Universität Karlsruhe (TH)

Institut für Innovatives Rechnen und Programmstrukturen (IPD)

Real Life Programming (Praktikum im SS 2005) http://www.info.uni-karlsruhe.de/
Dipl.-Inf. Michael Beck beck@ipd.info.uni-karlsruhe.de
Dipl.-Inform. Rubino Geiß rubino@ipd.info.uni-karlsruhe.de
Dipl.-Inform. Sebastian Hack hack@ipd.info.uni-karlsruhe.de

Übungsblatt 6 Ausgabe: 24.05.2005 Besprechung: 12.07.2005

Aufgabe 1: *Java / PHP/ Perl nach C rufen (7 Punkte)*

In dieser Aufgabe soll der PBQP-Solver aus Aufgabe 2 an eine Skriptsprache angeschlossen werden. Im Rest dieser Beschreibung beziehen wir uns auf Python, aber ihr dürft auch eine der Folgenden wählen:

- Chicken
- Guile
- Java
- Mzscheme
- Ocaml
- Perl
- Php
- Pike
- Python
- Ruby
- Tcl

Wenn ihr Aufgabe 2 wie besprochen gelöst habt, sollte der eigentliche Algorithmus etwa das folgende Interface besitzen:

```
void pbqp_init(int mode, int num_matrices);

void add_nodecost(int id, vec *vector);

void add_edgecosts(int u, int v, mat *costs);

void solve();

double get_min();

int is_optimal();

int get_numnodes();

int get_solution(int u);
```

Möglich ist auch die Ersetzung der Funktionen `add_*` durch andere Varianten. Der Parser aus dieser Aufgabe soll verschwinden (oje :-).

Schreibt für dieses Interface eine Umwicklerbibliothek, so dass man es möglichst einfach aus der gewählten Skriptsprache aufrufen kann. Das Schreiben solcher Umwickler ist eine (oft) langweilige und insbesondere auch fehlerträchtige Sache (was vielleicht daher kommt, das sowas im wahren Leben oft auf die „Anfänger“ abgewälzt wird ;-).

Glücklichweise gibt es SWIG, den „Simple Wrapper Interface Generator“, der solche Aufgabe fast automatisch erledigt. Iht müßt für diese Aufgabe SWIG nicht benutzen, der Rest der Beschreibung bezieht sich jedoch darauf.

SWIG ist gut dokumentiert, ihr braucht eigenlich nur das Manual der Version 1.1 zu lesen und höchsten in das 1.3 hereinzuschauen. Der Clou an SWIG ist, das es die Umwicklerfunktionen für elementare Datentypen ohne weiteres Zutun generiert. Das einzige Problem sind die selbstdefinierten Datentypen. Dafür konstruiert ihr entweder n Funktionen, die diese Typen aus elementaren Werten zusammenbauen oder verwendet Typmaps.

Hinweis: Sollte SWIG eure Typen `vec *` und `mat *` partout nicht mappen wollen, definiert `vec_ptr` und `mat_ptr`...

Weiterhin kann es sinnvoll sein, zunächst eine C++-Klasse für das Interface zu schreiben, dann kann Swig auch eine Stellvertreterklasse der Skriptsprache generieren.

Im folgenden noch ein Beispiel, wie das Resultat aussehen könnte, wenn ihr das originale Interface nicht verändert habt und Python verwendet. Hier ist das Interface in der Stellvertreterklasse PBQP zusammengefaßt:

```
import sys
from pbqp import *

def read_value(f):
  """Read a value from a text stream."""
  s = ''
  while 1:
    c = f.read(1)
    if not c: # EOF
      return s

    c = c.strip()
    if not c:
      if s:
        return s
      continue
    s = s + c

def read_int(f):
  return int(read_value(f))

def read_float(f):
  return float(read_value(f))

def v_read(f):
  """Read a vector from a text stream."""
  vec = []
  for i in xrange(read_int(f)):
    vec.append(read_float(f))
  return vec

def m_read(f):
  """Read a matrix from a text stream."""
  rows, cols = read_int(f), read_int(f)
  matrix = []
  for i in xrange(rows * cols):
    matrix.append(read_float(f))
  return (rows, cols, matrix)


def solve(fname, mode):
```

```
  f = file(fname)
  num_nodes = read_int(f)
  num_edges = read_int(f)

  solver = PBQP(mode, num_nodes)

  # read the node cost
  for u in xrange(num_nodes):
    costs = v_read(f)
    flags = v_read(f)
    solver.add_nodecost(u, costs)

  # read the edges and its costs
  for e in xrange(num_edges):
    u, v = read_int(f), read_int(f)
    costs = m_read(f)
    solver.add_edgecosts(u, v, costs)

  solver.solve()

  # print solutions of decision vectors
  for u in xrange(solver.get_numnodes()):
     print solver.get_solution(u)

  # print minimum of objective function
  print "%6.0f" % solver.get_min()

  # is solution optimal?
  print solver.is_optimal()

  f.close()

def usage():
  print "Usage: %s filename h|hd|bd" % sys.argv[0]
  sys.exit(1)

if __name__ == "__main__":
  if len(sys.argv) < 3:
    usage()

  if sys.argv[2] == 'h':
    mode = PBQP_HEURISTICAL
  elif sys.argv[2] == 'hd':
    mode = PBQP_HEURISTICAL
  elif sys.argv[2] == 'bd':
    mode = PBQP_HEURISTICAL
  else:
    print "Error: Wrong parameter %s", sys.argv[2]
    usage()

  solve(sys.argv[1], mode)
```