Εγχειρίδιο χρήσης της Amorgos

Φοίβος Θεοχάρης
f.theoharis@di.uoa.gr

19 Μαΐου 2008

Το εγχειρίδιο αυτό απευθύνεται στον προγραμματιστή εφαρμογών (στο εξής *χρήστη*) που επιθυμεί να χρησιμοποιήσει την Amorgos στα προγράμματά του. Βασική προυπόθεση είναι η γνώση της γλώσσας προγραμματισμού C++, μιας και η Amorgos είναι μία βιβλιοθήκη για τη γλώσσα αυτή. Η Amorgos είναι επέκταση του Naxos, οπότε κάποια γνώση του τελευταίου προαπαιτείται επίσης. Μία εισαγωγή στον προγραμματισμό του Naxos γίνεται στη συνέχεια.

1 Προγραμματίζοντας με τον επιλυτή Naxos

1.1 Το πρόβλημα των N βασιλισσών

Ας δούμε πώς μπορούμε, ως προγραμματιστές, να χρησιμοποιήσουμε τον επιλυτή Naxos για να λύσουμε ένα πρόβλημα, και συγκεκριμένα το πρόβλημα των N βασιλισσών. Στο πρόβλημα αυτό στόχος μας είναι να τοποθετήσουμε N βασίλισσες σε μία σκακιέρα διαστάσεων $N \times N$ έτσι ώστε καμία βασίλισσα να μην απειλεί καμία άλλη. Για όσους δεν είναι εξοικειωμένοι με το σκάκι ας διευκρινίσουμε ότι δύο βασίλισσες απειλούν η μία την άλλη αν βρίσκονται στην ίδια γραμμή, στήλη ή διαγώνιο της σκακιέρας. Έτσι, αποκλείεται σε κάποια λύση δύο βασίλισσες να βρίσκονται στην ίδια στήλη, οπότε σε κάθε στήλη θα υπάρχει μία βασίλισσα. Μένει να βρούμε σε ποια γραμμή θα βρίσκεται η βασίλισσα αυτή.

Θα έχουμε N μεταβλητές, τις X_i για $0 \leq i \leq N-1$. Κάθε μεταβλητή X_i θα παίρνει τιμές από το 0 ως το $N-1$. Δηλαδή αν η βασίλισσα της στήλης i βρίσκεται στη γραμμή j, τότε η X_i θα έχει την τιμή j.

Αφού εξασφαλίσαμε με τη μοντελοποίηση που κάναμε ότι δύο βασίλισσες δε θα βρίσκονται στην ίδια στήλη, μένει να καθορίσουμε και τους υπόλοιπους

περιορισμούς. Ο περιορισμός

$$\forall i \neq j \quad X_i \neq X_j$$

μας λέει ότι δύο βασίλισσες δεν μπορεί να βρίσκονται στην ίδια γραμμή, ενώ για τις διαγωνίους υπάρχει ο περιορισμός

$$\forall i \neq j \quad X_i + i \neq X_j + j \quad \text{και} \quad X_i - i \neq X_j - j$$

1.2 Επίλυση του προβλήματος των N βασιλισσών με τον Naxos

Ο κώδικας που λύνει το παραπάνω πρόβλημα είναι ο εξής:

```
#include <naxos.h>

#include <iostream>
#include <cstdlib>

using namespace std;
using namespace naxos;


int  main (int argc, char *argv[])
{
    try  {
      int  N = (argc > 1) ? atoi(argv[1]) : 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";

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

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

Ας δούμε τι κάνει ο παραπάνω κώδικας. Πρώτα τα διαδικαστικά: Για να χρησιμοποιήσουμε τον Naxos χρειάζονται οι δύο εντολές:

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

στην αρχή του αρχείου. Επίσης, για τη διαχείρηση των εξαιρέσεων, οφείλουμε να γράφουμε τον κώδικά μας μέσα σε ένα try-catch block:

```
try  {

    // code here...

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

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

Ας προχωρήσουμε τώρα στο ενδιαφέρον κομμάτι:

```
int  N = (argc > 1) ? atoi(argv[1]) : 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";
```

Όπως παρατηρείτε, στην αρχή δηλώνουμε έναν διαχειριστή προβλήματος (problem manager), ο οποίος θα περιέχει και θα διαχειρίζεται τις μεταβλητές και τους περιορισμούς του προβλήματος. Ο πίνακας `Var` περιέχει τις N μεταβλητές που αντιστοιχούν στις N στήλες της σκακέρας. Οι πίνακες `VarPlus` και `VarMinus` είναι βοηθητικοί και συνδέονται με τον πίνακα `Var` με περιορισμούς κατά τη δημιουργία τους.

Οι υπόλοιποι περιορισμοί είναι τύπου `AllDifferent`. Ο πρώτος από αυτούς δηλώνει ότι όλα τα στοιχεία του `Var` θα πρέπει να είναι διαφορετικά (δηλαδή ότι όλες οι βασίλισσες θα πρέπει να είναι σε διαφορετικές σειρές της σκακέρας. Οι υπόλοιποι κάνουν το αντίστοιχο για τις διαγωνίους.

Αφού δηλώσαμε τις μεταβλητές και τους περιορισμούς είμαστε έτοιμοι να λύσουμε το πρόβλημα. Η εντολή

```
pm.addGoal( new NsgLabeling(Var) );
```

επιλέγει για μέθοδο αναζήτησης την ενσωματωμένη στον Naxos DFS. Κάθε μέθοδος αναζήτησης είναι ένας στόχος. Ορίζεται ως κλάση που κληρονομεί από την `NsGoal`. Η συνάρτηση

```
pm.nextSolution()
```

ξεκινάει την αναζήτηση. Επιστρέφει `true` αν μία λύση του προβλήματος έχει βρεθεί ή `false` αν η αναζήτηση έχει τελειώσει χωρίς να βρεθεί λύση. Μπορούμε να διαβάσουμε τη λύση από τις τιμές των περιορισμένων μεταβλητών. Αν καλέσουμε επανειλημμένα την παραπάνω συνάρτηση η αναζήτηση θα συνεχίσει από το σημείο στο οποίο είχε σταματήσει. Έτσι μπορούμε να βρούμε όλες τις λύσεις του προβλήματος.

Για περισσότερες λεπτομέρειες σχετικά με τον τρόπο χρήσης του Naxos συμβουλευθείτε το εγχειρίδιο χρήσης του στο διαδίκτυο [1].

2 Χρησιμοποιώντας την Amorgos

Έστω ότι θέλουμε να λύσουμε το παραπάνω πρόβλημα με την έκδοση της DFS που υπάρχει στην Amorgos. Μόνο δύο αλλαγές χρειάζεται να γίνουν. Πρώτα πρέπει να συμπεριληφθεί η παρακάτω εντολή στην αρχή του αρχείου:

```
#include <amorgos.h>
```

και τέλος αρκεί να αντικατασταθεί η γραμμή

```
pm.addGoal( new NsgLabeling(Var) );
```

με την παρακάτω:

```
pm.addGoal( new goalDfsLabeling(Var) );
```

2.1 Ευριστικοί μηχανισμοί

Μία ευκολία που παρέχει η Amorgos είναι η παραμετρικότητα των μεθόδων αναζήτησης ως προς τους ευριστικούς μηχανισμούς. Έστω ότι θέλουμε να χρησιμοποιήσουμε το ευριστικό τυχαίας επιλογής μεταβλητής στην DFS μας. Θα χρησιμοποιήσουμε τον υπερφορτωμένο constructor:

```
pm.addGoal( new goalDfsLabeling(Var, new VarHeurRand) );
```

Κάθε ευριστικό είναι ορισμένο ως μία κλάση. Οι κλάσεις για τα ευριστικά επιλογής μεταβλητής κληρονομούν από την κλάση `VariableHeuristic` ενώ οι κλάσεις επιλογής τιμής από την κλάση `ValueHeuristic`. Τα διαθέσιμα ευριστικά είναι τα εξής:

- `VarHeurFirst`: Το 'ευριστικό' επιλογής της πρώτης μεταβλητής.
- `VarHeurRand`: Το ευριστικό επιλογής τυχαίας μεταβλητής.
- `VarHeurMRV`: Ελάχιστες εναπομένουσες τιμές.
- `ValHeurFirst`: Το 'ευριστικό' επιλογής της ελάχιστης τιμής.
- `ValHeurRand`: Το ευριστικό επιλογής τυχαίας τιμής.

Ο χρήστης που επιθυμεί να δημιουργήσει δικό του ευριστικό πρέπει να ορίσει μία κλάση που κληρονομεί από την `VariableHeuristic`, αν πρόκειται για ευριστικό επιλογής μεταβλητής, ή από την `ValueHeuristic`, αν πρόκειται για ευριστικό επιλογής τιμής. Στη συνέχεια οφείλει να επαναορίσει τη μέθοδο `select` για την κλάση αυτή.

Στα ευριστικά επιλογής μεταβλητής η `select` παίρνει ένα όρισμα τύπου αναφοράς σε `NsIntVarArray`, δηλαδή έναν πίνακα με τις μεταβλητές ανάμεσα στις οποίες πρέπει να γίνει η επιλογή. Επιστρέφει έναν ακέραιο, τη θέση δηλαδή στον πίνακα στην οποία βρίσκεται η επιλεγμένη μεταβλητή. Το παρακάτω παράδειγμα είναι η υλοποίηση του ευριστικού *ελάχιστες εναπομένουσες τιμές*.

```
int VarHeurMRV::select(const NsIntVarArray& Vars)
{
    int  index = -1;
    NsUInt  minDom = NsUPLUS_INF;
    for (NsIndex i=0; i < Vars.size(); ++i){
        if ( !Vars[i].isBound()   &&   Vars[i].size() < minDom ){
            minDom = Vars[i].size();
            index = i;
        }
    }
    return  index;
}
```

Στα ευριστικά επιλογής τιμής το όρισμα της `select` είναι τύπου αναφοράς σε `NsIntVar`, δηλαδή η μεταβλητή για την οποία πρέπει να επιλεγεί η τιμή. Επιστρέφεται ακέραιος τύπου `NsInt`, δηλαδή η επιλεγμένη τιμή. Στη συνέχεια φαίνεται η υλοποίηση του 'ευριστικού' επιλογής της ελάχιστης τιμής.

```
NsInt ValHeurFirst::select(const NsIntVar& V)
{
    return  V.min();
}
```

2.2 Μέθοδοι αναζήτησης

Στη συνέχεια θα δούμε τα προτότυπα των constructors για τις μεθόδους αναζήτησης που είναι διαθέσιμες στην Amorgos.

2.2.1 Αναζήτηση πρώτα κατά βάθος

Η `goalDfsLabeling` αναθέτει τιμές σε έναν πίνακα χρησιμοποιώντας τη μέθοδο DFS. Η `goalDfsInDomain` αναθέτει μία τιμή σε μία μεταβλητή. Για την πρώτη ο χρήστης μπορεί να καθορίσει ο ίδιος τα ευριστικά με βάση τα οποία θα γίνεται η επιλογή μεταβλητής και τιμής, ενώ για τη δεύτερη μπορεί να καθορίσει το ευριστικό επιλογής τιμής.

```
goalDfsLabeling(NsIntVarArray& Vars,
                VariableHeuristic *varHeuristic = new VarHeurMRV,
                ValueHeuristic *valHeuristic = new ValHeurFirst)

goalDfsInDomain(NsIntVar& Var,
                ValueHeuristic *valHeuristic = new ValHeurFirst)
```

2.2.2 Αναζήτηση με περιορισμένη ασυμφωνία

Ο constructor της μεθόδου παίρνει τα ακόλουθα επιπλέον προαιρετικά ορίσματα:

- `step`: Κατά πόσο αυξάνεται η ασυμφωνία σε κάθε επανάληψη της LDS. Προεπιλεγμένη τιμή: 1
- `lookahead`: Όταν η αναζήτηση φτάνει σε απόσταση `lookahead` από τα φύλλα του δένδρου, εξερευνά όλες τις εναλλακτικές επιλογές από εκεί και κάτω χρησιμοποιώντας DFS. Προεπιλεγμένη τιμή: 0
- `minDiscrepancy`: Κατά την αναζήτηση δε θα εξεταστούν μονοπάτια με λιγότερη ασυμφωνία από `minDiscrepancy`. Προεπιλεγμένη τιμή: 0
- `maxDiscrepancy`: Κατά την αναζήτηση δε θα εξεταστούν μονοπάτια με περισσότερη ασυμφωνία από `maxDiscrepancy`. Αν γίνει ίσο με 0, τότε δε θα υπάρχει άνω φράγμα στην ασυμφωνία. Προεπιλεγμένη τιμή: 0

```
goalLds(NsIntVarArray& Vars,
        int step = 1,
        int lookahead = 0,
        int minDiscrepancy = 0,
        int maxDiscrepancy = 0,
        VariableHeuristic *varHeuristic = new VarHeurMRV,
        ValueHeuristic *valHeuristic = new ValHeurFirst)
```

2.2.3 Αναζήτηση διαμοιρασμού πίστωσης

Για να χρησιμοποιηθεί αυτή η μέθοδος αρκεί να καθοριστεί το `credit`, δηλαδή η πίστωση που θα δοθεί στη ρίζα του δένδρου αναζήτησης.

```
goalCredit(NsIntVarArray& Vars,
           int credit,
           VariableHeuristic *varHeuristic = new VarHeurMRV,
           ValueHeuristic *valHeuristic = new ValHeurFirst)
```

2.2.4 Αναζήτηση φραγμένου βάθους με οπισθοδρόμηση

Η `goalDbsStepping` είναι η πλήρης έκδοση της μεθόδου και αποτελείται από n επαναλήψεις της `goalDbsLabeling`, σε κάθε μία από τις οποίες αυξάνεται κατά ένα το `depthLimit`. Προαιρετικά μπορεί να καθοριστεί το αρχικό `depthLimit`, η προεπιλεγμένη τιμή του οποίου είναι 0.

Αν ο χρήστης επιθυμεί να εκτελεστεί μόνο μία επανάληψη του αλγορίθμου με σταθερό `depthLimit`, τότε θα πρέπει να χρησιμοποιήσει την κλάση `goalDbsLabeling`, καθορίζοντας βέβαια το `depthLimit`.

```
goalDbsStepping(NsIntVarArray& Vars,
                int depthLimit=0,
                VariableHeuristic *varHeuristic = new VarHeurMRV,
                ValueHeuristic *valHeuristic = new ValHeurFirst)

goalDbsLabeling(NsIntVarArray& Vars,
                int depthLimit,
                VariableHeuristic *varHeuristic = new VarHeurMRV,
                ValueHeuristic *valHeuristic = new ValHeurFirst)
```

2.2.5 Αναζήτηση με περιορισμένες αναθέσεις

Η σειρά με την οποία εξετάζονται οι μεταβλητές πρέπει να παραμένει σταθερή, οπότε ο χρήστης δεν μπορεί να διαλέξει το ευριστικό επιλογής μεταβλητής για αυτή τη μέθοδο. Μπορεί όμως να διαλέξει το ευριστικό επιλογής τιμής. Επίσης, πρέπει να καθορίζεται το `lanLimit`, δηλαδή το όριο των αναθέσεων για κάθε μεταβλητή.

```
goalLan(NsIntVarArray& Vars,
        int lanLimit,
        ValueHeuristic *valHeuristic = new ValHeurFirst)
```

2.2.6 Αναζήτηση με φραγμένη κατά βάθος ασυμφωνία

Και σε αυτήν τη μέθοδο η σειρά εξέτασης των μεταβλητών πρέπει να είναι σταθερή, οπότε ο χρήστης μπορεί να διαλέξει μόνο το ευριστικό επιλογής τιμής.

```
goalDbds(NsIntVarArray& Vars,
         ValueHeuristic *valHeuristic = new ValHeurFirst)
```

2.2.7 Αναζήτηση με επαναληπτική διεύρυνση

Η πιο συνηθισμένη επιλογή είναι η χρήση της `goalIbroad`, που εκτελεί την πλήρη έκδοση της μεθόδου. Αν ο χρήστης επιθυμεί να εκτελέσει μόνο μία επανάληψη της μεθόδου με σταθερό `breadthLimit`, μπορεί να χρησιμοποιήσει την `goalIbroadLabeling`. Σε αυτήν την περίπτωση θα πρέπει να καθορίσει το `breadthLimit`. Διατίθεται επίσης και η `goalIbroadInDomain`, για όταν θέλουμε να κάνουμε ανάθεση σε μία μόνο μεταβλητή, δοκιμάζοντας όμως τις `breadthLimit` πρώτες τιμές της μεταβλητής. Και σε αυτήν την περίπτωση το `breadthLimit` θα πρέπει να καθοριστεί.

```
goalIbroad(NsIntVarArray& Vars,
           VariableHeuristic *varHeuristic = new VarHeurMRV,
           ValueHeuristic *valHeuristic = new ValHeurFirst)

goalIbroadLabeling(NsIntVarArray& Vars,
            unsigned breadthLimit,
            VariableHeuristic *varHeuristic = new VarHeurMRV,
            ValueHeuristic *valHeuristic = new ValHeurFirst)

goalIbroadInDomain(NsIntVar& Var,
                   unsigned breadthLimit,
                  ValueHeuristic *valHeuristic = new ValHeurFirst)
```

2.2.8 Αναζήτηση με φραγμένη οπισθοδρόμηση

Σε αυτή τη μέθοδο το ευριστικό επιλογής τιμής πρέπει να είναι τυχαίο για να μην εξερευνάται το ίδιο κομμάτι του δένδρου αναζήτησης σε κάθε επανάληψη. Το προεπιλεγμένο ευριστικό είναι το `ValHeurRand`. Ο χρήστης μπορεί να διαλέξει ευριστικό επιλογής μεταβλητής κατά βούληση. Οι υπόλοιπες παράμετροι του constructor είναι οι εξής:

- `times`: Πόσες επαναλήψεις του αλγορίθμου θα εκτελεστούν
- `lookahead`: Το φράγμα στην οπισθοδρόμηση

```
goalBbs(NsIntVarArray& Vars,
        int times,
        int lookahead,
        VariableHeuristic *varHeuristic = new VarHeurFirst,
        ValueHeuristic *valHeuristic = new ValHeurRand)
new ValHeurFirst)
```

2.2.9 Αναζήτηση πρώτα κατά βάθος με επανεκκινήσεις

Και σε αυτή τη μέθοδο το ευριστικό επιλογής τιμής πρέπει να είναι τυχαίο. Το προεπιλεγμένο είναι το `ValHeurRand`. Ο χρήστης μπορεί να καθορίσει και το ευριστικό επιλογής μεταβλητής. Οι υπόλοιπες παράμετροι του constructor είναι οι εξής:

- `times`: Πόσες επαναλήψεις του αλγορίθμου θα εκτελεστούν
- `timeout`: Πόσοι κόμβοι κατά μέγιστο θα εξερευνηθούν σε κάθε επανάληψη

```
goalRdfs(NsIntVarArray& Vars,
         int times,
         int timeout,
         VariableHeuristic *varHeuristic = new VarHeurMRV,
         ValueHeuristic *valHeuristic = new ValHeurRand)
```

2.2.10 Αναζήτηση με σταδιακό περιορισμό πλάτους

Η παρακάτω συνάρτηση υλοποιεί την πρώτη από τις δύο νέες μεθόδους αναζήτησης, την gradual narrowing search. Η χρήση της είναι ίδια με αυτήν της `goalDfsLabeling`: Ο χρήστης μπορεί να καθορίσει ο ίδιος τα ευριστικά με βάση τα οποία θα γίνεται η επιλογή μεταβλητής και τιμής.

```
goalGnsLabeling(NsIntVarArray& Vars,
                VariableHeuristic *varHeuristic = new VarHeurMRV,
                ValueHeuristic *valHeuristic = new ValHeurFirst)
```

2.2.11 Αναζήτηση με συναρτησιακό περιορισμό πλάτους

Η συνάρτηση `goalDfsLabeling` υλοποιεί τη δεύτερη από τις νέες μεθόδους, την functional narrowing search. Η χρήση της είναι παρόμοια με την GNS, με τη διαφορά ότι χρειάζεται ένα επιπλέον όρισμα. Το δεύτερο όρισμα αυτό είναι δείκτης σε μία συνάρτηση `function`. Η συνάρτηση αυτή μπορεί να οριστεί από το χρήστη και δέχεται τρία ορίσματα. Το πρώτο όρισμα είναι ο αριθμός των παιδιών $nChildren$ στο εκάστοτε επίπεδο, το βάθος $depht$ στο οποίο βρίσκεται η αναζήτηση τη στιγμή που η συνάρτηση καλείται και το ύψος $treeHeight$ του δέντρου. Η συνάρτηση οφείλει να επιστρέψει έναν ακέραιο μεταξύ 1 και $nChildren$, ο οποίος θα καθορίσει τον αριθμό των παιδιών του επόμενου επιπέδου που θα εξερευνηθούν.

Το σύνηθες είναι να ορίζει ο χρήστης τη συνάρτηση `function`. Υπάρχουν ωστόσο διαθέσιμα προς χρήση τρία έτοιμα δείγματα συναρτήσεων που μπορούν να χρησιμοποιηθούν ως παραδείγματα ή για πειραματισμό.

```
goalFnsLabeling(NsIntVarArray& Vars_init,
                int (*function)(int,int,int),
                VariableHeuristic *varHeuristic = new VarHeurMRV,
                ValueHeuristic *valHeuristic = new ValHeurFirst)

int function(int nChildren, int depth, int treeHeight);
```

2.2.12 Επαναληπτική δειγματοληψία

Στη μέθοδο αυτή το ευριστικό επιλογής τιμής έχει την προεπιλεγμένη τιμή `ValHeurRand`, όμως ο χρήστης μπορεί να το αντικαταστήσει με το ευριστικό επιλογής τιμής της αρεσκείας του. Το ευριστικό αυτό θα πρέπει να είναι τυχαίο, αλλιώς σε κάθε επανάληψη της μεθόδου θα εξερευνάται το ίδιο μονοπάτι.

```
goalIsampStepping(NsIntVarArray& Vars, int numProbes,
                  VariableHeuristic *varHeuristic = new VarHeurMRV,
                   ValueHeuristic *valHeuristic = new ValHeurRand)
```

2.2.13 Ένα-δείγμα

Η `goalOnesampLabeling` υλοποιεί τη μέθοδο ένα-δείγμα. Μπορεί να χρησιμοποιηθεί και η `goalOnesampInDomain`, η οποία απλά επιλέγει μία τιμή

σύμφωνα με το ευριστικό και την αναθέτει στη μεταβλητή που έχει λάβει ως όρισμα.

```
goalOnesampLabeling(NsIntVarArray& Vars,
            VariableHeuristic *varHeuristic = new VarHeurFirst,
            ValueHeuristic *valHeuristic = new ValHeurFirst)

goalOnesampInDomain(NsIntVar& Var,
                  ValueHeuristic *valHeuristic = new ValHeurFirst)
```

Αναφορές

[1] Ποθητός, Ν.: *Naxos Solver: Εγχειρίδιο Χρήσης* Διαθέσιμο στη διεύθυνση http://www.di.uoa.gr/∼grad0780/naxos/naxos.pdf.