimple  r1(pm, nperiods);
r1.add(var[0]);
r1.add(var[1]);
r1.close();

CUnarySimple  r2(pm, nperiods);
r2.add(var[0]);
r2.add(var[1]);
r2.add(var[2]);
r2.close();
```

Με μικρές τροποποιήσεις στη `CUnarySimple` μπορούμε να την κάνουμε να υποστηρίζει δραστηριότητες με χρονική διάρκεια και μεγαλύτερη της μίας περιόδου. Θα πειράξουμε λοιπόν μόνο τη μέθοδο `add()`:

```
1 void  CUnary::add (NsIntVar& vStart, int duration)
2 {
3     vActivities.push_back( vStart );
4     NsIntVar&  vActivityStart = vActivities.back();
5     for (int i=1;  i<duration;  ++i)
6         vActivities.push_back( vActivityStart + i );
7 }
```

Σε σχέση με την `CUnarySimple::add()`, η `CUnary::add()` παίρνει ένα επιπλέον όρισμα: τη διάρκεια της δραστηριότητας. Οι υποδοχές ενός πόρου που αφορούν την ίδια δραστηριότητα, είναι προφανώς διαδοχικές, πράγμα που εξασφαλίζεται στην 6^{η} γραμμή του παραπάνω κώδικα.

4.5.2 Πολλαπλοί πόροι

Στον μοναδιαίο πόρο έχουμε την «ευχέρεια» να επιλέξουμε το *πότε* θα εκχωρηθεί ένας πόρος. Σε έναν *πολλαπλό πόρο*, εκτός από το πότε επιλέγουμε και *ποιον*

πόρο θα δεσμεύσουμε. Π.χ. όταν έχουμε ένα λεωφορείο με το οποίο πρέπει να διεκπεραιωθούν κάποια δρομολόγια, το όχημα αυτό αποτελεί έναν μοναδιαίο πόρο. Αν υπάρχει ένας μόνο οδηγός, τότε και αυτός είναι μοναδιαίος πόρος. Αν υπάρχουν περισσότεροι οδηγοί διαθέσιμοι, τότε έχουμε έναν πολλαπλό πόρο, τους οδηγούς.

Με άλλα λόγια η έννοια του πολλαπλού πόρου έρχεται να υποστηρίξει τη διαζευκτική απαίτηση «κάποιου τυχαίου» πόρου από ένα σύνολο, για μία χρονική περίοδο. Π.χ. μία διάλεξη ενός μαθήματος χρειάζεται κάποια αίθουσα από το σύνολο των αιθουσών για να γίνει. Έστω λοιπόν ότι έχουμε δύο δίωρες διαλέξεις που αντιπροσωπεύονται από τις μεταβλητές V_1, V_2 και μία μονόωρη που αντιστοιχεί στη V_3. Επίσης υπάρχουν τρεις αίθουσες διαθέσιμες, οι c_1, c_2 και c_3, οι οποίες όλες μαζί αποτελούν έναν πολλαπλό όρο, τον r_3. Μία νόμιμη εκχώρηση του r_3 στις μεταβλητές φαίνεται στο Σχήμα 4.2(δ′). Αν κάναμε το λάθος και φτιάχναμε έναν μοναδιαίο πόρο για κάθε αίθουσα και συνδέαμε τους πόρους αυτούς με τις μεταβλητές, τότε αντί για διάζευξη, θα είχαμε το αντίθετο, τη σύζευξη: Κάθε διάλεξη θα απαιτούσε να της εκχωρηθούν όλες οι αίθουσες!

Σε έναν πολλαπλό πόρο (π.χ. με αίθουσες) δεν αναζητούμε μόνο το χρονικό διάστημα εκείνο, κατά το οποίο θα εκχωρηθεί ένας πόρος, αλλά θέλουμε να γνωρίζουμε και το ποιος «πραγματικός» πόρος θα εκχωρηθεί (π.χ. σε ποια αίθουσα θα γίνει η διάλεξη). Έτσι οι μεταβλητές (του προβλήματος ικανοποίησης περιορισμών) δεν θα αφορούν μόνο την ώρα έναρξης μιας δραστηριότητας (`var`), αλλά και το ποιον πόρο θα δεσμεύσουν. Όταν ένας πολλαπλός πόρος συνδέεται με τις μεταβλητές `var`, τότε απαραίτητα δημιουργούμε και τις μεταβλητές `res`, έτσι ώστε να ξέρουμε ποιος πόρος `res[i]` —που ανήκει στον εν λόγω πολλαπλό πόρο— έχει εκχωρηθεί στη `var[i]`. Είναι προφανές ότι οι μεταβλητές των πινάκων `var` και `res` είναι ισάριθμες, καθώς και ότι για κάθε πολλαπλό πόρο που συνδέεται με τις `var` χρειάζεται να ορίσουμε τον αντίστοιχο πίνακα μεταβλητών «`res`».

Για την υλοποίηση του πολλαπλού πόρου εκμεταλλευόμαστε τον υπερφορτωμένο, ως προς τις `NsIntVar`, τελεστή "`*`":

```
class CMultiple :  public CUnary  {
  private:
    int  multiplicity;

  public:
    CMultiple (NsProblemManager& pm_init,
```

```
                int multiplicity_init, int nperiods_init)
      :  CUnary(pm_init,nperiods_init),
         multiplicity(multiplicity_init)   {  }

    void  add (NsIntVar& vStart, NsIntVar& vRes,
                int duration)
    {
        vActivities.push_back( vRes*nperiods + vStart );
        NsIntVar&  vActivityStart = vActivities.back();
        for (int i=1;  i<duration;  ++i)
            vActivities.push_back( vActivityStart + i );
    }

    void  exclude (int p, int r)
    {
        for (NsIndex i = 0;  i<vActivities.size();  ++i)
            vActivities[i].remove(r*nperiods + p);
    }
};
```

Ο δισδιάστατος πίνακας με τις υποδοχές (όπως στο Σχήμα 4.2(δ′)) «σειριοποιήθηκε» σε ένα μονοδιάστατο, μέσω του τύπου:

```
vActivities[i] = res[i]*nperiods + var[i]
```

Για κάθε τιμή του `res[i]`, έχουμε `nperiods` υποδοχές, όπως στο σχήμα που αναφέρθηκε. Η `CMultiple` κληρονομεί τα χαρακτηριστικά της κλάσης `CUnary`. Ωστόσο υπάρχει μια διαφοροποίηση στη συνάρτηση κατασκευής, η οποία παίρνει ένα επιπλέον όρισμα, που δεν είναι άλλο από τον αριθμό των πόρων (`multiplicity`)· οι πόροι, όπως είναι λογικό, αριθμούνται από το 0 μέχρι το `multiplicity-1`. Επίσης η `exclude()` έχει τροποποιηθεί, έτσι ώστε να ορίζει μη διαθέσιμες ώρες *για συγκεκριμένο πόρο.*

Τώρα μπορούμε να διατυπώσουμε με κώδικα το εξής πρόβλημα: Διαθέτουμε ένα λεωφορείο (μοναδιαίος πόρος `bus`) και έχουμε να διαλέξουμε ανάμεσα από 5 οδηγούς, το ποιος θα το οδηγήσει σε καθένα από τα 3 δρομολόγια διάρκειας 7, 2 και 4 ωρών.

```
int  nroutes  = 3;
int  ndrivers = 5;
```

```
NsIntVarArray  var_route;
for (i=0;  i<nroutes;  ++i)
     var_route.push_back( NsIntVar(pm, 0, nhours-1) );
NsIntVarArray  res_driver;
for (i=0;  i<ndrivers;  ++i)
     res_driver.push_back( NsIntVar(pm, 0, ndrivers-1) );

CUnary  bus(pm, nhours);
bus.add(var_route[0], 7);
bus.add(var_route[1], 2);
bus.add(var_route[2], 4);
bus.close();

CMultiple  drivers(pm, ndrivers, nhours);
drivers.add(var_route[0], res_driver[0], 7);
drivers.add(var_route[1], res_driver[1], 2);
drivers.add(var_route[2], res_driver[2], 4);
drivers.close();
```

Η πλειάδα `var_route[i]-res_driver[i]` τελικά θα φανερώσει το πότε θα αρχίσει το i-οστό δρομολόγιο και το ποιος οδηγός θα είναι στο τιμόνι.

4.6 Εφαρμογή στο πρόβλημα ωρολογίου προγράμματος

Η έννοια του πόρου που αναλύθηκε προηγουμένως, είναι μείζονος σημασίας για τα προβλήματα ωρολογίων προγραμμάτων. Στην πράξη, για να λύσουμε ένα τέτοιο πρόβλημα ακολουθούμε τα εξής βήματα:

- Δημιουργούμε δύο πίνακες μεταβλητών, τον `var` και τον `classvar`, με μέγεθος όσο ο αριθμός των διαλέξεων. Το ζευγάρι `var[i]-classvar[i]` δείχνει σε ποια ώρα θα αρχίσει η i-οστή διάλεξη και σε ποια αίθουσα θα λάβει χώρα. Για κάθε διάλεξη πρέπει να έχει φυλαχτεί κάπου η πληροφορία της διάρκειάς της.

- Ένα μάθημα αποτελείται από μία ή περισσότερες διαλέξεις. Επομένως συνδέεται με κάποια ζευγάρια `var[i]-classvar[i]`, τα οποία θα πρέπει να υπόκεινται στους περιορισμούς του μαθήματος.

Πρώτα από όλα, αν έχουμε `nrooms` αίθουσες, η `classvar[i]` θα ισούται εξ ορισμού με `[0..nrooms-1]`, αφού θα ορίζεται με την εντολή:
`classvar.push_back( NsIntVar(pm, 0, nrooms-1) );`
Αν όμως δεν επιθυμούμε η `i`-οστή διάλεξη να γίνεται στην αίθουσα `k`, τότε θα πρέπει `classvar[i]=[0..k-1 k+1..nrooms-1]`. Αυτό μπορεί να επιτευχθεί απλά με την εντολή `classvar[i].remove(k);`

- Όμοια, οι προδιαγραφές του προβλήματος επιτρέπουν στον χρήστη να δηλώσει ότι «στις τάδε ώρες το συγκεκριμένο μάθημα δεν θα είναι διαθέσιμο». Ας πάρουμε τη γενική περίπτωση που ένα μάθημα δεν μπορεί να γίνει σε κάποια ώρα `x`. Καλούμε για κάθε διάλεξη `i` του μαθήματος τη `var[i].remove(x)`. Είναι όμως αυτό αρκετό; Η απάντηση είναι «ναι», αν η διάρκεια της διάλεξης `i` είναι μία ώρα. Γιατί αν είναι μεγαλύτερη, τότε η `x` θα πρέπει να αποκλειστεί όχι μόνο σαν ώρα έναρξης της διάλεξης (`var[i]`), αλλά και σαν τιμή των διδακτικών ωρών της ίδιας διάλεξης που έπονται (`var[i]+1`,`var[i]+2`,...,`var[i]+dur[i]-1`, όπου `dur[i]` η διάρκεια της διάλεξης). Για να γίνει το παραπάνω κατανοητό, ας πάρουμε το παράδειγμα μιας δίωρης διάλεξης, η οποία πρέπει να τοποθετηθεί σε ένα ωρολόγιο πρόγραμμα 2 ημερών με 4 ώρες/ημέρα. Έχουμε `days=2`, `hours=4` και `var[0]=[0..7]` (`7=days*hours-1`). Αν θέλουμε να αποκλείσουμε την ώρα 3 (4^{η} ώρα της $1^{\eta\varsigma}$ ημέρας) και καλέσουμε την `var[i].remove(3)`, θα πάρουμε `var[0]=[0..2 4..7]`. Βλέπουμε ότι η διάλεξη είναι πιθανόν να αρχίσει την ώρα 2 και έτσι η δεύτερη διδακτική περίοδός της να γίνει την ώρα 3! Άρα έπρεπε να είχαμε αποκλείσει και τη 2 σαν τιμή της `var[0]`. Ο παρακάτω κώδικας κάνει σωστά τη δουλειά (αποκλείει τις τιμές `[x-(dur[i]-1)..x]`):

```
for (h=0;  h<dur[i];  ++h)
    var[i].remove(x - h);
```

- Για κάθε διάλεξη διάρκειας μεγαλύτερης της μίας ώρας, τίθεται ένας ακόμα περιορισμός. Προκύπτει από τον σειριακό τρόπο που έχουμε περάσει τις ώρες της εβδομάδας στο πεδίο της `var[i]`:[4]

$$\texttt{var[i]=[}\underbrace{\texttt{0*H..1*H-1}}_{1^{\eta}\text{ ημέρα}}\ \underbrace{\texttt{1*H..2*H-1}}_{2^{\eta}\text{ ημέρα}}\ \ldots\ \underbrace{\texttt{(D-1)*H..D*H-1}}_{\text{τελευταία ημέρα}}\texttt{]}$$

[4] Όπου `D` ο αριθμός των ημερών και `H` ο αριθμός των ωρών του ωρολογίου προγράμματος.

Σε αυτήν την αναπαράσταση υπάρχει ο κίνδυνος μία διάλεξη με `dur[i]` >1 να σπάσει σε δύο ημέρες, με άλλα λόγια να αρχίσει στο τέλος μιας ημέρας και να τελειώσει στην αρχή της επόμενης! Π.χ. μία δίωρη διάλεξη `var[i]` θα μπορούσε να πάρει τη τιμή `H-1`, δηλαδή να αρχίζει την τελευταία ώρα της πρώτης ημέρας και έτσι η δεύτερη ώρα της να γίνεται την επόμενη ημέρα. Η λύση σε αυτό το ζήτημα είναι να αφαιρεθούν από την `var[i]`, με παρόμοιο τρόπο όπως στην προηγούμενη περίπτωση με τις μη διαθέσιμες ώρες ενός μαθήματος, για κάθε ημέρα `d` οι προβληματικές τιμές `[(d+1)*H-(dur[i]-1)..(d+1)*H-1]`:

```
for (d=0;  d<days;  ++d)  {
    for (h=(d+1)*hours-dur[i]+1;  h<(d+1)*hours;  ++h)
        var[i].remove(h);
}
```

- Σε ένα δένδρο αναζήτησης πολλές φορές παρουσιάζονται συμμετρίες ανάμεσα σε διαφορετικά μονοπάτια. Δύο συμμετρικά μονοπάτια αφορούν τα ίδια δεδομένα και φτάνουν πάντα στο ίδιο αποτέλεσμα: είτε καταλήγουν στην ίδια λύση, είτε και τα δύο δεν έχουν λύση. Συνεπώς αρκεί να εξετάσουμε μόνο ένα από τα δύο και με τον τρόπο αυτόν να κλαδέψουμε το δένδρο αναζήτησης. Η απομάκρυνση των συμμετριών σε όλο τους το φάσμα, κάθε άλλο παρά εύκολη υπόθεση είναι, καθώς αποτελεί ένα από τα ανοικτά θέματα της Τεχνητής Νοημοσύνης. Ωστόσο αυτό δεν πρέπει να μας αποθαρρύνει από το να προσπαθούμε να εντοπίζουμε όσες το δυνατό περισσότερες περιπτώσεις συμμετρίας στα προβλήματα που επιλύουμε και να τις εξαφανίζουμε.

  Τώρα στο πρόβλημά μας υπάρχει η εξής συμμετρία: Έστω ότι για δύο διαλέξεις `i` και `j` ενός μαθήματος, *που διαρκούν το ίδιο*, έχουμε `var[i]` $= a$ και `var[j]` $= b$. Ο επιλυτής θεωρεί αυτή την περίπτωση διαφορετική με εκείνη που `var[i]` $= b$ και `var[j]` $= a$. Όμως προφανώς οι δύο περιπτώσεις ταυτίζονται όσον αφορά το τελικό πρόγραμμα. Με τον περιορισμό `pm.add(var[i] < var[j])` αποφεύγουμε αυτήν τη διπλή λύση.

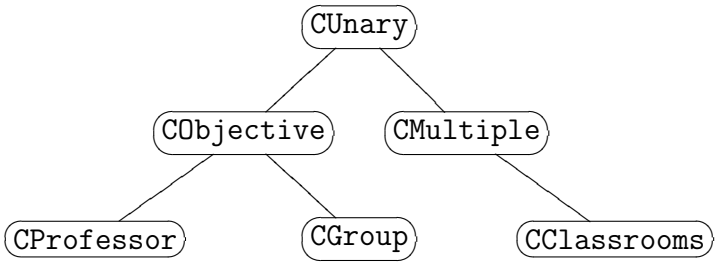
- Όσον αφορά στα ζεύγη ταυτόχρονων μαθημάτων, οι διαλέξεις τους ξεκινούν την ίδια ώρα. Αν οι μεταβλητές του πρώτου μαθήματος ενός ζεύγους είναι οι `var[i]`, `var[i+1]`, ..., `var[i+nl-1]` και του δεύτερου μαθήματος οι `var[j]`, `var[j+1]`, ..., `var[j+nl-1]`, όπου `nl` ο αριθμός των

διαλέξεων του κάθε μαθήματος του ζεύγους, τότε ο περιορισμός διατυπώνεται ρητά:

```
for (k=0;  k<nl;  ++k)
    pm.add(var[i+k] == var[j+k]);
```

- Οι καθηγητές και οι ομάδες μαθημάτων αντιμετωπίζεται σαν μοναδιαίοι πόροι και οι διαλέξεις που τους αφορούν συνδέονται μαζί τους μέσω της `add()` (§4.5.1), π.χ. `prof[k].add(var[i],dur[i])`, όπου `prof[k]` ο καθηγητής-πόρος για τη διάλεξη `i`. Όταν ένα μάθημα γίνεται από δύο καθηγητές (ή ανήκει σε περισσότερες από μία ομάδες), συνδέεται και με τους δύο πόρους, όπως είναι λογικό· απαιτεί ταυτόχρονα και τους δύο.

- Επειδή για κάθε διάλεξη χρειαζόμαστε «μία κάποια» αίθουσα από τις διαθέσιμες, ο αντίστοιχος πόρος είναι ένας και είναι πολλαπλός (§4.5.2)· περιλαμβάνει όλες τις αίθουσες. Για να τον συνδέσουμε με μία διάλεξη καλούμε την `rooms.add(var[i],classvar[i],dur[i])`, όπου `rooms` ο εν λόγω πόρος. Πριν κάνουμε τη σύνδεση αυτή, θα πρέπει να ελέγξουμε για το αν η χωρητικότητα της αίθουσας επαρκεί για τις ανάγκες του μαθήματος.



Σχήμα 4.3: Ιεραρχία των κλάσεων του προβλήματος

- Η ιεραρχία των κλάσεων του προβλήματος, που αρχίζει να χτίζεται σιγά-σιγά, φαίνεται στο Σχήμα 4.3. Η μόνη καινούργια κλάση που υπάρχει στο σχήμα, είναι η `CObjective`, με την οποία θα ασχοληθούμε αργότερα. Εν ολίγοις, η `CObjective` παίζει ρόλο όσον αφορά την ποιότητα της λύσης. Όταν αναφερόμαστε σε πανεπιστήμια, το κριτήριο ποιότητας μιας λύσης είναι η ισοκατανομή και η έλλειψη κενών μεταξύ των ωρών για τις ομάδες μαθημάτων (`CGroup`). Όταν αναφερόμαστε σε σχολεία, δεν έχουμε εν

γένει ομάδες μαθημάτων, αλλά ο στόχος είναι να βελτιώσουμε το προσωπικό πρόγραμμα του κάθε καθηγητή (`CProfessor`). Ως εκ τούτου, οι δύο κλάσεις `CGroup` και `CProfessor` παράγονται από τη `CObjective`.

4.6.1 Περισσότερο σύνθετοι περιορισμοί

Για τους καθηγητές, εκτός από το να διδάσκουν ένα το πολύ μάθημα ανά ώρα, υπάρχουν ακόμα τρεις πιο σύνθετοι περιορισμοί: α) επιβολή ορίου στον αριθμό των ωρών ανά ημέρα που διδάσκει ένας συγκεκριμένος καθηγητής, β) επιβολή ορίου στον αριθμό των ημερών ανά εβδομάδα στις οποίες διδάσκει ένας συγκεκριμένος καθηγητής και γ) επιβολή ορίου στον αριθμό των συνεχόμενων ωρών που διδάσκει ένας συγκεκριμένος καθηγητής. Η διατύπωσή τους στον επιλυτή, θα διευκολυνθεί αν δημιουργήσουμε έναν δισδιάστατο πίνακα `vOccupiedHour`, αποτελούμενο από περιορισμένες μεταβλητές με πεδία `[0..1]`. Η μεταβλητή `vOccupiedHour[`d`][`h`]` θα είναι `1` αν ο καθηγητής κάνει μάθημα την ημέρα d και ώρα h και `0` αν είναι ελεύθερος.

Για να πάρουμε όμως τον πίνακα `vOccupiedHour`, θα πρέπει να τον συνδέσουμε με τον πίνακα `vActivities` του πόρου (κλάση `CUnary`) με τον οποίο παριστάνουμε τον καθηγητή. Για τον σκοπό αυτό, θα χρησιμοποιήσουμε έναν «ενδιάμεσο» πίνακα, με όνομα `vTimetable`, τον οποίον κατασκευάζουμε «αντιστρέφοντας» τον `vActivities` (βλ. κ. §3.4.14):

```
vTimetable = NsInverse(vActivities, days*hours-1);
```

Μία μεταβλητή `vTimetable[`$d \cdot H + h$`]` θα περιέχει τις θέσεις των μεταβλητών στον πίνακα `vActivities`, οι οποίες παριστάνουν δραστηριότητες του καθηγητή (δηλαδή, μαθήματα που διδάσκει), οι οποίες θα γίνονται την ημέρα d και ώρα h. Αν δεν υπάρχει καμία δραστηριότητα για τη συγκεκριμένη ημέρα και ώρα, τότε η `vTimetable[`$d \cdot H + h$`]` θα έχει την ειδική τιμή `-1`. Επομένως για κάθε στοιχείο του πίνακα `vOccupiedHour`, θα έχουμε τον μετα-περιορισμό:

```
vOccupiedHour[d][h] = ( vTimetable[d*hours + h] >= 0 );
```

Δηλαδή η "boolean" περιορισμένη μεταβλητή `vOccupiedHour[d][h]` θα είναι `0`, μόνο αν η αντίστοιχη τιμή του πίνακα `vTimetable` είναι `-1`. Στη συνέχεια εξετάζουμε έναν-έναν τους τρεις περιορισμούς.

Όριο στον αριθμό των ωρών ανά ημέρα

Δημιουργούμε για τον συγκεκριμένο καθηγητή, έναν μονοδιάστατο πίνακα ονόματι `vHoursPerDay`, με το `vHoursPerDay[`d`]` να αναπαριστά τον αριθμό των

ωρών που ο καθηγητής διδάσκει την ημέρα d, αφού θα ισούται με το άθροισμα των μεταβλητών `vOccupiedHour[`d`][`h`]`, $\forall h$. Στη συνέχεια επιβάλλουμε κάθε στοιχείο του `vHoursPerDay` να είναι μικρότερο από έναν δοθέντα ακέραιο `maxHoursPerDay`.

```
vHoursPerDay[d] = NsSum(vOccupiedHour[d]);
pm.add( vHoursPerDay[d] <= maxHoursPerDay );
```

Όριο στον αριθμό των ημερών ανά εβδομάδα

Για να επιβάλλουμε αυτόν τον περιορισμό, θα δημιουργήσουμε πάλι έναν μονοδιάστατο πίνακα, τον `vOccupiedDay`, με τη "boolean" περιορισμένη μεταβλητή `vOccupiedDay[`d`]` να είναι αληθής αν κάποια ώρα της ημέρας d, ο καθηγητής παραδίδει κάποια διάλεξη.

```
for (d=0;  d<days;  ++d)
    vOccupiedDay.push_back( vHoursPerDay[d] > 0 );
pm.add( NsSum(vOccupiedDay) <= maxDaysPerWeek );
```

Όριο στον αριθμό των συνεχόμενων ωρών

Ακολουθώντας την ίδια φιλοσοφία, μπορούμε να βάλουμε ένα όριο (ίσο με `maxHoursContinuous`) στις συνεχόμενες ώρες διδασκαλίας για έναν καθηγητή. Ο «ενδιάμεσος» πίνακας περιορισμένος μεταβλητών που θα χρησιμοποιήσουμε, ονομάζεται `vContHoursUntil` και είναι δισδιάστατος. Με τη μεταβλητή `vContHoursUntil[`d`][`h`]` παριστάνουμε τον αριθμό των συνεχόμενων ωρών που διδάσκει ο καθηγητής, μέχρι την ώρα h (για την ημέρα d), της h συμπεριλαμβανομένης. Αν ο καθηγητής δεν κάνει μάθημα την ώρα h, τότε η μεταβλητή θα είναι 0. Για να επιβληθεί ο περιορισμός, δηλώνουμε ότι για κάθε ημέρα d, το $\max_h\{$`vContHoursUntil[`d`][`h`]`$\}$ πρέπει να φράσσεται από το όριο που έχει τεθεί.[5]

```
for (d=0;  d<days;  ++d)   {
    h = 0;
    vContHoursUntil[d][h] = ( vTimetable[d*hours + h] >= 0 );
    for (h=1;  h<hours;  ++h)
```

[5] Στον πραγματικό κώδικα για τον επιλυτή, αντί για `vContHoursUntil[d][h] = ...`, γράφουμε `vContHoursUntil[d].push_back( ... )`. Εδώ, χρησιμοποιήσαμε την πρώτη εκδοχή για λόγους αναγνωσιμότητας.

```
        vContHoursUntil[d][h] = (vTimetable[d*hours + h] >= 0)
                                * (vContHoursUntil[d][h-1] + 1);
    pm.add( NsMax(vContHoursUntil[d]) <= maxHoursContinuous );
}
```

4.6.2 Περιορισμοί αντικειμενικής συνάρτησης

Η αντικειμενική συνάρτηση (objective function), είναι ουσιαστικά μία περιορισμένη μεταβλητή. Μπορούμε να ενημερώσουμε τον επιλυτή ότι επιθυμούμε την ελαχιστοποίηση αυτής της μεταβλητής. Αν φτιάξουμε τη μεταβλητή έτσι ώστε να αντικατοπτρίζει τα μειονεκτήματα μιας λύσης, όσο περισσότερο την ελαχιστοποιούμε, τόσο η ποιότητα της λύσης θα αυξάνει (βλ. §4.3.3).

Στο πρόβλημα που εξετάζουμε, για να πάρουμε την «αντικειμενική μεταβλητή», απλά αθροίζουμε τις mini-αντικειμενικές-μεταβλητές των στιγμιοτύπων της κλάσης `CObjective`. Για να κατανοήσουμε την προηγούμενη πρόταση, θα δούμε ένα παράδειγμα. Καταρχήν, ανάλογα με το αν καταρτίζουμε ωρολόγιο πρόγραμμα για πανεπιστήμιο ή για σχολείο, ένα στιγμιότυπο της `CObjective` μπορεί να αναφέρεται αντίστοιχα σε μία ομάδα μαθημάτων ή σε έναν καθηγητή (Σχήμα 4.3). Και οι δύο περιπτώσεις αντιμετωπίζονται κατά τον ίδιο τρόπο — για αυτό εξάλλου παράγονται από την ίδια κλάση—, με τη διαφορά ότι για τις ομάδες μαθημάτων, οι mini-αντικειμενικές-μεταβλητές μπαίνουν με διαφορετικά βάρη —οριζόμενα από τον χρήστη— στο άθροισμα από το οποίο προκύπτει η τελική αντικειμενική μεταβλητή.

Ας πάρουμε λοιπόν μία ομάδα μαθημάτων. Ένα μειονέκτημα σε αυτήν, είναι η ανισοκατανομή των μαθημάτων μέσα στην εβδομάδα. Το μέτρο για αυτήν την ανισοκατανομή, δίδεται από την παρακάτω μεταβλητή (που προκύπτει από την παράσταση (4.15)· το `greedy` είναι ένας ακέραιος αριθμός που συνδέεται άμεσα με αυτήν την παράσταση).

```
vDistribution =
        NsMax(vHoursPerDay) - NsMin(vHoursPerDay) - greedy;
```

Μένει να δούμε το δεύτερο και τελευταίο μειονέκτημα, που δεν είναι άλλο από τον αριθμό των κενών ωρών μεταξύ των διαλέξεων μιας ομάδας, τον οποίο θα παραστήσουμε με την περιορισμένη μεταβλητή `vHoles`. Για να την κατασκευάσουμε, θα χρησιμοποιήσουμε τέσσερις βοηθητικούς πίνακες: τους δισδιάστατους `vBefore` και `vAfter` και τους μονοδιάστατους `vHolesArr` και `vTimetable`, ο οποίος ορίστηκε στην προηγούμενη παράγραφο.

```
for (d=0;  d<days;  ++d)   {
    vBefore[d][0]      = NsIntVar(pm,0,0);
    vAfter[d][hours-1] = NsIntVar(pm,0,0);
    for (h=1;  h<hours;  ++h)
        vBefore[d][h] = vBefore[d][h-1] == 1
                          ||  vTimetable[d*hours + h-1] >= 0;
    for (h=hours-2;  h>=0;  --h)
        vAfter[d][h]  = vAfter[d][h+1] == 1
                          ||  vTimetable[d*hours + h+1] >= 0;
    for (h=0;  h<hours;  ++h)
        vHolesArr.push_back(vBefore[d][h]==1 && vAfter[d][h]==1
                            &&  vTimetable[d*hours + h] < 0 );
}
vHoles = NsSum(vHolesArr);
```

Μία "boolean" περιορισμένη μεταβλητή `vBefore[`d`][`h`]` είναι 1, αν για την ημέρα d, υπάρχει διάλεξη μαθήματος της ομάδας, που γίνεται σε μία ώρα $h' \leq h$. Ανάλογα, ορίζεται και η `vAfter[`d`][`h`]`, η οποία είναι 1 αν υπάρχει μάθημα της ομάδας, την ημέρα d, που γίνεται σε μία ώρα $h' \geq h$.

Η "boolean" μεταβλητή `vHolesArr[`$d \cdot H + h$`]` είναι 1, αν για την ημέρα d, την ώρα h δεν γίνεται κάποιο μάθημα της ομάδας και για την ίδια ημέρα, πριν και μετά την h, λαμβάνουν χώρα κάποιες διαλέξεις τις ομάδας. Σε αυτήν την περίπτωση η h λογίζεται ως κενή και θα πρέπει να προστεθεί στο άθροισμα από το οποίο παίρνουμε τη `vHoles`.

Οι `vDistribution` και `vHoles` αθροίζονται, έχοντας συγκεκριμένα βάρη και έτσι προκύπτει η mini-αντικειμενική-μεταβλητή για τη `CObjective`.

4.6.3 Αναζήτηση

Ένας ενδεδειγμένος τρόπος αναζήτησης για τέτοια προβλήματα, είναι η *αναζήτηση περιορισμένων ασυμφωνιών* (limited discrepancy search – LDS) [34], σε συνδυασμό με την *τοπική αναζήτηση* (local search).

Ακόμα, για να μειωθεί ο χώρος αναζήτησης, η όποια οπισθοδρόμηση γίνεται, θα αφορά μόνο τις διαλέξεις (`var[i]`) και όχι τις αίθουσες (`classvar[i]`). Δηλαδή, όταν αναθέτουμε μία συγκεκριμένη ώρα σε μία `var[i]`, επιλέγουμε για αυτήν την καλύτερη —βάσει των προτιμήσεων που έχουν διατυπωθεί για το συγκεκριμένο μάθημα— διαθέσιμη αίθουσα. Αν χρειαστεί να οπισθοδρομήσουμε, *δεν* θα επιλέξουμε άλλη αίθουσα για τη διάλεξη. Απλώς, θα ακυρώσουμε

τη δέσμευση της αίθουσας και θα αναθέσουμε π.χ. μία άλλη ώρα στη `var[i]`. Με άλλα λόγια, δεν εξετάζουμε εναλλακτικές επιλογές για την αίθουσα μιας διάλεξης, αλλά μόνο για την ώρα που θα γίνεται η διάλεξη.

4.6.4 Ευριστικά

Για να κατευθυνθεί γρηγορότερα η αναζήτηση σε μία λύση, αντί να κάνουμε τυχαίες επιλογές (όπως π.χ. να αναθέτουμε τυχαίες τιμές στις μεταβλητές και έπειτα να ελέγχουμε αν από αυτές προκύπτει λύση), συμβουλευόμαστε τις ευριστικές συναρτήσεις

Επιλογή μεταβλητής

Η πρώτη ευριστική συνάρτηση, μας υποδεικνύει ποια μη δεσμευμένη μεταβλητή (ήτοι διάλεξη) θα επιλέξουμε για να της ανατεθεί τιμή (variable ordering heuristic).

1. Επιλέγουμε από το σύνολο των μεταβλητών-διαλέξεων, τη μεταβλητή εκείνη για την οποία παίρνουμε το *ελάχιστο* πηλίκο **μέγεθος πεδίου μεταβλητής**[6] διά **διάρκεια διάλεξης**.

2. Αν υπάρχει ισοβαθμία για το παραπάνω κριτήριο, τότε προχωράμε στο επόμενο κριτήριο, κοινώς εξετάζουμε το tie-break. Το 2^o κριτήριο ευνοεί τη διάλεξη με το *μεγαλύτερο* άθροισμα των βαρών των ομάδων μαθημάτων στις οποίες συμμετέχει. Ασφαλώς, αυτό το κριτήριο έχει νόημα όταν αναφερόμαστε σε πανεπιστήμια και όχι σε σχολεία.

Το επόμενο βήμα μετά την επιλογή της μεταβλητής-διάλεξης, είναι να της αναθέσουμε τιμή.

Επιλογή τιμής μεταβλητής

Για να διαλέξουμε μία τιμή (value ordering heuristic) για να ανατεθεί σε μία περιορισμένη μεταβλητή που αντιπροσωπεύει μία διάλεξη ℓ, θα συνδυάσουμε με συγκεκριμένα βάρη τα παρακάτω δύο κριτήρια. Για την απλοποίηση των όποιων υπολογισμών θα κάνουμε, όταν αναφερόμαστε σε κάποια άλλη διάλεξη ($\ell' \neq \ell$), θα εννοείται ότι σε αυτήν έχει ήδη ανατεθεί τιμή· μη «δεσμευμένες» διαλέξεις θα αγνοούνται.

[6] Ευριστικό first-fail.

- Επιλογή της *ημέρας* εκείνης, από την οποία θα προκύψει η μεγαλύτερη βελτίωση στην ισοκατανομή των διαλέξεων, για τις `CObjective`s στις οποίες συμμετέχει η διάλεξη ℓ. Όπως έχει προαναφερθεί, ένα στιγμιότυπο `obj` της `CObjective`, μπορεί να είναι είτε μία ομάδα μαθημάτων, είτε ένας καθηγητής. Το βάρος της ομάδας μαθημάτων συμβολίζεται με $w_{\texttt{obj}}$· όταν αναφερόμαστε σε καθηγητή, θα ισχύει $w_{\texttt{obj}} = 1$. Επομένως, επιζητούμε την ανάθεση τιμής για την οποία θα *ελαχιστοποιηθεί* η παράσταση[7]

$$\sum_{\texttt{obj}} w_{\texttt{obj}}\ \Delta\texttt{obj->vDistribution}$$

  Φυσικά, αρκεί να εξετάσουμε μόνο τα `obj` με τα οποία συνδέεται η ℓ — και όχι τα υπόλοιπα. Αυτό το ευριστικό στοχεύει μόνο στην επιλογή ημέρας και όχι ώρας για την ℓ, καθώς η ανισοκατανομή είναι έννοια που αφορά τη διασπορά των διαλέξεων στις *ημέρες* της εβδομάδας.

- Επιλογή της *ημέρας και ώρας* για την οποία θα προκύψουν τα λιγότερα κενά μεταξύ των διαλέξεων των στιγμιότυπων `obj` της `CObjective`, στα οποία ανήκει η ℓ. Επιθυμούμε λοιπόν, να διαλέξουμε μία ανάθεση για την ℓ, η οποία να *μεγιστοποιεί* την παράσταση

$$\sum_{\texttt{obj}} w_{\texttt{obj}}\ (\text{αριθμός των διαλέξεων } \ell' \in \texttt{obj} \text{ που κολλάνε στην } \ell)$$

  Μια διάλεξη ℓ' «κολλάει» στην ℓ, αν ακριβώς μετά το τέλος μίας από τις δύο διαλέξεις, αρχίζει η έτερη διάλεξη.

Αν από τον συνδυασμό των δύο ευριστικών προκύψει ισοβαθμία, υπάρχει το κριτήριο (tie-break) για να επηρεάσει την πορεία της αναζήτησης: Επιλογή της *ημέρας* για την οποία θα έχουμε το μεγαλύτερο άπλωμα των διαλέξεων του ίδιου μαθήματος — εφόσον το ℓ είναι διάλεξη ενός μαθήματος με περισσότερες της μίας διαλέξεις. Προτιμάμε δηλαδή τις ημέρες που βρίσκονται όσο το δυνατόν *περισσότερες* ημέρες, πριν την πρώτη ή μετά την τελευταία διάλεξη του μαθήματος στο οποίο ανήκει η ℓ.

[7] Για τη γρήγορη υλοποίηση της παράστασης, είναι δυνατόν να χρησιμοποιηθούν οι μεταβλητές `vHoursPerDay[`d`]` της `CObjective`.

Κεφάλαιο 5

Συμπεράσματα και μελλοντική δουλειά

Αν δεις ότι καταναλώνεις όλο σου σχεδόν τον χρόνο στη θεωρία, ασχολήσου λίγο και με πρακτικά πράγματα· έτσι θα βελτιώσεις τις θεωρίες σου. Αν δεις ότι ξοδεύεις όλο σου σχεδόν τον χρόνο στην πράξη, δώσε λίγη προσοχή και στα θεωρητικά ζητήματα· έτσι θα βελτιώσεις τις πρακτικές σου.

— Donald Knuth

Τα θέματα με τα οποία ασχολείται αυτή η εργασία συνοψίζονται στα εξής δύο: α) σχεδιασμός και υλοποίηση μιας βιβλιοθήκης για προγραμματισμό με περιορισμούς και β) σχεδιασμός και υλοποίηση ενός συστήματος κατάρτισης ωρολογίων προγραμμάτων. Και στις δύο υλοποιήσεις, κύριο μέλημά μας ήταν να δώσουμε λύσεις γενικού και όχι ειδικού χαρακτήρα.

Επειδή οι εφαρμογές οι οποίες χτίστηκαν ήταν ογκώδεις, το αντικειμενοστραφές περιβάλλον προγραμματισμού στο οποίο αναπτύχθηκαν βοήθησε πολύ στην καλύτερη οργάνωση του κώδικα. Επίσης η «ζεύξη» της C++ με τις διεπαφές της Visual Basic .NET, μέσω ενός DLL, συντέλεσε στον καλύτερο συνδυασμό της απόδοσης με τη λειτουργικότητα.

Η επιλογή μας να εργαστούμε πάνω στον προγραμματισμό με περιορισμούς —και μια σημαντική εφαρμογή του, όπως ο χρονοπρογραμματισμός— αποτελεί κομβικό σημείο για την επεκτασιμότητα των εφαρμογών. Η δηλωτικότητα με την οποία εκφράζουμε τους περιορισμούς στα προγράμματα μάς επιτρέπει την άμεση προσθαφαίρεση περιορισμών και την εύκολη τροποποίηση των κριτηρίων ποιότητας των λύσεων που δίνει ο επιλυτής.

Όσον αφορά τον ίδιο τον επιλυτή, δημιουργήθηκε τούτη την περίοδο, κατά την οποία υπάρχει μεγάλη κινητικότητα στον τομέα έρευνας για τον προγραμματισμό με περιορισμούς. Προσπαθήσαμε να λάβουμε υπόψη τους πιο διαδεδομένους αλγορίθμους στη βιβλιογραφία, ωστόσο, μελλοντικά, θα πρέπει να την εξετάσουμε σε μεγαλύτερο βάθος και να είμαστε ενήμεροι των όποιων σημαντικών εξελίξεων που είναι σχεδόν βέβαιο ότι θα υπάρξουν.

Υπάρχουν μερικά πιο απτά ζητήματα τα οποία θα πρέπει να ακουμπήσουμε. Οι πιο διαδεδομένοι περιορισμοί, έχουν ήδη υλοποιηθεί στον επιλυτή, όμως υπάρχουν πολλοί ακόμα που μπορούν να υλοποιηθούν, όπως αυτός του υπολοίπου ($V_1 \bmod V_2$). Ακόμα, ακολουθώντας και το παράδειγμα κάποιων άλλων επιλυτών όπως αυτού της GNU Prolog [35], καλό θα ήταν να ξεκαθαριστούν τα όρια μεταξύ 2-συνέπειας και συνέπειας-ορίων· στην παρούσα φάση υπάρχει ένα υβρίδιο, δηλαδή σε άλλους περιορισμούς εφαρμόζεται συνέπεια-ορίων και σε άλλους όχι. Επίσης, θα πρέπει να επιτραπεί η αφαίρεση *εύρους* τιμών —και όχι απλά μίας τιμής κάθε φορά— από μία μεταβλητή. Αυτό θα επιτάχυνε την εφαρμογή συνέπειας-ορίων, καθώς ο αλγόριθμος θα ασχολούνταν μόνο με ένα άκρο από το αφαιρεμένο εύρος τιμών και όχι με κάθε τιμή του. Τέλος, το σημαντικότερο ίσως από όλα είναι να γραφεί ένα εύληπτο εγχειρίδιο χρήσης του επιλυτή, το οποίο θα έδενε στο μέγιστο βαθμό τη θεωρία με την πράξη, μέσω παραδειγμάτων. Είναι εξαιρετικά κρίσιμο να μπορέσουμε να μυήσουμε, ακόμα και σχετικά άπειρους προγραμματιστές, στον κόσμο του προγραμματισμού με περιορισμούς και να ξεδιπλώσουμε σε μικρό χρονικό διάστημα τις δυνατότητές του. Σε αυτό το πλαίσιο, θα ήταν ενδιαφέρον αν ένας προγραμματιστής εφαρμογών που χρησιμοποιεί τον επιλυτή προχωρούσε σε σχεδιασμό στόχων —για τον μηχανισμό στόχων του επιλυτή— που θα επιδείκνυαν ποικίλων ειδών μεθόδους αναζήτησης και ευριστικά.

Όσον αφορά το σύστημα κατάρτισης ωρολογίων προγραμμάτων, υπάρχει αρκετή δουλειά να γίνει, αναφορικά με τις μεθόδους αναζήτησης και τα ευριστικά που αυτό εκμεταλλεύεται. Ως τώρα, έγινε προσπάθεια να υλοποιηθούν μόνο εκείνα που η εμπειρία του Orprog έδειξε ότι «δουλεύουν καλά». Συνεπώς, δεν έχει δοθεί το κατάλληλο βάρος στη μελέτη άλλων μεθόδων αναζήτησης, στις παραμέτρους τους και, γιατί όχι, στη χρήση συνδυασμών τους. Ακόμα, θα ήταν μία τολμηρή ιδέα, να μην ακολουθούμε τυφλά ένα συγκεκριμένο τρόπο αναζήτησης, κάθε φορά που μας δίνεται ένα πρόβλημα, αλλά να επιλέγουμε σε χρόνο εκτέλεσης (run-time) τη μέθοδο αναζήτησης που θα ταίριαζε καλύτερα στο συγκεκριμένο πρόβλημα. Αυτή η «έξυπνη» επιλογή της μεθόδου αναζήτησης θα μπορούσε να αλλάζει τις αποφάσεις της ακόμα και όταν η αναζήτηση βρίσκεται σε εξέλιξη.

Τέλος, υπάρχουν περιθώρια βελτίωσης και για τη διεπαφή του συστήματος κατάρτισης ωρολογίων προγραμμάτων. Θα μπορούσε να γίνει πιο φιλική στον χρήστη, μέσω της δημιουργίας μαγικών-διαλόγων (wizards), οι οποίοι θα διευκόλυναν τη γρήγορη και απλή διατύπωση προβλημάτων από τον πλέον αρχάριο χρήστη, είτε αυτός φτιάχνει πρόγραμμα για πανεπιστήμιο, είτε για σχολείο. Επίσης μία λειτουργία που γίνεται να υλοποιηθεί, είναι η διεθνοποίηση της διεπαφής: Η διεπαφή κάθε φορά που ξεκινά θα μπορούσε να διαβάζει ένα αρχείο με τις μεταφράσεις όλων των διαλόγων, στη γλώσσα του τελικού χρήστη.

Παράρτημα Α΄

Εγχειρίδιο χρήσης του Naxos Solver

Ο σκοπός αυτού του παραρτήματος είναι να δώσει τις πληροφορίες που χρειάζεται ένας προγραμματιστής εφαρμογών (application developer) της βιβλιοθήκης. Σε ένα σχήμα «πρωθύστερο», έχουμε ήδη αποκαλύψει κάποιες τεχνικές πτυχές του Naxos Solver, κατά την ανάλυση των αλγορίθμων του· π.χ. περιγράφηκε αναλυτικά ο μηχανισμός στόχων (§3.6).

Ο Naxos Solver είναι μία βιβλιοθήκη επίλυσης προβλημάτων ικανοποίησης περιορισμών για το αντικειμενοστραφές περιβάλλον της γλώσσας C++. Ο επιλυτής είναι ασφαλής για χρήση σε πολυνηματικό περιβάλλον (threadsafe). Δεν θα πρέπει να χρησιμοποιούνται «εσωτερικές» κλάσεις, μέθοδοι, συναρτήσεις του κ.λπ. οι οποίες δεν περιγράφονται στο εγχειρίδιο χρήσης, καθώς μπορεί να αλλάξουν στο μέλλον. Ακόμα, για αποφυγή παρεξηγήσεων, καλό είναι να μην ονομάζουμε τις δικές μας μεταβλητές, κλάσεις κ.λπ. με ονόματα που ξεκινούν από "`Ns`", καθώς αυτό είναι το πρόθεμα για τις ενσωματωμένες κλάσεις και σταθερές του επιλυτή. Τέλος, σημειώνεται ότι ο επιλυτής δεν κάνει κανένα έλεγχο για τυχόν υπερχειλίσεις (π.χ. κατά την πρόσθεση δύο μεγάλων ακεραίων), για λόγους απόδοσης.

Μέρος του σχεδιασμού και της «ονοματολογίας» του επιλυτή είναι επηρεασμένο από τη μοντελοποίηση της Standard Template Library (STL) για τη C++ [36]. Π.χ. χρησιμοποιήθηκαν και υλοποιήθηκαν αρκετοί επαναλήπτες (iterators).

Δεν έχουμε διάκριση των κλάσεων του Naxos Solver σε κλάσεις-χειριστήρια (handle-classes) και κλάσεις-υλοποίησης (implementation-classes), όπως γίνεται στον Ilog Solver. Ο λόγος αυτής της διαφοροποίησης στον

Ilog Solver, έχει να κάνει με την προσπάθεια να διαχειριστεί αυτόματα τη μνήμη αλά Java. Σε κάθε κλάση-χειριστήριο, υπάρχει μία αναφορά σε μία κλάση-υλοποίησης. Είναι δυνατόν πολλές κλάσεις-χειριστήρια να δείχνουν στην ίδια κλάση-υλοποίησης. Η κλάση-υλοποίησης διαγράφεται από τη μνήμη, μόνο όταν πάψει να υφίσταται και η τελευταία κλάση-χειριστήριο που δείχνει σε αυτήν. (Κάτι αντίστοιχο γίνεται και στην Java με τις αναφορές.) Επομένως, στον Ilog Solver είναι δυνατόν π.χ. σε μία συνάρτηση να δημιουργούμε αυτόματες μεταβλητές για να περιγράψουμε έναν περιορισμό και αυτός, καθώς και οι μεταβλητές που αφορά, να υπάρχουν και μετά το τέλος της συνάρτησης· σε αντίστοιχη περίπτωση στον Naxos Solver θα έχουμε σφάλμα κατάτμησης (segmentation fault).

Α΄.1 Διαχείριση σφαλμάτων

Ξεκινάμε από τη διαχείριση σφαλμάτων, για την οποία είναι σημαντικό να φροντίζουμε, όποιο πρόγραμμα και να γράφουμε. Στον Naxos Solver πρέπει να συλλαμβάνονται οι εξαιρέσεις τύπου `NsException`. Η κλάση αυτή είναι υποκλάση της `runtime_error`, η οποία με τη σειρά της παράγεται από την `exception`. Αρκεί λοιπόν να συλλαμβάνουμε τον τύπο `exception`· με τη μέθοδο `what()` αυτής της βασικής κλάσης, θα πάρουμε σε μία συμβολοσειρά ένα μήνυμα για το σφάλμα που συνέβη.

```
#include <naxos.h>
using namespace naxos;
using namespace std;

int  main (void)
{
    try  {

        // ... CODE OF THE PROGRAM ... //

    } catch (exception& exc)  {
        cerr << exc.what() << "\n";

    } catch (...)  {
        cerr << "Unknown exception" << "\n";
```

```
    }
}
```

Δεν είναι σωστή προγραμματιστική τακτική να εντάσσουμε τις εξαιρέσεις στο σώμα των αλγορίθμων μας. Συνήθως, οι εξαιρέσεις αποτελούν το «περιτύλιγμα» των προγραμμάτων μας.

Α′.2 Περιορισμένες μεταβλητές

Ο επιλυτής υποστηρίζει ακέραιες περιορισμένες μεταβλητές πεπερασμένων πεδίων. Η κλάση με τις οποίες αναπαρίστανται είναι η `NsIntVar`, η οποία περιέχει τις παρακάτω μεθόδους.

`NsIntVar(NsProblemManager& pm, NsInt min, NsInt max)` Είναι μία μέθοδος-κατασκευής (constructor). Το `pm` είναι ο διαχειριστής προβλήματος στον οποίον ανήκει η μεταβλητή (βλ. §Α′.4) και τα `min` και `max`, το ελάχιστο και το μέγιστο του πεδίου τιμών της, που συμβολίζεται και με `[min..max]`.

Ο τύπος `NsInt` μπορεί να θεωρηθεί ότι είναι σε θέση να αναπαραστήσει τουλάχιστον τους ακέραιους αριθμούς εκείνους που είναι δυνατόν να αναπαρασταθούν με έναν `long`. Η ελάχιστη τιμή του τύπου `NsInt`, ισούται με τη σταθερά `NsMINUS_INF` και η μέγιστη με `NsPLUS_INF`. (Για τον μη προσημασμένο τύπο `NsUInt`, η μέγιστη τιμή είναι `NsUPLUS_INF`.)

Η ελάχιστη τιμή ενός πεδίου πρέπει να είναι γνήσια μεγαλύτερη από `NsMINUS_INF`, όπως και η μέγιστη τιμή πρέπει να είναι γνήσια μικρότερη από `NsPLUS_INF`, καθώς αυτές οι σταθερές αποτελούν ειδικές τιμές που αντιπροσωπεύουν το άπειρο, όπως θα δούμε και παρακάτω.

`NsIntVar()` Στο στιγμιότυπο της κλάσης που δημιουργείται με αυτήν τη μέθοδο-κατασκευής, είναι δυνατόν στη συνέχεια να ανατεθεί κάποια έκφραση (μέσω και του υπερφορτωμένου τελεστή "="), όπως στην τρίτη γραμμή του παρακάτω παραδείγματος.

```
NsIntVar  X(pm,0,3), Y(pm,-1,15), Z;
NsIntVar  W = X + 3*Y;
Z = W * W;
```

Στη δεύτερη γραμμή του παραδείγματος, χρησιμοποιήθηκε μία άλλη μέθοδος-κατασκευής, που παίρνει σαν όρισμα μία *Expression* (εδώ, η έκφραση ήταν η "`X + 3*Y`").

`bool remove(NsInt val)` Αφαιρεί την τιμή `val` από το πεδίο της μεταβλητής.

`bool remove(NsInt min, NsInt max)` Αφαιρεί από το πεδίο της μεταβλητής τις τιμές που βρίσκονται στο διάστημα [`min`, `max`]. Αντί να τις αφαιρούμε μία-μία με τη `remove(val)`, είναι πιο αποδοτικό να καλούμε αυτή τη μέθοδο.

`void removeAll()` «Αδειάζει» το πεδίο τιμών της μεταβλητής. Πρακτικά, αυτό που γίνεται είναι η ενημέρωση του διαχειριστή προβλήματος, ότι υπήρξε ασυνέπεια — λόγω του κενού πεδίου τιμών. Αυτή η μέθοδος είναι χρήσιμη όταν θέλουμε να προκαλέσουμε αποτυχία στην αναζήτηση. Π.χ. όταν θέλουμε να αποτύχει ένας στόχος, κατά την εκτέλεσή του, καλούμε αυτή τη μέθοδο για μία οποιαδήποτε μεταβλητή.[1]

`void set(NsInt val)` Αναθέτει την τιμή `val` στη μεταβλητή· συνεπώς, η μεταβλητή γίνεται δεσμευμένη.

Ακολουθούν οι μέθοδοι που δεν προκαλούν καμία αλλαγή στη μεταβλητή για την οποία καλούνται.

`bool contains(NsInt val)` Επιστρέφει `true` αν η τιμή `val` ανήκει στο πεδίο της μεταβλητής. Η κλήση `Var.contains(i)` είναι ισοδύναμη με την `Var[i]`, που προκύπτει από τον υπερφορτωμένο τελεστή "`[]`".

`NsInt min()` Η ελάχιστη τιμή του πεδίου τιμών.

`NsInt max()` Η μέγιστη τιμή του πεδίου τιμών.

`NsUInt size()` Αριθμός των τιμών που περιέχονται στο πεδίο τιμών.

`bool isBound()` Επιστρέφει `true` αν η μεταβλητή είναι δεσμευμένη.

`NsInt value()` Χρησιμοποιείται όταν η μεταβλητή είναι δεσμευμένη και επιστρέφει τη (μοναδική) τιμή της. Αν κληθεί για μία μη δεσμευμένη μεταβλητή, προκαλείται σφάλμα.

[1] Ενώ για να δείξουμε ότι ένας στόχος πετυχαίνει, κάνουμε την `GOAL()` να επιστρέψει `0`, για να δείξουμε ότι απέτυχε, υπάρχει αυτός ο λιγότερο κομψός τρόπος.

Αντί αυτής, θα μπορούσε να χρησιμοποιηθεί μία μέθοδος, από τις `min()` και `max()`. Ο λόγος ύπαρξης αυτής της μεθόδου, αφορά την αναγνωσιμότητα του κώδικα.

`NsInt next(NsInt val)` Επιστρέφει τη μικρότερη τιμή στο πεδίο τιμών, που είναι γνήσια μεγαλύτερη από την `val`. Αν δεν υπάρχει κάποια τέτοια τιμή, επιστρέφεται `NsPLUS_INF`. Η `val` μπορεί να πάρει και τις ειδικές τιμές `NsMINUS_INF` και `NsPLUS_INF`.

`NsInt previous(NsInt val)` Ορίζεται ανάλογα με τη `next()`. Επιστρέφει τη μεγαλύτερη τιμή στο πεδίο τιμών, που είναι γνήσια μικρότερη από την `val`. Αν δεν υπάρχει κάποια τέτοια τιμή, επιστρέφεται `NsMINUS_INF`. Η `val` μπορεί να πάρει και τις ειδικές τιμές `NsMINUS_INF` και `NsPLUS_INF`.

`int varsConnected()` Ο αριθμός άλλων μεταβλητών που συνδέονται μέσω περιορισμών με τη μεταβλητή. Χρήσιμη μέθοδος για το ευριστικό στο οποίο, κατά την αναζήτηση, επιλέγουμε να ανατεθεί τιμή πρώτα στη μεταβλητή που συμμετέχει στους περισσότερους περιορισμούς.

Στη συνέχεια παρουσιάζονται δύο επαναλήπτες για την κλάση και παραδείγματα χρήσης τους.

`NsIntVar::const_iterator` Διατρέχει τις τιμές του πεδίου της μεταβλητής. Π.χ. με τον παρακάτω κώδικα, τυπώνονται οι τιμές της μεταβλητής, σε αύξουσα σειρά.

```
for (NsIntVar::const_iterator v = Var.begin();
                              v != Var.end();   ++v)
        cout << *v << "\n";
```

`NsIntVar::const_gap_iterator` Διατρέχει τα κενά που υπάρχουν εντός του πεδίου της μεταβλητής. Αν χρησιμοποιούσαμε τη γλώσσα των μαθηματικών, θα λέγαμε ότι παίρνουμε όλες τις τιμές του συμπληρώματος του πεδίου τιμών της μεταβλητής, ως προς το [`min`, `max`] (όπου `min` το ελάχιστο και `max` το μέγιστο του πεδίου τιμών της μεταβλητής). Π.χ.

```
for (NsIntVar::const_gap_iterator g = Var.gap_begin();
                                  g != Var.gap_end();   ++g)
        cout << *g << "\n";
```

Τέλος, σημειώνεται ότι ο τελεστής "<<" έχει υπερφορτωθεί, για να είναι δυνατόν να τυπώνεται μία μεταβλητή σε ένα ρεύμα εξόδου, γράφοντας π.χ. "`cout << Var;`".

Α΄.3 Πίνακες περιορισμένων μεταβλητών

Ο προκαθορισμένος τύπος (ευέλικτου) πίνακα του επιλυτή, με στοιχεία περιορισμένες μεταβλητές, είναι ο `NsIntVarArray`. Μπορούμε να αναφερόμαστε στο i-οστό στοιχείο ενός τέτοιου πίνακα `VarArr`, μέσω της γνωστής παράστασης `VarArr[`i`]`. Ο προκαθορισμένος τύπος για το i, είναι ο `NsIndex`.

Μετά τη δημιουργία ενός `NsIntVarArray`, αυτός δεν περιέχει κανένα στοιχείο. Οι μεταβλητές που τον συνθέτουν, εισάγονται είτε στην αρχή, είτε στο τέλος του, όπως μπορεί να γίνει σε μία συνδεδεμένη λίστα. Η μέθοδος-κατασκευής του δεν παίρνει ορίσματα. Ακολουθούν οι υπόλοιπες μέθοδοι αυτής της κλάσης.

`NsIndex size()` Επιστρέφει το μέγεθος του πίνακα.

`bool empty()` Επιστρέφει `true` αν ο πίνακας είναι άδειος.

`NsIntVar& back()` Η τελευταία περιορισμένη μεταβλητή του πίνακα.

`NsIntVar& front()` Η πρώτη περιορισμένη μεταβλητή του πίνακα.

`void push_back(`**Expression**`)` Εισαγωγή στο τέλος του πίνακα, της μεταβλητής που θα ισούται με την έκφραση *Expression*. Όπως θα αναλυθεί σε επόμενη παράγραφο, μία *Expression* μπορεί απλά να είναι μία περιορισμένη μεταβλητή, ή κάποιος συνδυασμός μεταβλητών. Π.χ.

```
VarArr.push_back( NsIntVar(pm, 10, 30) );
VarArr.push_back( 3*VarX + VarY );
VarArr.push_back( VarX > 11 );
```

Στην πρώτη εντολή παραπάνω, ουσιαστικά εισάγαμε στο τέλος του πίνακα `VarArr` μία καινούργια μεταβλητή με πεδίο `[10..30]`.

Με την τελευταία εντολή, εισάγουμε μία περιορισμένη μεταβλητή στο τέλος του πίνακα, που θα είναι 1 αν ισχύει `VarX > 11`, αλλιώς 0. (Μπορεί φυσικά το πεδίο της να είναι και το `[0..1]`, σε περίπτωση που κάποιες τιμές της `VarX` είναι μικρότερες και κάποιες μεγαλύτερες του 11.) Έχουμε δηλαδή έναν μετα-περιορισμό.

`void push_front(`**Expression**`)` Όπως η `push_back()`, αλλά η εισαγωγή γίνεται στην αρχή του πίνακα.

Ακολουθούν οι επαναλήπτες για τους πίνακες. Χρησιμοποιώντας τους, μπορούμε να διατρέχουμε εύκολα όλες τις μεταβλητές του πίνακα.

`NsIntVarArray::iterator` Επαναλήπτης για να παίρνουμε κατά σειρά τις μεταβλητές του πίνακα. Π.χ. παρακάτω αφαιρούμε την τιμή 2 από όλες τις μεταβλητές του `VarArr`:

```
for (NsIntVarArray::iterator  V = VarArr.begin();
                              V != VarArr.end();   ++V)
        V->remove(2);
```

`NsIntVarArray::const_iterator` Αυτός ο επαναλήπτης υπάρχει για την περίπτωση που δεν θέλουμε να τροποποιήσουμε τις μεταβλητές. Μπορούμε π.χ. χρησιμοποιώντας τον, να τυπώσουμε τις μεταβλητές ενός πίνακα.

```
for (NsIntVarArray::const_iterator  V = VarArr.begin();
                                    V != VarArr.end();   ++V)
        cout << *V << "\n";
```

Τέλος, σημειώνεται ότι ο τελεστής "<<" έχει υπερφορτωθεί και για τους πίνακες, οι οποίοι μπορούν να τυπωθούν, γράφοντας π.χ. "`cout << VarArr;`".

Α΄.4 Διαχειριστής προβλήματος

Πριν ξεκινήσουμε να διατυπώνουμε ένα πρόβλημα, πρέπει να ορίσουμε ένα *διαχειριστή προβλήματος* (κλάση `NsProblemManager`). Ο διαχειριστής αυτός συγκρατεί όσες πληροφορίες χρειάζονται για τις μεταβλητές, το δίκτυο περιορισμών που κατασκευάζεται και τους στόχους που θα εκτελεστούν. Η μέθοδος-κατασκευής του δεν παίρνει ορίσματα. Ακολουθούν οι υπόλοιπες μέθοδοι.

`void add(`**ExprConstr**`)` Προσθέτει τον περιορισμό που περιγράφει η έκφραση περιορισμού *ExprConstr* (βλ. §Α΄.5.1). Σε μία έκφραση περιορισμού χρησιμοποιούνται η τελεστές των συνθηκών (`<`, `==`, `!=` κ.λπ.), ή ενσωματωμένες εκφράσεις όπως η `NsAllDiff()` που επιβάλλει όλα τα στοιχεία ενός πίνακα να είναι διαφορετικά. Π.χ.

```
pm.add( 3*VarX != VarY/2 );
pm.add( VarW == -2 || VarW >= 5 );
pm.add( NsAllDiff(VarArr) );
```

`void addGoal(NsGoal* goal)` Προσθήκη του `goal` στη λίστα με τους προς ικανοποίηση στόχους (βλ. §3.6).

`bool nextSolution()` Εύρεση της επόμενης λύσης. Ικανοποίηση των στόχων που έχουν τεθεί. Αν δεν βρεθεί λύση, επιστρέφεται `false`.

`void minimize(`**Expression**`)` Δίνει οδηγία στον επιλυτή να προσπαθήσει να ελαχιστοποιήσει την τιμή της *Expression*. Κάθε φορά που ο επιλυτής βρίσκει μία λύση, θα πρέπει η τιμή της *Expression* να είναι μικρότερη από αυτήν της προηγούμενης λύσης (αλγόριθμος branch-and-bound). Ουσιαστικά, μετά από κάθε λύση που δίνει η `nextSolution()`, αν η μέγιστη τιμή της *Expression* είναι a, επιβάλλεται ο περιορισμός *Expression* $< a$, για την επόμενη φορά που θα κληθεί η `nextSolution()`. Δηλαδή σε κάθε λύση, η *Expression* βγαίνει μειωμένη. Αν δεν μπορεί να μειωθεί περαιτέρω, η `nextSolution()` θα επιστρέψει `false` και θα πρέπει να έχουμε αποθηκευμένη κάπου την τελευταία (βέλτιστη) λύση. Π.χ.

```
pm.minimize( VarX + VarY );
while (pm.nextSolution() != false)
    {  /* STORE SOLUTION */  }
```

Αν επιθυμούμε να μεγιστοποιήσουμε μία *Expression*, απλά βάζουμε ένα "−" μπροστά της και καλούμε τη `minimize()` για αυτήν.

`void timeLimit(unsigned long secs)` Η αναζήτηση θα διαρκέσει το πολύ `secs` δευτερόλεπτα. Μετά το πέρας αυτού του χρονικού διαστήματος, η `nextSolution()` επιστρέφει `false`.

`void realTimeLimit(unsigned long secs)` Κάνει ό,τι και η προηγούμενη μέθοδος, με τη διαφορά ότι εδώ τα `secs` δευτερόλεπτα είναι πραγματικός χρόνος, ενώ στην προηγούμενη μέθοδο ήταν ο καθαρός χρόνος που έχει δοθεί από το σύστημα, αποκλειστικά στον επιλυτή (CPU time).

Α΄.5 Εκφράσεις

Για να συνδέσουμε τις μεταβλητές, εκμεταλλευόμαστε τους υπερφορτωμένους τελεστές και δημιουργούμε εκφράσεις και συνδυασμούς από εκφράσεις. Μία απλή έκφραση μπορεί να είναι και μία μεταβλητή ή ένας ακέραιος. Μία έκφραση συμβολίζεται με *Expression*.

Α΄.5.1 Εκφράσεις περιορισμών

Συμβολίζονται με *ExprConstr* και είναι υποκατηγορία των εκφράσεων *Expression*. Χρησιμοποιούνται κυρίως σαν ορίσματα της `NsProblemManager::add()` και για τη δημιουργία μετα-περιορισμών. Οι παρακάτω παραστάσεις είναι *ExprConstr*:

- *Expression*$_1$ op *Expression*$_2$, op $\in \{$`<`, `<=`, `>`, `>=`, `==`, `!=`$\}$
- `!(` *ExprConstr* `)`
- *ExprConstr*$_1$ op *ExprConstr*$_2$, op $\in \{$`&&`, `||`$\}$
- `NsAllDiff(` $VarArr$ `)`

Ο ορισμός είναι λοιπόν αναδρομικός. Η τελευταία έκφραση συμβολίζει ότι οι περιορισμένες μεταβλητές του $VarArr$ (που είναι πίνακας του τύπου `NsIntVarArray`) είναι διαφορετικές μεταξύ τους. Παρακάτω παρουσιάζονται μερικές εκφράσεις περιορισμών:

```
VarX < VarY
!( X==Y  ||  X+Y==-3 )
(X==Y) != 1
```

Α΄.5.2 Γενικές εκφράσεις

Εκτός από τις *ExprConstr*, στην κατηγορία των γενικών εκφράσεων *Expression*, ανήκουν και οι εξής παραστάσεις:

- *Expression*$_1$ op *Expression*$_2$, op $\in \{$`+`, `-`, `*`, `/`$\}$
- `NsMin(` $VarArr$ `)`
- `NsMax(` $VarArr$ `)`

- `NsSum(` $VarArr$ `)`
- `NsSum(` $VarArr, start, length$ `)`

Μία *Expression* μπορεί —εκτός από το να συμμετέχει σε μία έκφραση περιορισμού— να ανατεθεί σε μία μεταβλητή. Π.χ. μπορούμε να γράψουμε

```
NsIntVar  X = Y + 3/Z;
NsIntVar  W = NsSum(VarArrA) - (NsMin(VarArrB) == 10);
```

Σημειώνεται μάλιστα, ότι από το να γράφουμε `VarArr[0] + VarArr[1] + VarArr[2]`, είναι πιο αποδοτικό να χρησιμοποιούμε την ισοδύναμη έκφραση `NsSum(VarArr,0,3)`. Το δεύτερο και το τρίτο όρισμα του `NsSum` είναι προαιρετικά και συμβολίζουν αντίστοιχα τη θέση του πρώτου στοιχείου του `VarArr` το οποίο θα μπει στο άθροισμα και τον αριθμό των μετέπειτα στοιχείων (μαζί με το πρώτο) που θα συνυπολογιστούν στο άθροισμα. Αν δεν υπάρχουν αυτά τα δύο ορίσματα, τότε όλος ο πίνακας μπαίνει στο άθροισμα.

Ο λόγος που το `NsSum` είναι η αποδοτικότερη από τις δύο εκφράσεις, έχει να κάνει με το γεγονός ότι για την πρώτη παράσταση *θα δημιουργηθεί μία ενδιάμεση μεταβλητή* που θα ισούται με `VarArr[0] + VarArr[1]` και σε αυτήν θα προστεθεί η `VarArr[2]`. Αυτό δεν θα γίνει για το `NsSum`.

Τα `NsMin` και `NsMax`, δίνουν το ελάχιστο και το μέγιστο αντίστοιχα του πίνακα τον οποίο δέχονται σαν όρισμα.

Α′.5.3 Εκφράσεις πινάκων

Τέλος, υπάρχει μια ειδική, ανεξάρτητη κατηγορία εκφράσεων, που αφορά ανάθεση σε πίνακα μεταβλητών τύπου `NsIntVarArray`. Περιλαμβάνει τις παρακάτω παραστάσεις, για τον περιορισμό Inverse (βλ. §3.4.14).

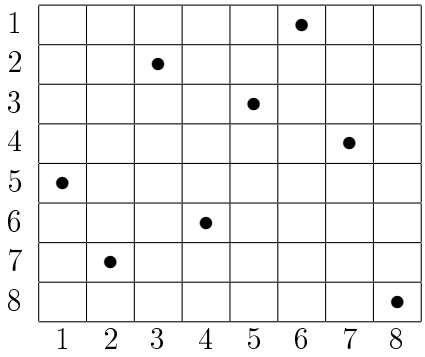
- `NsInverse(` $VarArr$ `)`
- `NsInverse(` $VarArr, maxdom$ `)`

Με $maxdom$ συμβολίζουμε τον αριθμό των στοιχείων του αντίστροφου πίνακα που θέλουμε να φτιάξουμε. Αν δεν υπάρχει τιμή για αυτό το όρισμα, θεωρούμε ότι $maxdom = \max_{V \in VarArr}\{V.\max\}$. Σε κάθε περίπτωση, το $maxdom$ πρέπει να είναι μεγαλύτερο ή ίσο από αυτήν την τιμή. Π.χ.

```
NsIntVarArray  VarArrB = NsInverse(VarArrA);
NsIntVarArray  VarArrC;
VarArrC = NsInverse(VarArrA, 100);
```

Α΄.6 Το πρόβλημα των N βασιλισσών

Σαν παράδειγμα θα διατυπώσουμε ένα πραγματικό πρόβλημα.



Σχήμα Α΄.1: 8 βασίλισσες που δεν απειλούνται [37]

Α΄.6.1 Ορισμός

Το πρόβλημα των N βασιλισσών συνίσταται στην τοποθέτηση N βασιλισσών σε μία $N \times N$ σκακιέρα, με τέτοιο τρόπο ώστε καμία να μην απειλείται. Με άλλα λόγια θέλουμε να τοποθετήσουμε N στοιχεία σε ένα πλέγμα $N \times N$, έτσι ώστε κανένα από αυτά να μην έχει κοινή γραμμή, στήλη, ή διαγώνιο με κάποιο άλλο. Στο Σχήμα Α΄.1 φαίνεται ένα παράδειγμα για $N = 8$.

Συνεπώς σε κάθε στήλη $0, 1, \ldots, N-1$ θα αντιστοιχεί και μία βασίλισσα. Μένει να δούμε σε ποια γραμμή θα τοποθετήσουμε την καθεμία. Οπότε οι άγνωστοι του προβλήματος είναι οι μεταβλητές X_i με $0 \leq X_i \leq N-1$, όπου X_i είναι η γραμμή στην οποία βρίσκεται η βασίλισσα της στήλης i.

Όσον αφορά τους περιορισμούς που ισχύουν, καταρχήν οι βασίλισσες πρέπει να μην απειλούνται στις γραμμές, δηλαδή

$$X_i \neq X_j, \quad \forall i \neq j \tag{Α΄.1}$$

Επίσης δεν πρέπει να απειλούνται στις δύο διαγώνιους, οπότε

$$X_i + i \neq X_j + j \quad \text{και} \quad X_i - i \neq X_j - j, \quad \forall i \neq j \tag{Α΄.2}$$

Το $X_i + i$ αντιστοιχεί στην πρώτη και το $X_i - i$ στη δεύτερη διαγώνιο της βασίλισσας της στήλης i.

Α′.6.2 Κώδικας

Στον κώδικα για τον επιλυτή, οι μεταβλητές X_i αναπαρίστανται με έναν πίνακα `Var` για τον οποίο θα απαιτήσουμε, σύμφωνα με την (Α′.1), να έχει διαφορετικά μεταξύ τους στοιχεία. Όσον αφορά την (Α′.2), φτιάχνουμε δύο ακόμα πίνακες, τον `VarPlus` και τον `VarMinus`, με στοιχεία τα $X_i + i$ και $X_i - i$ αντίστοιχα. Και για αυτούς τους πίνακες, θα δηλώσουμε ότι ισχύει ο περιορισμός ότι τα στοιχεία που περιέχουν διαφέρουν μεταξύ τους.

```
int  N = 8;
NsProblemManager  pm;

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) );

pm.addGoal( new NsgLabeling(Var) );
while (pm.nextSolution() != false)
    cout << "Solution: " << Var << "\n";
```

Α′.7 $SEND + MORE = MONEY$

Ένα άλλο παράδειγμα, αφορά ένα γνωστό πρόβλημα *κρυπτάριθμου* (cryptarithm). Σε αυτά τα προβλήματα, έχουμε κάποιες αριθμητικές σχέσεις μεταξύ λέξεων, όπως η $SEND + MORE = MONEY$. Κάθε γράμμα των λέξεων αναπαριστά ένα συγκεκριμένο ψηφίο (0 ως 9) και έτσι κάθε λέξη αντιπροσωπεύει έναν δεκαδικό αριθμό. Δύο διαφορετικά γράμματα δεν πρέπει να παριστάνουν το ίδιο ψηφίο. Π.χ. στη σχέση $SEND + MORE = MONEY$, όπου υπάρχει το E, θα βάλουμε ένα ψηφίο. Το ίδιο και για τα υπόλοιπα γράμματα, στα οποία όμως θα ανατεθεί διαφορετικό ψηφίο, από αυτό του E. Το ζητούμενο είναι μετά από τις όποιες αναθέσεις, να ισχύει η σχέση. Το πρόβλημα διατυπώνεται στον επιλυτή ως εξής:

```
NsProblemManager  pm;

NsIntVar  S(pm,1,9), E(pm,0,9), N(pm,0,9), D(pm,0,9),
          M(pm,1,9), O(pm,0,9), R(pm,0,9), Y(pm,0,9);

NsIntVar  send  =           1000*S + 100*E + 10*N + D;
NsIntVar  more  =           1000*M + 100*O + 10*R + E;
NsIntVar  money = 10000*M + 1000*O + 100*N + 10*E + Y;

pm.add( send + more == money );

NsIntVarArray  letters;
letters.push_back( S );
letters.push_back( E );
letters.push_back( N );
letters.push_back( D );
letters.push_back( M );
letters.push_back( O );
letters.push_back( R );
letters.push_back( Y );
pm.add( NsAllDiff(letters) );

pm.addGoal( new NsgLabeling(letters) );
if (pm.nextSolution() != false)   {
    cout << "    " << send.value() << "\n"
         << " +  " << more.value() << "\n"
         << " = " << money.value() << "\n";
}
```

Το αποτέλεσμα που προκύπτει αν τρέξουμε το εκτελέσιμο για το παραπάνω πρόγραμμα, είναι το εξής:

```
   9567
 +  1085
 = 10652
```

Παράρτημα Β′

Η διεπαφή για την κατάρτιση ωρολογίου προγράμματος

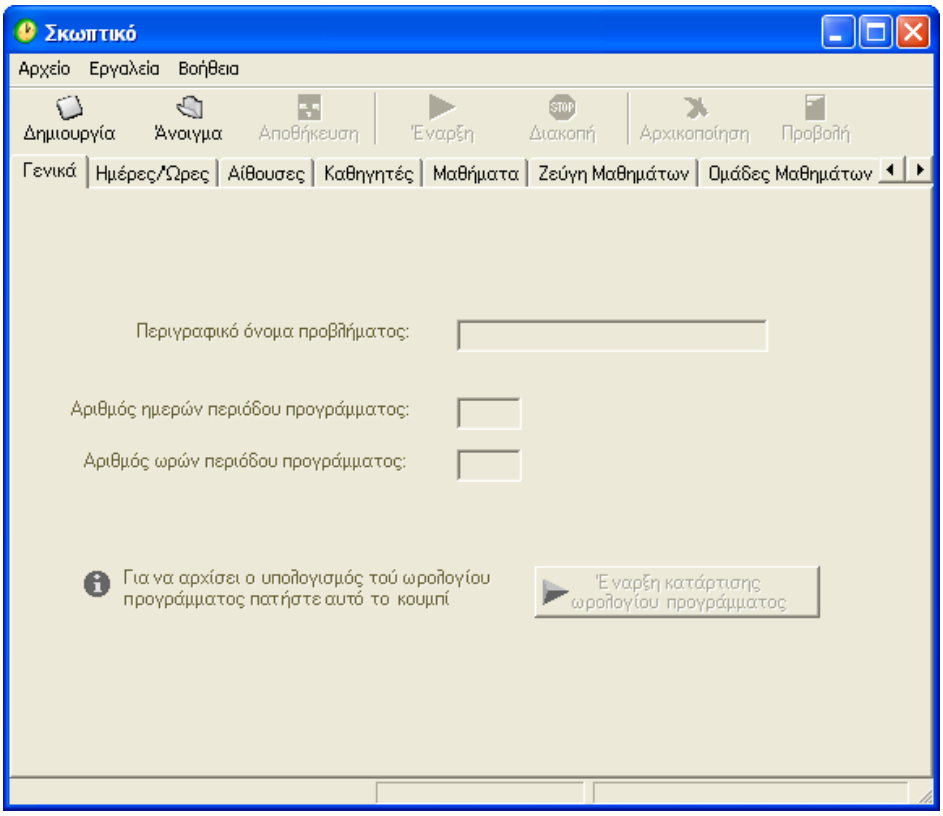
Η διεπαφή της εφαρμογής υλοποιήθηκε σε Visual Basic .NET [38, 39]. Οι φόρμες της είναι κατασκευασμένες έτσι ώστε ο χρήστης να μπορεί να παρέχει γρήγορα και απλά τα στοιχεία ενός προβλήματος ωρολογίου προγράμματος. Τα προβλήματα με τα οποία καταπιάνεται η εφαρμογή καθώς και ο τύπος των αποτελεσμάτων που αυτή παράγει ταυτίζονται με τα αντίστοιχα του διαδικτυακού συστήματος κατάρτισης ωρολογίου προγράμματος μαθημάτων ORPROG.

Β′.1 Δημιουργία ενός προβλήματος

Όταν τρέξουμε το πρόγραμμα εμφανίζεται η Οθόνη Β′.1. Στην αρχή της γραμμής εργαλείων (toolbar) υπάρχει το πλήκτρο **Δημιουργία**. Αν το πατήσουμε εμφανίζεται μια οθόνη διαλόγου στην οποία θα εισαγάγουμε τις βασικές παραμέτρους του νέου προβλήματος που θα φτιάξουμε. Αυτές είναι το όνομά του και ο αριθμός των ημερών και των ωρών του ωρολογίου προγράμματος. Όταν συμπληρώσουμε αυτές τις πληροφορίες πατάμε OK. Οι «καρτέλες» της κεντρικής διεπαφής ενεργοποιούνται πλέον. Καθεμία από αυτές χρησιμοποιείται για την περιγραφή ξεχωριστών πλευρών του προβλήματος.

Β′.2 Ημέρες/Ώρες

Επιλέγοντας την καρτέλα αυτή έχουμε τη δυνατότητα να μετονομάσουμε τις υπάρχουσες ημέρες και ώρες. Π.χ. μπορούμε να κάνουμε κλικ πάνω στο προ-



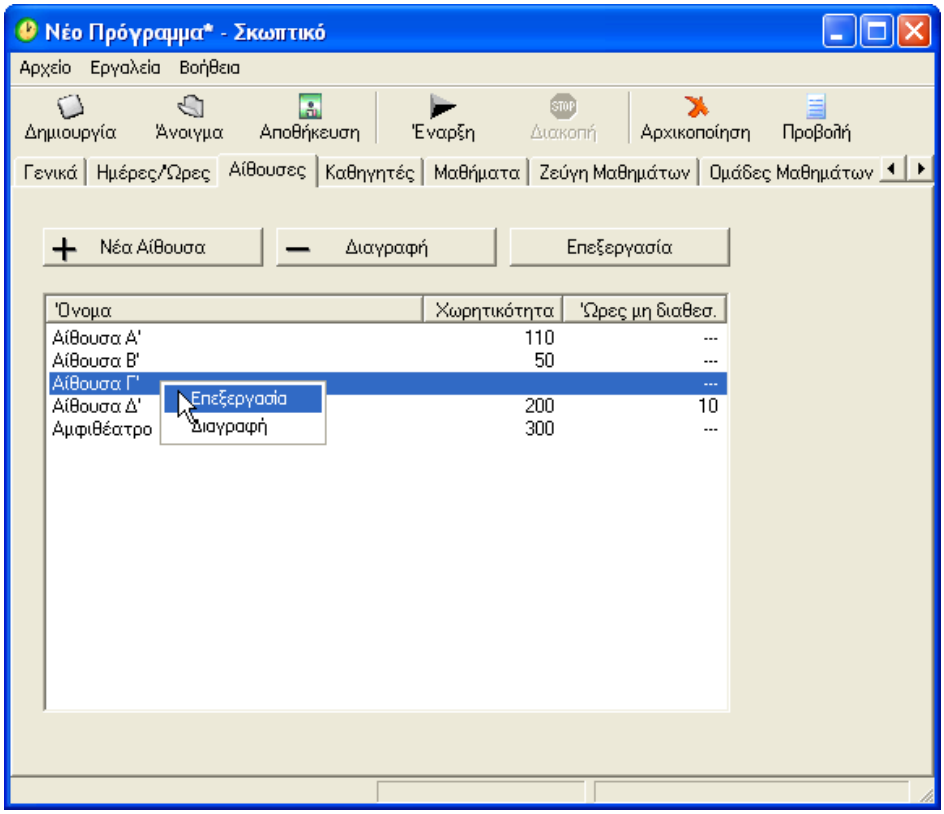
Σχήμα Β′.1: Αρχική οθόνη

καθορισμένο όνομα **1η ημέρα** και να γράψουμε **Δευτέρα**. Υπάρχει επίσης η δυνατότητα αύξησης του αριθμού των ημερών και των ωρών. Με τους όρους «ημέρες» και «ώρες» εννοούμε διδακτικές ημέρες και διδακτικές ώρες. Έτσι μία διδακτική ώρα θα μπορούσε να διαρκεί χρόνο διαφορετικό της μιας ώρας, π.χ. να είναι 45-λεπτη και να φέρει το όνομα 9:00–9:45.

Β′.3 Αίθουσες

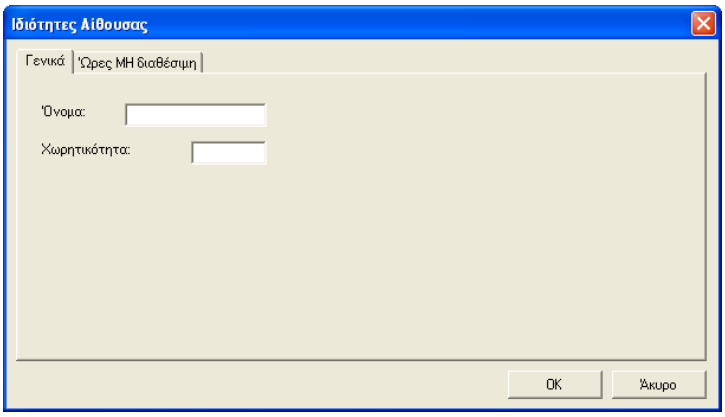
Οι καρτέλες που θα περιγραφούν από δω και στο εξής αφορούν σύνολα οντοτήτων του προβλήματος, όπως αίθουσες, καθηγητές, μαθήματα. Καθεμία καρτέλα ουσιαστικά δεν διαφέρει από τις άλλες στην εμφάνιση και στον τρόπο λειτουργίας της. Όλες περιέχουν έναν «πίνακα-λίστα» με τις αντίστοιχες οντότητες (στη λίστα της καρτέλας **Αίθουσες** για παράδειγμα, περιέχονται όλες οι

αίθουσες). Οι λειτουργίες του πίνακα-λίστας είναι κοινές για όλες τις παρακάτω καρτέλες, για αυτό αρκεί να περιγράψουμε μόνο το τι γίνεται στην καρτέλα **Αίθουσες** (Οθόνη Β΄.2).

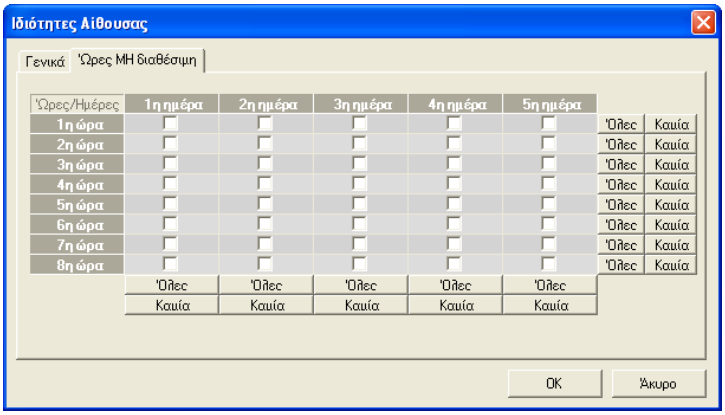


Σχήμα Β΄.2: Καρτέλα **Αίθουσες**

Πατώντας το κουμπί **Νέα Αίθουσα** εμφανίζεται ένα παράθυρο διαλόγου για να συμπληρώσουμε τα στοιχεία της αίθουσας που θέλουμε να προσθέσουμε στη λίστα. Αν επιλέξουμε με το ποντίκι ένα στοιχείο της λίστας —μια αίθουσα εν προκειμένω—, μπορούμε να το επεξεργαστούμε ή να το διαγράψουμε, πατώντας το πλήκτρο **Επεξεργασία** ή **Διαγραφή** αντίστοιχα. Το ίδιο αποτέλεσμα παίρνουμε αν κάνουμε δεξί κλικ στην αίθουσα και χρησιμοποιήσουμε το αναδυόμενο μενού (context menu). Τέλος, το να κάνουμε διπλό κλικ σε κάποιο στοιχείο της λίστας, είναι ισοδύναμο με το να πατήσουμε το κουμπί **Επεξεργασία**, αλλά σαφώς πιο εύχρηστο από το τελευταίο. Να τονίσουμε πάλι ότι τα παραπάνω ισχύουν για τις λίστες που υπάρχουν σε όλες τις καρτέλες.



Σχήμα Β΄.3: Παράθυρο διαλόγου για τις αίθουσες



Σχήμα Β΄.4: Καρτέλα Ώρες ΜΗ διαθέσιμη

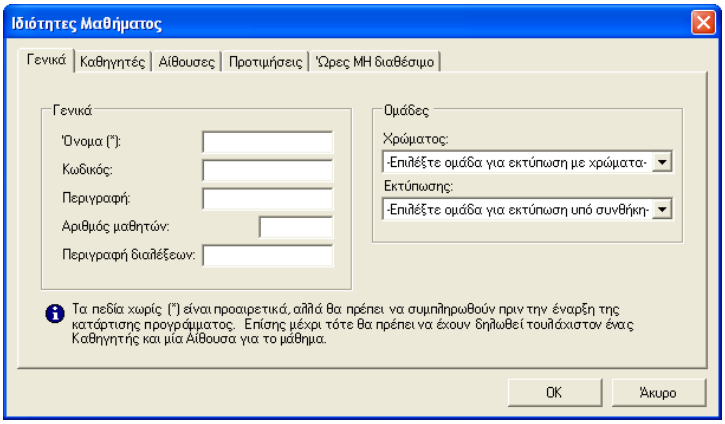
Όταν εισάγουμε ή επεξεργαζόμαστε μία οντότητα, εμφανίζεται ένα παράθυρο διαλόγου με τα αντίστοιχα πεδία-χαρακτηριστικά της. Όσον αφορά την αίθουσα έχουμε την Οθόνη Β΄.3 η οποία περιέχει δύο καρτέλες. Στην πρώτη εισάγουμε το **Όνομα** και τη **Χωρητικότητα** της αίθουσας. Η δεύτερη καρτέλα που ονομάζεται **Ώρες ΜΗ διαθέσιμη** (Οθόνη Β΄.4), υπάρχει και στις οντότητες των καθηγητών και των μαθημάτων. Περιέχει έναν δισδιάστατο πίνακα στον οποίο απεικονίζονται οι μέρες και οι ώρες του ωρολογίου προγράμματος σε "check-box." Αν επιλέξουμε ένα check-box του πίνακα, δηλώνουμε ότι τη συγκεκριμένη ημέρα και ώρα η αίθουσα δεν είναι διαθέσιμη. Κάτω από κάθε στήλη και δεξιά από κάθε γραμμή βρίσκονται δύο κουμπιά: το **Όλες** και το

Καμία. Αυτά ενεργοποιούν και απενεργοποιούν αντίστοιχα όλες τις ώρες, στη στήλη ή στη γραμμή που αναφέρονται.[1]

Β′.4 Καθηγητές

Στο παράθυρο διαλόγου **Ιδιότητες Καθηγητή** θα πρέπει να συμπληρώσουμε το πεδίο **Ονοματεπώνυμο**, όπως επίσης και τις **συνεχόμενες ώρες**, **ώρες/ημέρα** και **ημέρες/εβδομάδα** που μπορεί το πολύ να διδάξει. Μπορούμε να εισαγάγουμε και τις ώρες που δεν θα είναι διαθέσιμος (βλ. παραπάνω).

Β′.5 Μαθήματα



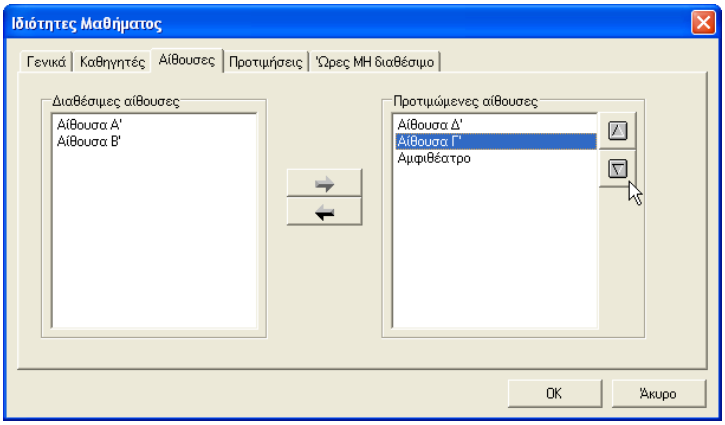
Σχήμα Β′.5: Παράθυρο διαλόγου για τα μαθήματα

Στην Οθόνη Β′.5 βλέπουμε την πρώτη καρτέλα του παράθυρου διαλόγου **Ιδιότητες Μαθήματος**. Υπάρχουν τα πεδία **Όνομα**, **Κωδικός**, **Περιγραφή**. Στο πεδίο **Αριθμός μαθητών** εισάγουμε τον αριθμό των φοιτητών που θα παρακολουθήσουν το μάθημα, έτσι ώστε να μπορεί να ελεγχθεί αν επαρκεί η χωρητικότητα των αιθουσών στις οποίες θα γίνουν οι διαλέξεις.

[1] Τα control (check-box και κουμπιά) στην καρτέλα δημιουργούνται δυναμικά. Με τη λέξη «δυναμικά» εννοούμε ότι τα control σχεδιάζονται και τοποθετούνται την ώρα εκτέλεσης του προγράμματος (run-time), αφού ο αριθμός τους δεν είναι σταθερός. Για παράδειγμα αν ο αριθμός των ωρών του ωρολογίου προγράμματος είναι 8, τότε ο αριθμός των γραμμών από check-box θα είναι 8, ενώ ομοίως αν οι ώρες είναι 10, θα παραχθούν 10 γραμμές από check-box.[40]

Το πεδίο **Περιγραφή διαλέξεων** συμπληρώνεται με μια λίστα με τις ώρες που θα διαρκέσει κάθε διάλεξη του μαθήματος, διαχωρισμένες μεταξύ τους με ένα "+". Π.χ. έστω ότι ένα μάθημα με τίτλο **Φυσική** παραδίδεται σε δυο διαλέξεις (την εβδομάδα): μία των 3 ωρών και μία των 2. Τότε στο πεδίο **Περιγραφή διαλέξεων** θα γράψουμε 2+3. Αν είχαμε μία διάλεξη 4 ωρών θα πληκτρολογούσαμε 4 και αν είχαμε τρεις δίωρες διαλέξεις θα γράφαμε 2+2+2.

Στα πεδία **Ομάδα Χρώματος** και **Ομάδα Εκτύπωσης** επιλέγουμε τις αντίστοιχες ομάδες στις οποίες ανήκει το μάθημα, που θα πρέπει όμως να έχουν δημιουργηθεί προηγουμένως (βλ. §Β′.8, Β′.9). Δεν αφορούν παραμέτρους του προβλήματος κατάρτισης ωρολογίου προγράμματος, αλλά βοηθούν στην καλύτερη παρουσίαση του τελικού αποτελέσματος.



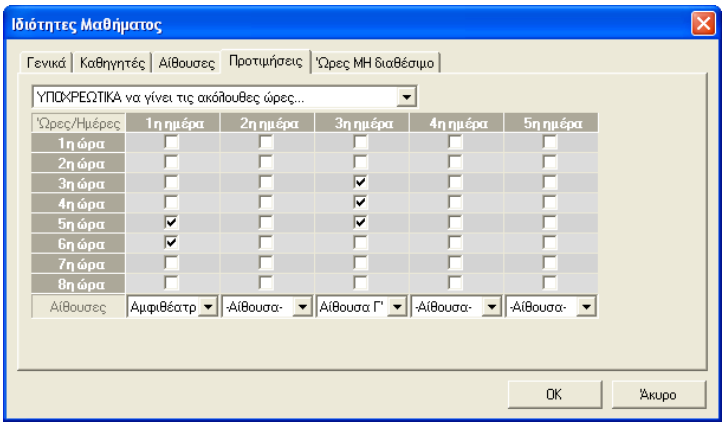
Σχήμα Β′.6: Καρτέλα **Αίθουσες**

Στην καρτέλα με τις **Αίθουσες** του μαθήματος (Οθόνη Β′.6) εισάγουμε τις προτιμώμενες αίθουσες για το μάθημα. Ένα μάθημα μπορεί να γίνει σε μία ή περισσότερες ή και όλες τις προτιμώμενες αίθουσές του —δεδομένου ότι αποτελείται από διαλέξεις οι οποίες μπορούν να γίνονται σε διαφορετικές αίθουσες—, αλλά *όχι* σε αίθουσα που δεν έχει προταθεί. Η διεπαφή για την επιλογή των αιθουσών συμπίπτει με αυτή των καθηγητών του μαθήματος, όπως και με αυτή που υπάρχει στο παράθυρο διαλόγου **Ιδιότητες Ομάδας Μαθημάτων** που θα σχολιαστεί παρακάτω.

Στην Οθόνη Β′.6 υπάρχουν δύο «κουτιά-λίστες»: οι **Διαθέσιμες αίθουσες** και οι **Προτιμώμενες αίθουσες**. Για να μεταφέρουμε μία αίθουσα από τη μία λίστα στην άλλη, την επιλέγουμε κάνοντας κλικ πάνω της και στη συνέχεια πατάμε ένα από τα κουμπιά $\boxed{\rightarrow}$ και $\boxed{\leftarrow}$, ανάλογα με τη λίστα στην οποία θα

την τοποθετήσουμε. Ένας πιο γρήγορος τρόπος για την ίδια λειτουργία, είναι να κάνουμε διπλό κλικ στην αίθουσα.

Οι προτιμώμενες αίθουσες είναι διατεταγμένες, έχουν δηλαδή σειρά προτεραιότητας. Για να αυξήσουμε την προτεραιότητα μίας αίθουσας, μπορούμε να «ανεβάσουμε» τη θέση της και αντιστρόφως: Επιλέγουμε με απλό κλικ την αίθουσα που θέλουμε να της αλλάξουμε σειρά στη λίστα **Προτιμώμενες αίθουσες**. Για να της ανεβάσουμε τη θέση πατάμε $\boxed{\triangle}$ και για να την κατεβάσουμε το $\boxed{\triangledown}$.



Σχήμα Β′.7: Καρτέλα **Προτιμήσεις**

Οι δύο τελευταίες καρτέλες του παράθυρου διαλόγου μοιάζουν. Η τελευταία με όνομα **Ώρες ΜΗ διαθέσιμο** παίζει τον ίδιο ρόλο με την ομώνυμη καρτέλα στο παράθυρο διαλόγου για τις αίθουσες (Οθόνη Β′.4). Η προτελευταία καρτέλα με όνομα **Προτιμήσεις** διαφέρει από την παραπάνω στο ότι κάτω από κάθε στήλη υπάρχει και ένα πεδίο με τις διαθέσιμες αίθουσες. Στο πάνω μέρος υπάρχει ένα πεδίο στο οποίο δηλώνουμε εάν θέλουμε να συμπεριλάβουμε ή όχι το μάθημα στο ωρολόγιο πρόγραμμα ή το εάν επιθυμούμε να προκαθορίσουμε τις ώρες στις οποίες θα γίνεται.

Στην τελευταία περίπτωση θα πρέπει να επιλέξουμε τις προκαθορισμένες ώρες στον πίνακα, σύμφωνα με ό,τι έχουμε γράψει στο πεδίο **Περιγραφή διαλέξεων**. Εδώ *παίζει ρόλο* η σειρά των διαλέξεων. Για παράδειγμα η περιγραφή των διαλέξεων για την Οθόνη Β′.7 είναι 2+3 και όχι 3+2. Κάτω από κάθε ημέρα που υπάρχει μια διάλεξη επιλέγουμε την αίθουσα στην οποία αυτή θα γίνεται. Η αίθουσα πρέπει να περιλαμβάνεται στις προτιμώμενες αίθουσες του μαθήματος.

Β′.6 Ζεύγη ταυτόχρονων μαθημάτων

Πολλές φορές μία αίθουσα δεν φτάνει για τις διαλέξεις ενός μαθήματος. Η έλλειψη χωρητικότητας ή διάφορες συγκυρίες μπορεί να οδηγήσουν στη χρήση της μεθόδου της τηλεσυνδιάσκεψης. Με τη μέθοδο αυτή, οι διαλέξεις ενός μαθήματος καταλαμβάνουν δύο αίθουσες και όχι μία όπως συνήθως. Η διεπαφή δεν μας δίνει τη δυνατότητα να δηλώσουμε άμεσα ότι ένα μάθημα Α παραδίδεται με τηλεσυνδιάσκεψη. Αυτό θα διατυπωθεί «περιφραστικά».

Έστω ένα μάθημα Μ1, με καθηγητές τους Κ1 και Λ1, το οποίο διδάσκεται με τηλεσυνδιάσκεψη στις αίθουσες **Αίθουσα Α′** και **Αίθουσα Β′**. Για να το δηλώσουμε αυτό, ακολουθούμε τα εξής βήματα:

- Δημιουργούμε «αντίγραφα» των καθηγητών του μαθήματος. Εδώ φτιάχνουμε τους Κ2 και Λ2. (Ωστόσο, η διεπαφή μάς δίνει τη δυνατότητα, αν το επιθυμούμε, να έχουμε δύο καθηγητές με το ίδιο όνομα. Έτσι αντί για τους Κ2 και Λ2, θα μπορούσαμε να φτιάξουμε τους Κ1 και Λ1.) Οι περιορισμοί που υπάρχουν στους Κ1 και Λ1, δεν είναι ανάγκη να περαστούν και στα «αντίγραφά» τους Κ2 και Λ2. Να σημειωθεί επίσης ότι η ίδια διαδικασία ακολουθείται, ανεξαρτήτως του αριθμού των καθηγητών του μαθήματος, που εδώ τυχαίνει να είναι δύο.

- Δημιουργούμε ένα «αντίγραφο» του μαθήματος Μ1, έστω Μ2. (Και εδώ αντί για Μ2 θα μπορούσαμε να το ονομάσουμε Μ1.) Βάζουμε την ίδια περιγραφή διαλέξεων και στα δύο μαθήματα. Όσον αφορά τους καθηγητές, το Μ1 έχει τους Κ1 και Λ1, ενώ το Μ2 έχει τους Κ2 και Λ2. Στις αίθουσες του Μ1 πρέπει να υπάρχει η **Αίθουσα Α′** και σε αυτές του Μ2 η **Αίθουσα Β′**. Οι περιορισμοί του Μ1, δηλαδή οι ώρες που τυχόν είναι προκαθορισμένο να γίνεται, καθώς και οι ώρες που δεν είναι διαθέσιμο, δεν χρειάζεται να αντιγραφούν στο Μ2.

- Φτιάχνουμε ένα ζεύγος ταυτόχρονων μαθημάτων {Μ1,Μ2} στην καρτέλα **Ζεύγη Μαθημάτων**.

Β′.7 Ομάδες μαθημάτων

Σε ένα εκπαιδευτικό ίδρυμα, όπως μια πανεπιστημιακή σχολή, ένας φοιτητής έχει το δικαίωμα να παρακολουθεί όποια μαθήματα τον ενδιαφέρουν, ανεξαρτήτως εξαμήνου. Με άλλα λόγια, επιτρέπεται μια ευρεία επιλογή διαφορετικών

μαθημάτων από κάθε φοιτητή. Συνεπώς, το ωρολόγιο πρόγραμμα δεν είναι δυνατόν να ικανοποιήσει πλήρως όλους τους φοιτητές, που πιθανότατα έχουν αντικρουόμενες προτιμήσεις. Είναι σύνηθες φαινόμενο κάποιος να δυσανασχετεί όταν θέλει να παρακολουθήσει μαθήματα των οποίων οι διαλέξεις αλληλεπικαλύπτονται στον χρόνο, έχουν κενά μεταξύ τους, ή δεν ισοκατανέμονται στις ημέρες τις εβδομάδας.

Οι Ομάδες Μαθημάτων υπάρχουν για να βάζουν κάποιους κανόνες, ώστε να προκύψει το αντικειμενικά καλύτερο αποτέλεσμα. Τα μαθήματα που ανήκουν στην ίδια ομάδα δεν επιτρέπεται να γίνονται ταυτόχρονα, ενώ η μηχανή επίλυσης, ανάλογα με τη βαρύτητα της ομάδας, προσπαθεί να μην αφήνει κενά ανάμεσα στις διαλέξεις τους και να τα ισοκατανέμει μέσα στην εβδομάδα. Ένα παράδειγμα μιας ομάδας μαθημάτων που απαιτεί τα παραπάνω είναι τα μαθήματα συγκεκριμένου εξάμηνου σε μια σχολή. Οι φοιτητές του αντίστοιχου εξάμηνου προτιμούν συνήθως να τα παρακολουθούν όλα.

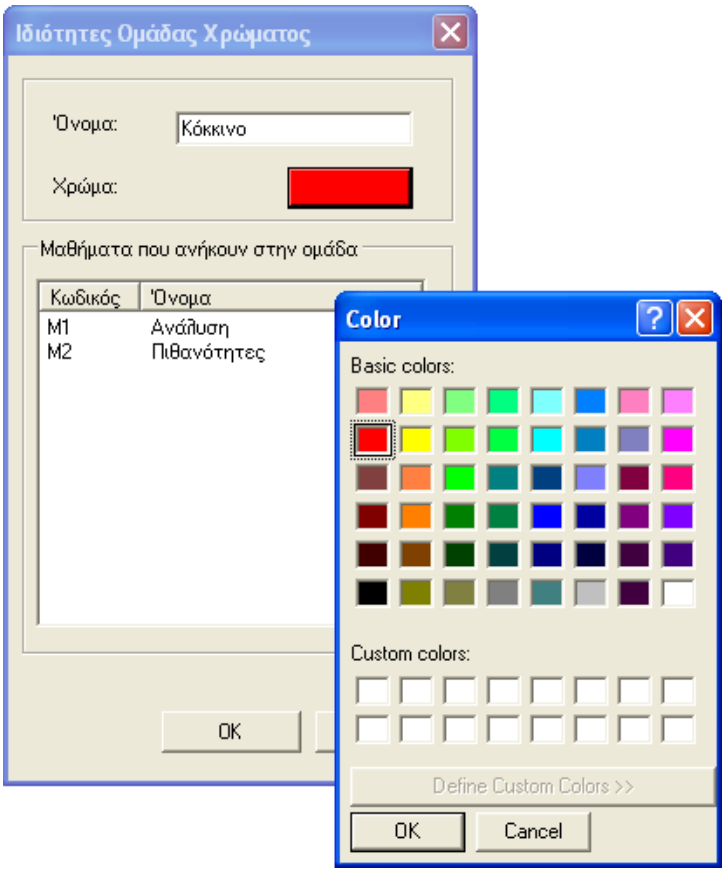
Στο παράθυρο διαλόγου **Ιδιότητες Ομάδας Μαθημάτων** συμπληρώνουμε το **Όνομα** της ομάδας και τη **Βαρύτητά** της. Ο τρόπος που επιλέγουμε τα μαθήματα της ομάδας, είναι ο ίδιος με αυτόν που επιλέγουμε τις αίθουσες ενός μαθήματος (βλ. Οθόνη Β′.6).

Τα δυο μαθήματα ενός ζεύγους ταυτόχρονων μαθημάτων δεν πρέπει να ανήκουν στην ίδια ομάδα· αρκεί να ανήκει το ένα. Ο λόγος που ισχύει αυτό είναι ότι δυο μαθήματα που ανήκουν στην ίδια ομάδα δεν μπορούν να γίνονται ταυτόχρονα!

Β′.8 Ομάδες μαθημάτων για εκτύπωση με χρώματα

Οι Ομάδες Χρώματος είναι, όπως ίσως φαντάζεται κανείς, κάτι που αφορά τη διεπαφή και όχι τη μηχανή υπολογισμού: Τα μαθήματα που ανήκουν σε μια τέτοια ομάδα, προβάλλονται στο τελικό ωρολόγιο πρόγραμμα με φόντο το χρώμα της.

Στο παράθυρο **Ιδιότητες Ομάδας Χρώματος** επιλέγουμε το **Όνομα** και το **Χρώμα** της ομάδας, ενώ στο «κουτί» που υπάρχει μπορούμε να δούμε ποια μαθήματα ανήκουν στην ομάδα (Οθόνη Β′.8). Δεν μπορούμε να προσθαφαιρέσουμε σε αυτό το παράθυρο τα μαθήματα. Για να το κάνουμε αυτό, επεξεργαζόμαστε το ίδιο το μάθημα που θέλουμε να προσθαφαιρέσουμε.



Σχήμα Β΄.8: Παράθυρο διαλόγου Ιδιότητες Ομάδας Χρώματος

Β΄.9 Ομάδες μαθημάτων για εκτύπωση υπό συνθήκη

Ορισμένες φορές, στο τελικό ωρολόγιο πρόγραμμα, είναι επιθυμητό να απομονωθούν κάποιες κατηγορίες μαθημάτων και να αποκρυφτούν οι υπόλοιπες. Ένα πρακτικό παράδειγμα είναι ο διαχωρισμός των προπτυχιακών και μεταπτυχιακών μαθημάτων ενός τμήματος: Όλα αυτά τα μαθήματα γίνονται στις ίδιες αίθουσες και ως εκ τούτου συμμετέχουν στην κατάρτιση του ίδιου ωρολογίου προγράμματος. Όμως θα ήταν χρήσιμο να μπορούμε να φτιάξουμε ένα ξεχωριστό πρόγραμμα για τα προπτυχιακά και ένα άλλο για τα μεταπτυχιακά μαθήματα.

Μια Ομάδα Εκτύπωσης έχει πεδία το **Όνομά** της και τον Χαρακτηριστικό αριθμό της. Αν φτιάξουμε τις ομάδες εκτύπωσης **Προπτυχιακά** και **Μεταπτυχιακά** και επεξεργαστούμε όλα τα μαθήματα έτσι ώστε να ανήκουν σε μία από αυτές τις δύο ομάδες, τότε, κατά την προβολή του ωρολογίου προγράμματος (§Β′.11), θα έχουμε τη δυνατότητα να φτιάξουμε ξεχωριστά προγράμματα για την κάθε ομάδα. Τα μαθήματα που δεν ανήκουν σε κάποια ομάδα εκτύπωσης θα τυπώνονται πάντοτε.

Β′.10 Κατάρτιση ωρολογίου προγράμματος

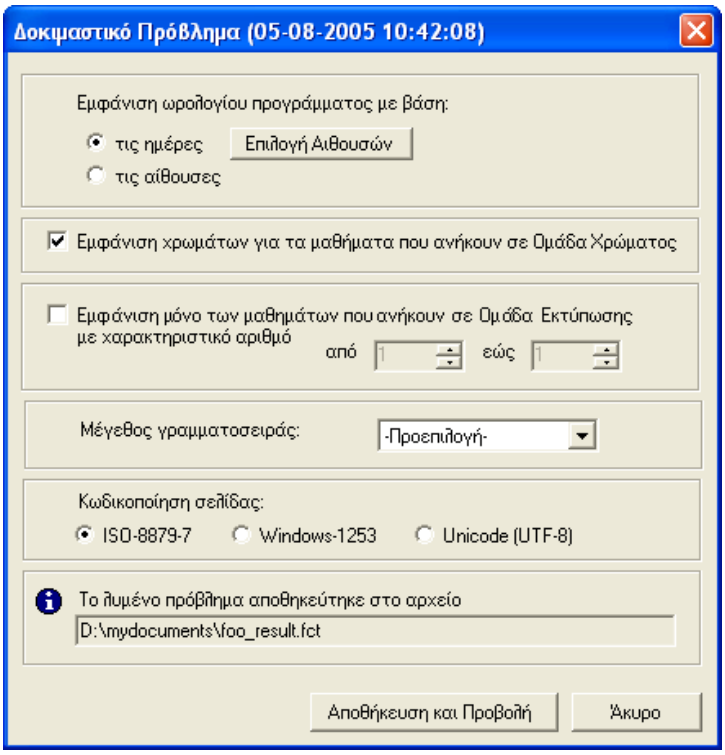
Για να αρχίσει η κατάρτιση του ωρολογίου προγράμματος πατάμε το κουμπί **Έναρξη** στη γραμμή εργαλείων (toolbar) της εφαρμογής. Αν έχουμε δώσει σωστά και ολοκληρωμένα όλες τις παραμέτρους, αυτό που θα κάνει η εφαρμογή είναι να αντιγράψει στη μνήμη της το τρέχον στιγμιότυπο του προβλήματος και να μεταβιβάσει στη μηχανή υπολογισμού τα στοιχεία που χρειάζεται για να ξεκινήσει μια ίσως χρονοβόρα αναζήτηση λύσης.

Μέχρι να ολοκληρωθεί η αναζήτηση, η εφαρμογή είναι στη διάθεση του χρήστη για τροποποίηση του τρέχοντος προβλήματος, ή για φόρτωση ενός άλλου. Όταν η αναζήτηση ολοκληρωθεί επιτυχώς, τότε το αποτέλεσμα έρχεται να ενωθεί με το αντίγραφο του στιγμιότυπου που αποθηκεύτηκε προηγουμένως. Η λύση του προβλήματος αποθηκεύεται σε ένα αρχείο με κατάληξη `_result.fct`. Π.χ. αν τη στιγμή που ξεκινούσαμε τον υπολογισμό επεξεργαζόμασταν το αρχείο `foo.fct`, στο τέλος θα πάρουμε το `foo_result.fct`. Σε κάθε περίπτωση ο χρήστης ενημερώνεται για το όνομα αυτό. Το αρχείο της λύσης είναι όμοιο με το αρχικό, με τη διαφορά ότι όλα τα μαθήματα έχουν προκαθορισμένες ώρες, τις οποίες ο χρήστης έχει την ευχέρεια να αλλάξει.

Β′.11 Προβολή ωρολογίου προγράμματος

Με το πέρας ενός υπολογισμού, ή πατώντας το πλήκτρο **Προβολή** στη γραμμή εργαλείων όταν έχουμε φορτώσει ένα αρχείο με αποτελέσματα (π.χ. το `foo_result.fct`), εμφανίζεται η Οθόνη Β′.9.

Σε αυτήν μπορούμε να επιλέξουμε αν το πρόγραμμα θα εμφανιστεί με βάση τις ημέρες (ένας πίνακας για κάθε ημέρα), ή τις αίθουσες (ένας πίνακας για κάθε αίθουσα). Στην πρώτη περίπτωση, κάθε στήλη των πινάκων θα αντιστοιχεί σε μία αίθουσα. Αν έχουμε πολλές αίθουσες, το πλάτος του πίνακα θα είναι



Σχήμα Β′.9: Παράθυρο διαλόγου για την προβολή του τελικού προγράμματος

μεγάλο. Πατώντας το κουμπί **Επιλογή Αιθουσών** έχουμε τη δυνατότητα να αποκλείσουμε κάποιες αίθουσες, έτσι ώστε να φτιάξουμε ένα πιο μικρό πρόγραμμα. (Εν συνεχεία μπορούμε να ξαναπροβάλλουμε το πρόγραμμα επιλέγοντας τις υπόλοιπες αίθουσες.)

Υπάρχει η επιλογή για προβολή των χρωμάτων για τα μαθήματα που ανήκουν σε ομάδες χρώματος. Ένας χρήστης μπορεί να διαλέξει να προβληθούν τα μαθήματα που ανήκουν σε ομάδες εκτύπωσης με χαρακτηριστικό αριθμό εντός συγκεκριμένου εύρους. Τα μαθήματα που δεν ανήκουν σε ομάδες εκτύπωσης τυπώνονται πάντοτε. Τέλος, γίνεται να ρυθμίσουμε το μέγεθος της γραμματοσειράς και την κωδικοποίηση της σελίδας στην οποία θα αποθηκευτεί το ωρολόγιο πρόγραμμα.

Πατώντας το πλήκτρο **Αποθήκευση και Προβολή** το πρόβλημα αποθηκεύεται σε ένα αρχείο HTML και έπειτα προβάλλεται με τον φυλλομετρητή (Οθόνη Β′.10).

Τρίτη				
Ωρα / Αίθουσα	Α	Β	Γ	Ε
09-10	**Διακριτά Μαθηματικά** Β' Εξάμηνο - Κ *Εμίρης Ι.*	**Ειδικά Θέματα Υπολογιστικών Συστημάτων και Εφαρμογών Πληροφορικής** Η' Εξάμηνο - Ε2 *Χαλάτσης Κ.*		**Διακριτά Μαθηματικά** Β' Εξάμηνο - Κ *Εμίρης Ι.*
10-11	**Διακριτά Μαθηματικά** Β' Εξάμηνο - Κ *Εμίρης Ι.*	**Ειδικά Θέματα Υπολογιστικών Συστημάτων και Εφαρμογών Πληροφορικής** Η' Εξάμηνο - Ε2 *Χαλάτσης Κ.*		**Διακριτά Μαθηματικά** Β' Εξάμηνο - Κ *Εμίρης Ι.*
11-12	**Αντικειμενοστραφής Προγραμματισμός** Β' Εξάμηνο - Κ *Καράλη Ι.*	**Ειδικά Θέματα Υπολογιστικών Συστημάτων και Εφαρμογών Πληροφορικής** Η' Εξάμηνο - Ε2 *Χαλάτσης Κ.*		**Αντικειμενοστραφής Προγραμματισμός** Β' Εξάμηνο - Κ *Καράλη Ι.*
12-13	**Αντικειμενοστραφής Προγραμματισμός** Β' Εξάμηνο - Κ *Καράλη Ι.*	**Επεξεργασία Ομιλίας** Η' Εξάμηνο - Ε3 *Κουρουπέτρογλου Γ.*		**Αντικειμενοστραφής Προγραμματισμός** Β' Εξάμηνο - Κ *Καράλη Ι.*
13-14	**Δομές Δεδομένων** Β' Εξάμηνο - Κ *Μισυρλής Ν.*	**Επεξεργασία Ομιλίας** Η' Εξάμηνο - Ε3 *Κουρουπέτρογλου Γ.*	**Ανάλυση και Σχεδίαση Ηλεκτρονικών και Τηλεπικοινωνιακών Κυκλωμάτων** ΣΤ' Εξάμηνο - Ε2 Ε3 *Μολυμπάκης Μ.*	**Δομές Δεδομένων** Β' Εξάμηνο - Κ *Μισυρλής Ν.*
14-15	**Δομές Δεδομένων** Β' Εξάμηνο - Κ *Μισυρλής Ν.*	**Επεξεργασία Ομιλίας** Η' Εξάμηνο - Ε3 *Κουρουπέτρογλου Γ.*	**Ανάλυση και Σχεδίαση Ηλεκτρονικών και Τηλεπικοινωνιακών Κυκλωμάτων** ΣΤ' Εξάμηνο - Ε2 Ε3 *Μολυμπάκης Μ.*	**Δομές Δεδομένων** Β' Εξάμηνο - Κ *Μισυρλής Ν.*

Σχήμα Β′.10: Μέρος ενός καταρτισμένου ωρολογίου προγράμματος

Β′.12 Ο τύπος αρχείων fct

Τα προβλήματα αποθηκεύονται σε αρχεία τύπου fct. Τα αρχεία αυτά είναι συμβατά με το διάγραμμα οντοτήτων-συσχετίσεων που παρουσιάστηκε στην §4.4. Στο παρακάτω πλαίσιο βλέπουμε ένα παράδειγμα των περιεχομένων ενός fct αρχείου.

```
DAYS 2
HOURS 3
problemname('Μικρό πρόγραμμα').
dayname(1,'Δευτέρα').
dayname(2,'Τρίτη').
hourname(1,'8:00-9:00').
hourname(2,'9:00-10:00').
hourname(3,'10:00-11:00').
classroom(502,'Αμφιθέατρο',350,[]).
classroom(500,'Αίθουσα A\'',,[na(1,3),na(2,2),na(2,3)]).
teacher(797,'Παπαδόπουλος Α.',[na(1,1)],0,0,0).
teacher(798,'Bob',[],4,5,3).
teacher(799,'Tom',[],0,0,0).
subject(301,'M1','Αρχαία Γραμματεία',40,2+1,[799],
    3,[0 502 5 502],[500 502],'Θεωρητική Κατεύθυνση',[]).
subject(302,'M2','Ανάλυση',59,2+2,[798,799],
    1,[],[500],'Θετική Κατεύθυνση',[]).
subject(303,'M2','Ανάλυση',59,2+2,[798,799],
    1,[],[502],'Θετική Κατεύθυνση',[]).
subject(304,'','Πιθανότητες',,,[],
    0,[],[],'',[na(3,3)]).
dgroup(7,'Θεωρητικά Μαθήματα',[301]).
cgroup(8,'Θετικά Μαθήματα',[302,304],50).
cgroup(9,'Ξένες Γλώσσες',[],10).
colorgroup(5,'Κόκκινο',[303,302],'FF0000').
printgroup(5,'Απογευματινά Μαθήματα',[304],1000).
equal(303,302).
```

Ένας που γνωρίζει Prolog εύκολα διαπιστώνει ότι το συντακτικό των αρχείων fct έχει δανειστεί πολλά στοιχεία από τη γλώσσα αυτή. Αλλά ας ξεκινήσουμε από την αρχή. Στις δύο πρώτες γραμμές ορίζεται ο αριθμός των ημερών και των ωρών. Στην τρίτη γραμμή διατυπώνεται το όνομα του προβλήματος μέσω του κατηγορήματος[2] `problemname/1` που παίρνει σαν όρισμα μία συμβολοσειρά. Σε αυτό το σημείο να αναφέρουμε ότι οι συμβολοσειρές περικλείονται σε

[2]Μιλώντας με ορολογία της γλώσσα λογικού προγραμματισμού Prolog, οποιαδήποτε έκφραση της μορφής π.χ. "`parent(john,nick).`" είναι ένα γεγονός που εκφράζει ότι ένας ισχυρισμός είναι αληθινός, για παράδειγμα, ότι «ο Γιάννης είναι γονιός του Νίκου». Το σύμβολο `parent` είναι ένα *κατηγόρημα* που χρησιμοποιείται για να εκφράσει κάποια σχέση, αυτήν

απλά εισαγωγικά '. Ένας χαρακτήρας "'" εντός μιας συμβολοσειράς αναπαρίσταται με την ακολουθία διαφυγής "\'". Π.χ. μέσα στο παραπάνω αρχείο fct, υπάρχει το `'Αίθουσα A\''` που μεταφράζεται στη συμβολοσειρά `Αίθουσα A'`.

Τα κατηγορήματα `dayname/2` και `hourname/2` παίρνουν σαν $1^ο$ όρισμα τον αύξοντα αριθμό της ημέρας/ώρας της οποίας ορίζουν το όνομα και σαν $2^ο$ όρισμα το όνομα. Με το `classroom/4` περιγράφουμε μία αίθουσα. Το $1^ο$ όρισμά του είναι ο μοναδικός κωδικός αριθμός της (πρωτεύον κλειδί), το $2^ο$ όρισμα είναι το όνομά της, το $3^ο$ όρισμά της είναι η χωρητικότητά της και στο $4^ο$ υπάρχει μια λίστα με τις ώρες που δεν είναι διαθέσιμη. Μια λίστα στην Prolog περιλαμβάνει τις αγκύλες "[" και "]" εντός των οποίων κλείνονται τα στοιχεία της διαχωρισμένα με κόμματα. Η συγκεκριμένη λίστα περιέχει συναρτησιακά σύμβολα `na`. Το `na(`d`,`h`)` σημαίνει «μη διαθέσιμο την ώρα h της ημέρας d». Π.χ. το `na(2,3)` σημαίνει «μη διαθέσιμο την $3^η$ ώρα της $2^{ης}$ ημέρας».

Το $1^ο$ όρισμα του `teacher/6` είναι το πρωτεύον κλειδί του, το $2^ο$ είναι το ονοματεπώνυμό του καθηγητή, το $3^ο$ είναι οι ώρες που δεν είναι διαθέσιμος σε μια λίστα όμοια με αυτή που υπάρχει στο `classroom/4`. Το $4^ο$, $5^ο$ και $6^ο$ όρισμά του είναι αντίστοιχα οι συνεχόμενες ώρες, οι ώρες/ημέρα και οι ημέρες/εβδομάδα που μπορεί το πολύ να διδάξει.

Το `subject/11` περιέχει όλα τα δεδομένα ενός μαθήματος και των σχέσεων του με τις αίθουσες και τους καθηγητές. Το $1^ο$ όρισμα είναι το πρωτεύον κλειδί, το $2^ο$ είναι ο κωδικός και το $3^ο$ το όνομα του μαθήματος. Το $4^ο$ είναι ο αριθμός των μαθητών που το παρακολουθούν, ενώ το $5^ο$ είναι η περιγραφή των διαλέξεών του. Στο $6^ο$ όρισμα υπάρχει μια λίστα με τα πρωτεύοντα κλειδιά των καθηγητών που διδάσκουν το μάθημα. Το $7^ο$ όρισμα είναι ένας αριθμός με τις εξής ερμηνείες: αν είναι `0`, το μάθημα δεν θα συμπεριληφθεί στην κατάρτιση του ωρολογίου προγράμματος, ενώ αν είναι `1` θα συμβεί το αντίθετο. Αν είναι `3`, τότε το $8^ο$ όρισμα θα περιέχει μια λίστα με τις προκαθορισμένες ώρες και αίθουσες στις οποίες θα γίνονται οι διαλέξεις — για τις προηγούμενες δύο περιπτώσεις (του `0` και του `1`), το $8^ο$ όρισμα θα περιέχει απλώς την κενή λίστα "`[]`". Η λίστα είναι της μορφής [h_1 c_1 h_2 c_2 $\cdots$ h_n c_n]. Τα στοιχεία της διαχωρίζονται με έναν κενό χαρακτήρα —και όχι με κόμματα— και είναι όπως φαίνεται της μορφής "h_i c_i". Ένα τέτοιο στοιχείο αντιστοιχεί στην i-οστή διάλεξη του μαθήματος (όπως αυτή προκύπτει από το $5^ο$ όρισμα) και σημαίνει ότι η τελευταία θα αρχίζει την ώρα h_i στην αίθουσα με πρωτεύον κλειδί c_i. Το h_i είναι ο αύξων αριθμός της ώρας, ήτοι η απόσταση της ώρας από την πρώτη ώρα

του γονιού-παιδιού, μεταξύ δύο οντοτήτων, γι' αυτό και λέμε ότι έχει *βαθμό 2* και μπορούμε να το γράψουμε και σαν `parent/2`.[41]

όλου του ωρολογίου προγράμματος. Π.χ. στο αρχείο fct του παραδείγματος η ώρα 0 είναι η 1^{η} ώρα της $1^{\eta\varsigma}$ ημέρας και η ώρα 5 είναι η 3^{η} ώρα της $2^{\eta\varsigma}$ ημέρας. Να σημειωθεί ότι υπάρχει η σπάνια περίπτωση[3] το c_i να μην υπάρχει, π.χ. να έχουμε τη λίστα "[0␣␣4␣502]".[4] Το 9^o όρισμα του `subject/11` είναι μια λίστα με τα πρωτεύοντα κλειδιά των προτιμώμενων αιθουσών για το μάθημα σε σειρά προτεραιότητας και διαχωρισμένα με κενά. Το 10^o όρισμα είναι η περιγραφή του μαθήματος και το 11^o οι ώρες που αυτό δεν είναι διαθέσιμο, όρισμα που είναι ίδιο στη σύνταξή του με τα αντίστοιχα των `classroom/4` και `teacher/6`.

Οι ομάδες μαθημάτων, ενώ μοιράζονται τα ίδια πρωτεύοντα κλειδιά, εμφανίζονται σε δύο κατηγορήματα, το `dgroup/3` και το `cgroup/4`, ανάλογα με τη βαρύτητά τους. Αν η βαρύτητα είναι μεγαλύτερη του 0, τότε η ομάδα μαθημάτων αναπαρίσταται μέσω του `cgroup/4` και η βαρύτητα μπαίνει στο 4^o όρισμα. Αλλιώς, αν είναι ίση με 0, η ομάδα αναπαρίσταται με το `dgroup/3` και η βαρύτητά της αποσιωπάται. Κατά τα άλλα στο 1^o όρισμα των δύο κατηγορημάτων υπάρχει το πρωτεύον κλειδί, στο 2^o το όνομα της ομάδας και στο 3^o η λίστα με τα πρωτεύοντα κλειδιά των μαθημάτων που ανήκουν σε αυτήν. Όσον αφορά τα 3 πρώτα ορίσματα, τον ίδιο ακριβώς ρόλο εξυπηρετούν και στο `colorgroup/4` για τις ομάδες χρώματος και στο `printgroup/4` για τις ομάδες εκτύπωσης. Το 4^o όρισμα του `colorgroup/4` είναι το χρώμα της ομάδας σε δεκαεξαδική RGB μορφή (όπως η μορφή των χρωμάτων στη γλώσσα HTML). Το 4^o όρισμα στο `printgroup/4` είναι ο χαρακτηριστικός αριθμός της ομάδας εκτύπωσης. Τέλος, το `equal/2` παριστάνει ένα ζεύγος ταυτόχρονων μαθημάτων. Στα ορίσματά του μπαίνουν τα πρωτεύοντα κλειδιά των μαθημάτων που το αποτελούν.

[3] Συμβαίνει όταν διαγράψουμε μία αίθουσα στην οποία έχει προκαθοριστεί να γίνεται κάποια διάλεξη ενός μαθήματος.

[4] Οι χαρακτήρες του κενού διαστήματος οπτικοποιήθηκαν για μεγαλύτερη σαφήνεια.

Παράρτημα Γ′

Σύνδεση της διεπαφής με τη βιβλιοθήκη επίλυσης

Ο συνδυασμός μιας όμορφης διεπαφής με μια ισχυρή μηχανή επίλυσης, εφόσον αμφότερες είναι υλοποιημένες σε διαφορετικές γλώσσες, δεν είναι μια τετριμμένη υπόθεση. Σε αυτό το κεφάλαιο παρουσιάζεται η μέθοδος δημιουργίας ενός DLL,[1] στο οποίο μπορούν να απομονωθούν χρονοβόροι υπολογισμοί, καθώς και η ενσωμάτωσή του σε ένα νήμα της κύριας εφαρμογής.

Γ′.1 Διαδικασία σύνδεσης της Visual Basic .NET με ένα DLL

Η διεπαφή της εφαρμογής σχεδιάστηκε με τη γλώσσα προγραμματισμού Visual Basic .NET. Πρόκειται για μια γλώσσα που είναι ενημερωμένη πάνω στις τελευταίες εξελίξεις του λειτουργικού συστήματος των Windows. Επιτρέπει τη γρήγορη δημιουργία φορμών (διεπαφών) με σύγχρονη εμφάνιση και φιλικότητα προς τον χρήστη.

Αποτελεί μια γλώσσα *υψηλού επιπέδου*· δηλαδή προσεγγίζει περισσότερο την ανθρώπινη γλώσσα και τρόπο σκέψης, από ό,τι μια γλώσσα προγραμματισμού *χαμηλού επιπέδου*, όπως η C για παράδειγμα.

Το τίμημα της φιλικότητας του συντακτικού των γλωσσών υψηλού επιπέδου είναι η σχετικά χαμηλή ταχύτητά τους. Αυτός είναι ένας από τους λόγους για τους οποίους αυτή καθεαυτή η επίλυση των προβλημάτων ωρολογίων προγραμ-

[1] Dynamic-Link Library

μάτων γίνεται μέσα από κώδικα C++. Ο άλλος λόγος για τη χρήση της C++ είναι ότι η βιβλιοθήκη επίλυσης προβλημάτων με περιορισμούς είναι γραμμένη σε C++ και συνεπώς αρχικοποιείται στη γλώσσα αυτή. Μέσα από ένα παράδειγμα που αφορά το λογισμικό της πτυχιακής εργασίας, θα προσπαθήσουμε να περιγράψουμε τη γενικότερη διαδικασία που θα πρέπει να ακολουθήσει κανείς για να διασυνδέσει ένα πρόγραμμα Visual Basic .NET με ένα DLL το οποίο προήλθε από κώδικα C/C++.

Γ′.1.1 Δημιουργία ενός DLL

Ο κώδικας C/C++

Καταρχήν θα περιγράψουμε τον τρόπο δημιουργίας με κώδικα C/C++ ενός DLL, το οποίο θα είναι συμβατό με τις .NET γλώσσες και όχι μόνο [42].

Ένα DLL περιέχει συναρτήσεις. Οι συναρτήσεις αυτές είναι διαθέσιμες σε κάθε πρόγραμμα. Π.χ. δύο διαφορετικά προγράμματα μπορούν να καλούν ταυτόχρονα την ίδια ή και διαφορετικές συναρτήσεις του DLL. Το λειτουργικό σύστημα, για εξοικονόμηση μνήμης, φορτώνει *ένα* στιγμιότυπο του DLL σε αυτήν, ανεξαρτήτως του αριθμού των διεργασιών που το χρησιμοποιούν. Εξάλλου μιλάμε για *δυναμική* βιβλιοθήκη.[2]

Ας περάσουμε όμως σε πιο πρακτικά θέματα. Τα «σημεία» στα οποία κάποιος μπορεί να συνδεθεί με ένα DLL είναι οι συναρτήσεις με πρωτότυπα τύπου C που αυτό περιέχει. Αρκεί λοιπόν να ξέρουμε *το όνομα* της συνάρτησης που μας ενδιαφέρει, *τον αριθμό και τον τύπο των ορισμάτων* της, καθώς και *τον τύπο μεταβλητής που επιστρέφει*.

Ποιους τύπους μεταβλητών μπορούμε να χρησιμοποιήσουμε στο πρωτότυπο μιας συνάρτησης; Όπως αναφέρθηκε παραπάνω, οι συναρτήσεις ενός DLL έχουν *πρωτότυπα τύπου C* και έτσι περιοριζόμαστε στις μεταβλητές της γλώσσας αυτής. Τι γίνεται τότε με τον κώδικα που γράφουμε σε C++; Ένας κανόνας που θα μπορούσαμε να ακολουθήσουμε είναι αντί για κλάσεις (classes) να χρησιμοποιούμε δομές (structs) και αντί για vector να δουλεύουμε με δείκτες (pointers) για να φτιάχνουμε πίνακες. Εφόσον η συνάρτηση C++ ικανοποιεί τα παραπάνω, θα το δηλώσουμε ρητά περικλείοντας τη δήλωσή της (ή το πρωτότυπό της) εντός του `extern "C" { ... }`. Με αυτή τη δήλωση επισημαίνουμε ότι το πρωτότυπο της συνάρτησης είναι συμβατό με τη C. Τα παραπάνω ισχύουν για τις συναρτήσεις οι οποίες θέλουμε να είναι διαθέσιμες για χρήση

[2] Και στα συστήματα UNIX υπάρχουν δυναμικές βιβλιοθήκες σαν τα DLL.

από προγράμματα εκτός DLL. Δεν υπάρχουν περιορισμοί για τις «εσωτερικές» συναρτήσεις.

Παραθέτουμε μια ελαφρώς απλουστευμένη έκδοση της συνάρτησης που υπάρχει στο DLL το οποίο φτιάχτηκε στα πλαίσια της πτυχιακής εργασίας:

```
struct room_t  {
    int  *unavail;
    int  nunavail;
};

extern "C"  {
    char*  orprog (struct room_t *room, int nroom)
    {
     ⋮
    }
}
```

Η συνάρτηση `orprog()` παίρνει σαν ορίσματα έναν πίνακα από δομές αιθουσών και τον αριθμό τους. Κάθε δομή αίθουσας περιέχει ένα πίνακα με τις ώρες που η αίθουσα δεν είναι διαθέσιμη, καθώς και των αριθμό των ωρών αυτών. Η συνάρτηση επιστρέφει μια συμβολοσειρά. Στο σώμα της, που «αποσιωπήθηκε», υπάρχουν και πολλά στοιχεία που αφορούν μόνο στη γλώσσα C++. Ωστόσο το πρωτότυπό της είναι προσεκτικά φτιαγμένο έτσι ώστε να είναι συμβατό με τη C.

Τέλος θα πρέπει να επισημάνουμε κάποια προβλήματα, τα οποία μπορεί να προκύψουν από τον τρόπο που θα γράψουμε τον κώδικα C/C++ για το DLL. *Το DLL πρέπει να είναι ασφαλές σε πολυνηματικό περιβάλλον (threadsafe).* Αυτή η πρόταση, όσον αφορά τις C/C++ μεταφράζεται ως εξής: Απαγορεύονται οι καθολικές (global) και οι στατικές (static) μεταβλητές, εφόσον δεν είναι σταθερές (const). Για να πειστεί κανείς για αυτό, αρκεί να σκεφτεί το τι μπορεί να γίνει στην περίπτωση που δύο προγράμματα που καλούν το ίδιο DLL, επεξεργάζονται ταυτόχρονα την ίδια καθολική μεταβλητή! Θυμηθείτε ότι υπάρχει *ένα* στιγμιότυπο του DLL στη μνήμη και συνεπώς *μία* θέση μνήμης για την καθολική μεταβλητή. Ακόμα θα πρέπει να φροντίσουμε ιδιαιτέρως τη διαχείριση της μνήμης. Το DLL περιέχει συναρτήσεις οι οποίες «επιστρέφουν» με ένα "`return`"· σε καμία περίπτωση δεν απελευθερώνουν τη μνήμη που δεσμεύθηκε κατά τη λειτουργία τους.

Παρακάτω θα αναφέρουμε δύο τρόπους για να μεταγλωττίσουμε τον κώδικα που φτιάξαμε.

Μεταγλώττιση με Visual C++ .NET [43]

Από το μενού του Visual Studio .NET επιλέγουμε File → New → Project. Στο αριστερό μέρος του παραθύρου που εμφανίζεται διαλέγουμε Visual C++ Projects και στο δεξί Win32 Project. Γράφουμε ένα όνομα για το project, έστω `MyProj`, στο αντίστοιχο πεδίο και πατάμε OK. Στο νέο παράθυρο διαλόγου πατάμε τον σύνδεσμο Application Settings που βρίσκεται στα αριστερά. Επιλέγουμε ως Application Type το DLL, ενεργοποιούμε το Export symbols και πατάμε Finish.

Θα ασχοληθούμε με τα αρχεία `MyProj.h` και `MyProj.cpp` που δημιουργούνται αυτόματα, τα ονόματα των οποίων προέρχονται προφανώς από το όνομα του project. Αρχικά περιέχουν κάποια παραδείγματα κώδικα. Αν εξαιρέσουμε τις εντολές για τον προεπεξεργαστή, όλα τα υπόλοιπα μπορούμε να τα σβήσουμε στα αρχεία αυτά. Στο `MyProj.h` θα βάλουμε το πρωτότυπο της συνάρτησης `orprog()` και στο `MyProj.cpp` τη δήλωσή της. Με μια διαφορά: στη Visual C++ .NET πριν από τον τύπο μεταβλητής που επιστρέφει η συνάρτηση `orprog()` πρέπει να βάλουμε τη σταθερά `MYPROJ_API` — να τονίσουμε ότι το όνομα της σταθεράς σχετίζεται και εδώ με το όνομα του project. Πιο συγκεκριμένα στο αρχείο `MyProj.h` βάζουμε (μαζί με τη δήλωση της δομής `struct room_t`):

```
extern "C"  {
    MYPROJ_API  char* orprog(struct room_t *room, int nroom);
}
```

Ενώ στο `MyProj.cpp`:

```
MYPROJ_API  char* orprog(struct room_t *room, int nroom)
{
 ⋮
}
```

Κάθε επιπλέον αρχείο `*.cpp` που τυχόν επιθυμούμε να χρησιμοποιήσουμε, πέρα από τα ήδη υπάρχοντα, θα πρέπει να κάνει `#include` το `stdafx.h`.

Μεταγλώττιση με gcc

Ένας άλλος τρόπος για να μεταγλωττίσουμε κώδικα C++ είναι να χρησιμοποιήσουμε τον g++ ο οποίος περιλαμβάνεται στο πακέτο του gcc. Τέτοιο πακέτο

για Windows είναι το MinGW.[3] Το Cygwin[4] είναι μια άλλη, ευρύτερη ομάδα προγραμμάτων. Για να παραχθεί το DLL δημιουργούμε και μεταγλωττίζουμε ένα αρχείο C++ σύμφωνα με αυτά που ειπώθηκαν στην παράγραφο «Ο κώδικας C/C++». Ο g++ όταν κληθεί σαν linker πρέπει να πάρει την παράμετρο `-shared` και να ορίσουμε σαν αρχείο εξόδου (παράμετρος `-o`) κάποιο αρχείο `*.dll`. Εφόσον το DLL μεταγλωττίστηκε μέσω Cygwin, για να λειτουργήσει χρειάζεται και το `cygwin1.dll`, αρχείο το οποίο μπορούμε να αντιγράψουμε από τον κατάλογο του Cygwin, στον κατάλογο του DLL που φτιάξαμε.

Η βιβλιοθήκη της μηχανής επίλυσης μεταγλωττίζεται μέσω του Cygwin. Ο λόγος για τον οποίον δεν χρησιμοποιήθηκε η Visual C++ .NET είναι ότι δεν είναι πλήρως συμβατή με τον g++, πάνω στον οποίο στηρίχθηκε η ανάπτυξή της βιβλιοθήκης. Η τελευταία χρησιμοποιεί και κάποιες δυνατότητες των επεκτάσεων STL, οι οποίες δεν είναι καταγεγραμμένες σε κάποιο πρότυπο και έτσι η μεταφορά του κώδικα σε άλλη πλατφόρμα έγινε ακόμα πιο δυσχερής.

Γ′.1.2 Κλήση συνάρτησης ενός DLL στη Visual Basic .NET

Η Visual Basic .NET παρέχει τη δυνατότητα κλήσης συναρτήσεων ενός DLL. Γενικότερα οι .NET εφαρμογές μπορούν να επικοινωνούν εύκολα μεταξύ τους, μέσα από τη χρήση "managed" κώδικα. Υπάρχει όμως η δυνατότητα ανταλλαγής δεδομένων και με εφαρμογές εκτός .NET, οι οποίες χρησιμοποιούν "unmanaged" κώδικα για να επικοινωνούν, σύμφωνα με την ορολογία της Microsoft. Ο κώδικας αυτός περιέχει τους βασικούς τύπους μεταβλητών, όπως αυτοί της C. Για να αντιμετωπιστούν προβλήματα συμβατότητας έχει δημιουργηθεί η βιβλιοθήκη Marshal, που κάνει προσαρμογή τύπων (Marshaling).

Αν θέλουμε να καλέσουμε μία συνάρτηση ενός DLL, πρώτα από όλα πρέπει να γνωρίζουμε το πρωτότυπό της, το οποίο τεκμηριώνεται συνήθως από τον κατασκευαστή του DLL. Επίσης πρέπει να ξέρουμε τη θέση του DLL στο δίσκο. Αν τα γνωρίζουμε όλα αυτά είμαστε έτοιμοι να ενσωματώσουμε τη συνάρτηση στον κώδικα Visual Basic .NET μέσα από τη δήλωση `Declare`, όπως στον παρακάτω κώδικα:[5]

```
Imports System.Runtime.InteropServices
```

[3] http://www.mingw.org/

[4] http://www.cygwin.com/

[5] Για να «σπάσει» μια εντολή σε δύο γραμμές, χρησιμοποιείται ο χαρακτήρας "_" στο τέλος της πρώτης γραμμής.

```
<StructLayout(LayoutKind.Sequential)> _
Private Structure Room_t
    Public Unavail As IntPtr
    Public UnavailLength As Integer
End Structure

Private Declare Auto _
Function orprog Lib "naxos.dll" _
    (<[In](), Out()> ByVal Room() As Room_t, _
     ByVal RoomLength As Integer) _
    As <MarshalAsAttribute(UnmanagedType.LPStr)> String
```

Συγκρίνοντας τον παραπάνω κώδικα με αυτόν της προηγούμενης παραγράφου, μπορούμε εύκολα να εντοπίσουμε ομοιότητες. Αλλά ας πάρουμε μία μία της ενότητες του κώδικα.

Στην πρώτη γραμμή δηλώνουμε ότι θα χρησιμοποιήσουμε τις κλάσεις που κάνουν το Marshaling. Έπειτα ορίζουμε τη δομή της αίθουσας `Structure Room_t` με τα μέλη της σε πλήρη αντιστοιχία με αυτά της `struct room_t` του C++ κώδικα. Η δομή ορίζεται όπως συνήθως, με τη διαφορά ότι έχουμε το πρόθεμα `<StructLayout(LayoutKind.Sequential)>`. Αυτό μπαίνει για να ενημερώσουμε τον μεταγλωττιστή ότι θα χρησιμοποιήσουμε έναν *πίνακα* από δομές `Structure Room_t` ο οποίος πρέπει να έχει τα στοιχεία του τοποθετημένα σειριακά στη μνήμη, όπως γίνεται άλλωστε με τους πίνακες της C.

Τελευταία έχουμε τη δήλωση `Declare` στην οποία υπάρχει το όνομα της συνάρτησης `orprog()` που περιέχεται στο DLL. Μετά τη λέξη-κλειδί `Lib` υπάρχει η διαδρομή του καταλόγου και το όνομα του DLL. Στο παράδειγμα βέβαια δεν υπάρχει διαδρομή αφού το `naxos.dll` βρίσκεται στον ίδιο κατάλογο με αυτόν της εφαρμογής που το καλεί. Όσον αφορά το πρώτο όρισμα ο προσδιορισμός `<[In](), Out()>` προηγείται της δήλωσης του πίνακα με τις δομές. Με αυτόν δηλώνουμε ότι θα εισαγάγουμε ή/και θα πάρουμε δεδομένα μέσω αυτού του ορίσματος. Η συνάρτηση `orprog()` επιστρέφει μία συμβολοσειρά. Αυτή μετατρέπεται σε συμβολοσειρά τύπου `String` της Visual Basic .NET αν βάλουμε το πρόθεμα `<MarshalAsAttribute(UnmanagedType.LPStr)>`.[6]

[6]Μπορεί κανείς να κατανοήσει την αναγκαιότητα αυτής της προσαρμογής τύπου, αν σκεφτεί ότι στη C κάθε χαρακτήρας είναι ένα byte, ενώ στη Visual Basic .NET χρησιμοποιείται το σύστημα Unicode στο οποίο ένας χαρακτήρας μπορεί να αποθηκεύεται μέχρι και σε 4 byte. Ένας απλούστερος τρόπος για να πετύχουμε αυτήν την προσαρμογή τύπου χωρίς να

Στη δομή `Structure Room_t` υπάρχει ο τύπος `IntPtr` που δεν είναι και ο πλέον συνηθισμένος στη Visual Basic. Το όνομά του προέρχεται από το Integer Pointer και αντιστοιχεί στο `int*` της C.[7] Η διαφορά αυτού τού τύπου από τους υπόλοιπους είναι ότι χρησιμοποιείται όπως οι δείκτες στην C. Δεν υπάρχει για αυτόν Garbage Collection.[8]

Η δέσμευση και η απελευθέρωση της μνήμης που δείχνει ένας `IntPtr` γίνεται με παρόμοιο τρόπο με αυτόν της C. Στις `malloc()` και `free()` αντιστοιχούν οι μέθοδοι `AllocHGlobal()` και `FreeHGlobal()`, οι οποίες ανήκουν στην κλάση `Marshal`. Στον παρακάτω κώδικα γίνεται αποστολή δεδομένων σε ένα DLL και ανάγνωση των δεδομένων που επεστράφησαν από αυτό. Θα μπορούσε κανείς να πει ότι αποτελεί ένα παράδειγμα μεταφοράς από τον κόσμο του managed κώδικα, σε αυτόν του unmanaged και τούμπαλιν.[9]

```
Dim MyVbArray(ArbitrarySize - 1) As Integer
  ' Ανάθεση τιμών στα στοιχεία του MyVbArray...

Dim MyDllArray As IntPtr
MyDllArray = Marshal.AllocHGlobal( _
               Marshal.SizeOf(MyVbArray(0)) * MyVbArray.Length)
Marshal.Copy(MyVbArray, 0, MyDllArray, MyVbArray.Length)

  ' Το DLL καλείται να επεξεργαστεί το MyDllArray...

Marshal.Copy(MyDllArray, MyVbArray, 0, MyVbArray.Length)
Marshal.FreeHGlobal(MyDllArray)
```

Αρχικά ορίζουμε έναν πίνακα ακεραίων `MyVbArray` μεγέθους `Arbitrary-Size`. Μετά από αυτή τη γραμμή μπορούμε πλέον να περάσουμε όποια δεδομένα θέλουμε στον πίνακα αυτό. Για να τα μεταβιβάσουμε στο DLL ορίζουμε έναν έναν δείκτη σε πίνακα ακεραίων `MyDllArray`. Ο `IntPtr` τύπος του αντιστοιχεί στο `int*` του unmanaged κώδικα.

βάλουμε το Marshal πρόθεμα, είναι να αλλάξουμε στη δήλωση `Declare` το `Auto` σε `Ansi`.

[7] Υπάρχει και ο τύπος `UIntPtr` που αντιστοιχεί στο `unsigned*` της C.

[8] Το Garbage Collection είναι μία μέθοδος διαχείρισης μνήμης στο οποίο η αποδέσμευση πόρων μνήμης γίνεται αυτόματα, χωρίς να χρειάζεται ο προγραμματιστής να προνοήσει για αυτήν. Εκτός από τη Visual Basic .NET, Garbage Collection χρησιμοποιούν και άλλες διαδεδομένες γλώσσες όπως η Java.

[9] Στη Visual Basic τα σχόλια αρχίζουν με την απόστροφο "'" και τελειώνουν στο τέλος της γραμμής.

Με την `AllocHGlobal()` δεσμεύουμε μνήμη για το `MyDllArray`. Επειδή η `SizeOf(MyVbArray(0))` ισούται με το μέγεθος ενός ακεραίου, τελικά παίρνουμε όσο αποθηκευτικό χώρο απαιτείται για να αντιγραφεί το `MyVbArray`. Αξίζει να σημειωθεί ότι η `SizeOf()` δέχεται σαν όρισμα ένα *στιγμιότυπο* ενός τύπου και όχι τον ίδιο τον τύπο.

Η πρώτη `Copy()` αντιγράφει το `MyVbArray` στο `MyDllArray`. Η δεύτερη `Copy()` κάνει την αντίστροφη διαδικασία. Είναι εμφανές ότι η `Copy()` χρησιμοποιεί υπερφόρτωση (overloading). Το "`0`" αντιπροσωπεύει τη θέση εκκίνησης και το "`MyVbArray.Length`" τον αριθμό των στοιχείων που θα αντιγραφούν. Ανάμεσα στις δύο `Copy()` μπορούμε να καλέσουμε μία συνάρτηση του DLL. Αν υπάρχει στα ορίσματά της το `MyDllArray`, το DLL θα μπορεί να διαβάσει και να εγγράψει εμμέσως τα περιεχόμενα του `MyVbArray`.

Γ′.2 Υλοποίηση ενός νήματος σε Visual Basic .NET

Η Visual Basic .NET έχει ενσωματώσει στις κλάσεις της τα νήματα. Στη συνέχεια θα παρουσιαστεί ένα παράδειγμα, εμπνευσμένο από τον κώδικα της εφαρμογής της πτυχιακής, σχετικά με το πώς κάποιος μπορεί να κατασκευάσει ένα πολυνηματικό περιβάλλον.

Η ιδέα για τη δημιουργία νήματος προτάθηκε όταν εμφανίστηκε η συμπεριφορά της εφαρμογής να «παγώνει» περιμένοντας τη συνάρτηση υπολογισμού να καταλήξει σε κάποια λύση. Ο χρόνος που η μηχανή εκτέλεσης εξέταζε εκατοντάδες λύσεις του προβλήματος περνούσε ανεκμετάλλευτος, ενώ μέσα σε αυτόν ο χρήστης θα μπορούσε να επεξεργάζεται τα δεδομένα εισόδου του ίδιου ή ενός άλλου προβλήματος. Ούτως ή άλλως η μηχανή κατάρτισης ωρολογίου προγράμματος έχει ήδη πάρει τα δεδομένα του προβλήματος που επιλύει. Αφού επιλέξαμε να τρέξουμε τη μηχανή επίλυσης σε ένα νήμα, προέκυψε η ανάγκη ανατροφοδότησης του κύριου νήματος με πληροφορίες σχετικά με τη μηχανή επίλυσης: σε ποιο στάδιο βρίσκεται η αναζήτηση της λύσης, αν το νήμα έχει επιστρέψει, ή αν υπήρξε κάποιο σφάλμα.

Μέσω της επεξήγησης του κώδικα θα ακουμπήσουμε τα τεχνικά θέματα.

```
Private ThreadMy As New Threading.Thread(AddressOf SolverSub)

Private Sub SolverStart()
    ' Αντιγραφή προβλήματος προς επίλυση...
```

```
    ThreadMy.Start()
    TimerMy.Start()
End Sub

Private Sub TimerMy_Tick(ByVal sender As System.Object, _
                         ByVal e As System.EventArgs) _
                        Handles TimerMy.Tick
    ProgressBarMy.Value = ProgressOfSolution
    If ThreadMy.Join(1) Then
        TimerMy.Stop()
        ' Συνδυασμός λύσης και αντίγραφου προβλήματος...
    End If
End Sub
```

Έστω ότι έχουμε εισάγει όλα τα δεδομένα για ένα πρόβλημα κατάρτισης ωρολογίου προγράμματος. Για να ξεκινήσει η επίλυσή του, καλείται η `Solver-Start()` η οποία αρχικά αντιγράφει κάπου το τρέχον πρόβλημα έτσι ώστε όταν αυτό επιλυθεί, να υπάρχουν αποθηκευμένες όλες οι πληροφορίες του. Υπενθυμίζουμε ότι ο χρήστης θα έχει τη δυνατότητα να αλλάξει το τρέχον πρόβλημα και έτσι πρέπει να το κάνουμε "backup."

Η μεταβλητή νήματος `ThreadMy` είναι καθολική για να έχει εμβέλεια και στις δύο συναρτήσεις. Όταν ξεκινήσει το νήμα θα καλέσει τη συνάρτηση `SolverSub`, η οποία δεν έχει ορίσματα και δεν επιστρέφει τίποτα. Σύμφωνα με την ορολογία της Visual Basic είναι υπορουτίνα (`Sub`). Με τη μέθοδο `Start()` το νήμα εκκινεί.

Ο `TimerMy` είναι ένας control[10] της Visual Basic που προκαλεί ένα event ανά τακτά χρονικά διαστήματα. Η συχνότητα του event σε milliseconds καθορίζεται από τη μεταβλητή-μέλος `TimerMy.Interval`. Κάθε φορά που προκαλείται το event `TimerMy.Tick` καλείται η συνάρτηση-χειριστής του `TimerMy_Tick()`. Ο προγραμματιστής πρέπει να προσέξει ότι το event δεν έχει εγγυημένα σταθερή συχνότητα και δεν εμφανίζεται όταν η εφαρμογή είναι απασχολημένη.

Έτσι μετά τη `ThreadMy.Start()` το κύριο νήμα εκτελεί την `TimerMy-.Start()`. Στο εξής θα έχουμε περιοδική κλήση της `TimerMy_Tick()` και μέσω αυτής θα μπορούμε να βλέπουμε την πορεία του νήματος της μηχανής επίλυσης. Μπαίνοντας στο σώμα της συγκεκριμένης συνάρτησης βλέπουμε ότι η πρώτη εντολή της ενημερώνει έναν control `ProgressBar`, τον `ProgressBarMy`, με την

[10] Οι control είναι στοιχεία των φορμών, ορατά και αόρατα. Π.χ. `TextBox`, `Button`, `CheckBox`, `ImageList` (αόρατο) κ.ά.

τιμή της μεταβλητής `ProgressOfSolution` που έχει περαστεί με αναφορά στη συνάρτηση υπολογισμού.

Η `ThreadMy.Join(t As Integer)` περιμένει `t` milliseconds και αν μέχρι τότε το νήμα `ThreadMy` δεν έχει τερματίσει, επιστρέφει `False`. Σε άλλη περίπτωση επιστρέφει `True` και η συνθήκη του `If` στην `TimerMy_Tick()` γίνεται αληθής. Τότε καλείται η `TimerMy.Stop()`. Διαβάζουμε τη λύση που επεστράφη από τη μηχανή επίλυσης και τη συνδυάζουμε με τις υπόλοιπες πληροφορίες (π.χ. ονόματα αιθουσών/καθηγητών) που υπάρχουν στο αντίγραφο του προβλήματος που φτιάχτηκε στην `SolverStart()`.

Σε κάθε περίπτωση μπορούμε να σταματήσουμε την εκτέλεση του νήματος καλώντας τη `ThreadMy.Abort()`.

Βιβλιογραφία

[1] T. Cooper and J. Kingston, "The complexity of timetable construction problems," in *Proc. 1st Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT'95)*. Springer, 1995, pp. 283–295.

[2] D. de Werra, "The combinatorics of timetabling," *European Journal Of Operational Research*, vol. 96, no. 3, pp. 504–513, 1997.

[3] E. Burke and S. Petrovic, "Recent research directions in automated timetabling," *European Journal Of Operational Research*, vol. 140, no. 2, pp. 266–280, 2002.

[4] S. Elmohamed, P. Coddington, and G. Fox, "A comparison of annealing techniques for academic course scheduling," in *Proc. 2nd Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT'97)*, vol. 1408. Springer, 1997, pp. 92–112.

[5] J. Boufflet and S. Nègre, "Three methods used to solve an examination timetable problem," in *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT'95)*, vol. 1153. Springer, 1995, pp. 327–344.

[6] R. Weare, E. Burke, and D. Elliman, "A hybrid genetic algorithm for highly constrained timetabling problems," in *Proc. 6th Int. Conf. on Genetic Algorithms*. LJ Eshelman, 1995, pp. 605–610.

[7] C. Gueret, N. Jussien, P. Boizumault, and C. Prins, "Building university timetables using constraint logic programming," in *Proc. 1st Int. Conf.*

on the Practice and Theory of Automated Timetabling (PATAT'95), vol. 1153. Springer, 1995, pp. 130–145.

[8] H. Frangouli, V. Harmandas, and P. Stamatopoulos, "UTSE: Construction of optimum timetables for university courses – A CLP based approach," in *Proc. 3rd Int. Conf. on the Practical Applications of Prolog (PAP'95)*, 1995, pp. 225–243.

[9] P. Stamatopoulos, E. Viglas, and S. Karaboyas, "Nearly optimum timetable construction through CLP and intelligent search," *Int. Journal on Artificial Intelligence Tools*, vol. 7, no. 4, pp. 415–442, 1998.

[10] C. Valouxis and E. Housos, "Constraint programming approach for school timetabling," *Computers and Operations Research*, vol. 30, no. 10, pp. 1555–1572, 2003.

[11] Ι. Βελεσιώτης, Κ. Ζερβουδάκης και Π. Σταματόπουλος, "ORPROG: Διαδικτυακό σύστημα κατάρτισης ωρολογίου προγράμματος μαθημάτων," διαθέσιμο στη διεύθυνση http://www.di.uoa.gr/~orprog/.

[12] *ILOG Solver 4.4: Reference manual*, ILOG S.A., 1999.

[13] *ILOG Solver 4.4: User's manual*, ILOG S.A., 1999.

[14] Κ. Ζερβουδάκης, "Κατάρτιση ωρολογίου προγράμματος σε αντικειμενοστραφές περιβάλλον με περιορισμούς," Πτυχιακή Εργασία, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών, 1999.

[15] Ι. Βλαχάβας, Π. Κεφαλάς, Ν. Βασιλειάδης, Ι. Ρεφανίδης, Φ. Κόκκορας και Η. Σακελλαρίου, *Τεχνητή Νοημοσύνη*. Εκδόσεις Γαρταγάνη, 2002, pp. 68–82.

[16] R. Barták, "Online guide to constraint programming," διαθέσιμο στη διεύθυνση http://kti.ms.mff.cuni.cz/~bartak/constraints/.

[17] K. Apt, *Principles of Constraint Programming*. Cambridge University Press, 2003.

[18] R. Dechter, *Constraint Processing*. Morgan Kaufmann, 2003.

[19] V. Kumar, "Algorithms for constraint-satisfaction problems: A survey," *AI Magazine*, vol. 13, no. 1, pp. 32–44, 1992.

[20] E. Tsang, "A glimpse of constraint satisfaction," *Artificial Intelligence Review*, vol. 13, no. 3, pp. 215–227, 1999.

[21] C. Unsworth, "First year report," University of Glasgow, Tech. Rep., 2005, p. 7.

[22] C. Bessière, J. Régin, R. Yap, and Y. Zhang, "An optimal coarse-grained arc consistency algorithm," *Artificial Intelligence*, vol. 165, no. 2, pp. 165–185, 2005.

[23] P. van Hentenryck, Y. Deville, and C. Teng, "A generic arc-consistency algorithm and its specializations," *Artificial Intelligence*, vol. 57, no. 2-3, pp. 291–321, 1992.

[24] J. McGregor, "Relational consistency algorithms and their application in finding subgraph and graph isomorphism," *Information Science*, vol. 19, pp. 229–250, 1979.

[25] A. Chmeiss and P. Jégou, "Efficient path-consistency propagation," *Int. Journal on Artificial Intelligence Tools*, vol. 7, no. 2, pp. 121–142, 1998.

[26] Η. Κουτσουπιάς, *Μαθηματικά της πληροφορικής επιστήμης*. Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών, 2005, p. 41.

[27] B. Kernighan and R. D.M., *Η γλώσσα προγραμματισμού C*. Prentice Hall (Ελληνική μετάφραση, εκδόσεις Κλειδάριθμος), 1988, p. 67.

[28] D. Sabin and E. Freuder, "Understanding and improving the MAC algorithm," in *Proc. 3rd Int. Conf. on Principles and Practices of Constraint Programming (CP'97)*, 1997, pp. 167–181.

[29] ——, "Contradicting conventional wisdom in constraint satisfaction," in *Proc. 2nd Int. Workshop on Principles and Practice of Constraint Programming (PPCP'94)*, 1994, pp. 125–129.

[30] J. Ferland, "Generalized assignment-type problems: A powerful modeling scheme," in *Proc. 2nd Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT'97)*. Springer, 1997, pp. 53–77.

[31] V. Bardadym, "Computer-aided school and university timetabling: The new wave," in *Proc. 1st Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT'95)*. Springer, 1995, pp. 22–45.

[32] K. Zervoudakis and P. Stamatopoulos, "A generic object-oriented constraint based model for university course timetabling," in *Proc. 3rd Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT 2000)*. Springer, 2000, pp. 128–147.

[33] J. Kleinberg and É. Tardos, *Algorithm Design*. Pearson/Addison-Wesley, 2005, pp. 637–643.

[34] W. Harvey and M. Ginsberg, "Limited discrepancy search," in *Proc. 14th Int. Joint Conf. on Artificial Intelligence (IJCAI-95)*, vol. 1, 1995, pp. 607–615.

[35] D. Diaz, *GNU Prolog: A native Prolog compiler with constraint solving over finite domains*, 1st ed., 2002, p. 172.

[36] B. Eckel and C. Allison, *Thinking In C++ Vol. 2: Practical Programming*, 2nd ed. Prentice Hall, 2003.

[37] Π. Σταματόπουλος, *Σημειώσεις λογικού προγραμματισμού*. Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών, 2005, pp. 68–71, 177–180.

[38] *MSDN Library – Visual Studio .NET 2003*, Microsoft Corporation, διαθέσιμο στη διεύθυνση http://msdn.microsoft.com/.

[39] R. Bowman, *Visual Basic .NET*. Hungry Minds, 2002.

[40] E. Petroutsos, *Mastering Visual Basic .NET*. Sybex, 2002.

[41] Π. Σταματόπουλος, *Λογικός και συναρτησιακός προγραμματισμός*. Ελληνικό Ανοικτό Πανεπιστήμιο, 2000, pp. 25–26, 54–55, 60, υπό έκδοση.

[42] D. Chapman, *Teach yourself Visual C++ 6 in 21 days*. Sams, 1998, pp. 405–426.

[43] K. Gregory, *Microsoft Visual C++ .NET 2003 Kick Start*. Sams, 2003, ch. 7: δύο πρώτες ενότητες.

Ευρετήριο