

Constraint-Satisfaction-Probleme

B. Nebel, S. Wölf
R. Mattmüller, M. Westphal
Wintersemester 2009/2010

Universität Freiburg
Institut für Informatik

Projekt P2

Abgabe: Mittwoch, 16. Dezember 2009

Im Rahmen dieses Projektes sollen Forward-Checking und Arc-Consistency Look-Ahead anhand von zufällig generierten binären Constraintnetzen verglichen werden.

Hinweis: Bei den Projekten geht es um die Implementierung, nicht so sehr um theoretische Aspekte. Wenn Sie an bestimmten Stellen nicht genau wissen, wie die Algorithmen und Definitionen aus der Vorlesung in die Praxis umgesetzt werden, können Sie daher gerne Fragen stellen — entweder per E-Mail oder in der Übungsgruppe.

Projekte können in C, C++, Java oder Python bearbeitet werden. Andere Programmiersprachen sind nach Absprache möglich; in diesem Fall bitte vor Bearbeitung des Projekts bei uns melden.

Die eingereichten Programme müssen die geforderten Ein- und Ausgabeformate verwenden, einige Tests bestehen und **ausreichend kommentiert** sein. Programme, die diesen Anforderungen nicht genügen, werden nicht akzeptiert, aber es besteht die Möglichkeit, innerhalb der Abgabefrist nachzubessern. Daher bitten wir darum, frühzeitig abzugeben, um ausreichend Zeit für Nachbesserungen zu haben.

Aufgabe P2.1 (Vorarbeit, 0 Punkte)

Schreiben Sie einen Generator, der binäre Constraintnetze in Abhängigkeit bestimmter Parameter zufällig erzeugt. Verwenden Sie die folgenden Parameter:

- v** für die Anzahl der Variablen,
- w** für die Anzahl der Werte in jeder Domäne,
- c** für die Anzahl der Constraints und
- d** für die Dichte der Constraints, d.h. die Anzahl der tatsächlichen Einträge geteilt durch die Anzahl der möglichen Einträge.

Die Parameter sollen dabei als Kommandozeilenargumente übergeben werden. Orientieren Sie sich bei den Ein- und Ausgaben Ihres Programms am folgenden Beispiel (beachten Sie bitte insbesondere die Sortierung der Constraints und Mengenelemente):

```
# random-network -v 4 -w 3 -c 4 -d 0.4
Variables:
V={v_0, v_1, v_2, v_3}
Domains:
```

$D_0 = \{V_{0_0}, V_{0_1}, V_{0_2}\}$
 $D_1 = \{V_{1_0}, V_{1_1}, V_{1_2}\}$
 $D_2 = \{V_{2_0}, V_{2_1}, V_{2_2}\}$
 $D_3 = \{V_{3_0}, V_{3_1}, V_{3_2}\}$
Constraints:
 $R_{0_1} = \{(V_{0_0}, V_{1_0}), (V_{0_1}, V_{1_0}), (V_{0_2}, V_{1_0}), (V_{0_2}, V_{1_1})\}$
 $R_{0_2} = \{(V_{0_0}, V_{2_1}), (V_{0_0}, V_{2_2}), (V_{0_1}, V_{2_0}), (V_{0_2}, V_{2_1})\}$
 $R_{1_3} = \{(V_{1_0}, V_{3_1}), (V_{1_1}, V_{3_1}), (V_{1_2}, V_{3_1}), (V_{1_2}, V_{3_2})\}$
 $R_{2_3} = \{(V_{2_0}, V_{3_0}), (V_{2_0}, V_{3_1}), (V_{2_1}, V_{3_1}), (V_{2_2}, V_{3_1})\}$

Falls Sie bereits Projekt 1 bearbeitet haben, können Sie Ihre Lösung von Aufgabe P1.1 verwenden.

Wählen Sie im Folgenden Variablen und Werte jeweils entsprechend der Sortierung, die auch für die Ausgabe der CSPs verwendet wird.

Aufgabe P2.2 (Forward-Checking, 2 Punkte)

Implementieren Sie den LookAhead-Algorithmus und verwenden Sie dabei ForwardChecking zur Auswahl der Variablenwerte.

LookAhead(\mathcal{C}, a):

Input: a constraint network $\mathcal{C} = \langle V, D, C \rangle$ and
a partial solution a of \mathcal{C}
(possible: the empty instantiation $a = \{ \}$)

Output: a solution of \mathcal{C} or “inconsistent”

SelectValue(v_i, a, \mathcal{C}): procedure that selects and deletes a
consistent value $x \in D_i$; side-effect: \mathcal{C} is refined;
returns 0 if all $a \cup \{v_i \mapsto x\}$ are inconsistent

if a is defined for all variables in V :

return a

else:

 select a variable v_i for which a is not defined

$\mathcal{C}' \leftarrow \mathcal{C}, D'_i \leftarrow D_i$ // (work on a copy)

while D'_i is non-empty:

$x, \mathcal{C}' \leftarrow \text{SelectValue}(v_i, a, \mathcal{C}')$

if $x \neq 0$:

$a' \leftarrow \text{LookAhead}(\mathcal{C}', a \cup \{v_i \mapsto x\})$

if a' is not “inconsistent”:

return a'

$D'_j \leftarrow D_j$ for all $j \neq i$

return “inconsistent”

SelectValue-ForwardChecking(v_i, a, \mathcal{C}):

```
select and delete  $x$  from  $D_i$ 
for each  $v_j$  ( $i \neq j$ ) for which  $a$  is not defined:
  for each value  $y \in D_j$ :
    if not consistent( $a \cup \{v_i \mapsto x, v_j \mapsto y\}$ ):
      remove  $y$  from  $D_j$ 
  if  $D_j$  is empty: // ( $v_i \mapsto x$  leads to a dead end)
    return 0
return  $x$ 
```

Geben Sie am Anfang das zufällig generierte Constraintnetz aus. Zudem soll jeder Aufruf von **LookAhead**(\mathcal{C}, a) eine Ausgabe analog zu folgendem Beispiel erzeugen:

```
LookAhead:
<Ausgabe der Domänen von C wie oben>
a={v_0:V_0_1, v_1:V_1_4, v_2:V_2_2, v_3:None}
```

None gibt an, dass der entsprechenden Variable noch kein Wert zugewiesen wurde. Geben Sie am Ende noch das Ergebnis des Algorithmus aus: Falls eine Lösung existiert, ist dies die resultierende Belegung, falls das Netzwerk nicht lösbar ist, der String **inconsistent**.

Aufgabe P2.3 (Arc-Consistency Look-Ahead, 1 Punkt)

Implementieren Sie die Funktion **SelectValue-ArcConsistency** und verwenden Sie sie in der **LookAhead**-Implementierung aus Aufgabe P2.2.

SelectValue-ArcConsistency(v_i, a, \mathcal{C}):

```
select and delete  $x$  from  $D_i$ 
repeat:
  for each  $v_j$  ( $j \neq i$ ) for which  $a$  is not defined:
    for each  $v_k$  ( $k \neq j$ ) for which  $a$  is not defined:
      for each value  $y \in D_j$ :
        if there is no value  $z \in D_k$  such that
          consistent( $a \cup \{v_i \mapsto x, v_j \mapsto y, v_k \mapsto z\}$ ):
            remove  $y$  from  $D_j$ 
      if  $D_j$  is empty: // ( $v_i \mapsto x$  leads to a dead end)
        return 0
until no value was removed
return  $x$ 
```