Exam 10 May 2016 **Functional Programming and Intelligent Algorithms**
Duration 4 hours **IE501614**

Calculators and other accessories are not permitted.
Please jusftify your answers, including intermediate calculations and arguments where appropriate. Feel free to use diagrams and drawings where it can support your explanations.

Problem 1 ..... *(6%)*

Evaluate the following Haskell expressions:

(a) `[ x | x <- [1..10], 1 == x `mod` 2]`

(b) `map (*2) [2,3,7]`

(c)
```
zipWith f [-1,0,1] [3,2,0]
    where f x y = x + 2*y
```

Problem 2 ..... *(4%)*

Division is denoted by `/`. In Haskell, `(1/)` is a valid expression.

(a) What does `(1/)` mean?

(b) What is the type of `(1/)`?

Problem 3 ..... *(12%)*

Suppose you have a continuous function $f(x)$, and you know that $f(x_1) < 0$ and $f(x_2 > 0)$. The bisection method is used to find a root x_0 on the interval $[x_1, x_2]$ or $[x_2, x_1]$ such that $f(x = x_0) = 0$.

1. Give a detailed description of the bisection method as it would be implemented in Haskell. (Strict Haskell syntax is not necessary. Pseudo-code or a mathematical description is satisfactory as long as only purely functional concepts are used.)
2. Explain why recursion is the only way to implement bisection in a pure functional programming language such as Haskell.

Suppose you want to print the best estimate for the root x_0 after each recursive step.

3. Explain why such intermediate output is difficult to achieve in pure functional programming.
4. How could you implement such intermediate output in Haskell? If possible, address multiple approaches with a brief mention of advantages and disadvantages.

Problem 4 ..... *(9%)*

This problem concerns classification problems.

(a) Explain what we mean by classification problems.

(b) Give two different practical examples of common classification problems.

(c) Explain what we mean by a linear and non-linear classifiers respectively.

Problem 5 ..... *(12%)*

1. What are the main purposes of *data cleaning* routines (also known as data pre-processing)?
2. Describe various methods for handling the *missing values* problem.

Suppose we have observed an attribute (feature) *price* with the following characteristics: minimum value \$170, maximum value \$600, mean \$395, and standard deviation \$140.

3. Use min-max normalization to transform a price value of \$470 to the range [0.0, 1.0].
4. Use z-score normalization to transform a price value of \$470.

Problem 6 ..... *(12%)*

(a) Define *underfitting* and *overfitting* of classifiers.

(b) When working with neural network, what are the typical causes of underfitting and overfitting?

(c) Discuss different approaches to preventing under- and overfitting in a neural network.

(d) Explain what we mean by *bias-variance tradeoff*.

Problem 7 ..... *(9%)*

Many neural networks (such as backpropagation networks) are organised in layers, with one output layer and any number of hidden layers.

(a) Draw a (simple) diagram showing a backpropagation network, indicating each layer and all data which is transmitted through the network.

(b) What are the advantages of increasing the number of hidden layers?

(c) What are the disadvantages of increasing the number of hidden layers?

Problem 8 ..... *(9%)*

(a) Explain how a genetic algorithm (GA) can be used for intelligent computer-automated product design. Illustrate your answer with a brief example.

(b) Receding horizon control can be used to solve dynamic optimisation problems, that is, optimisation problems that have to be solved repeatedly at every time step. Explain the principles of receding horizon control. Illustrate your answer with a brief example.

(c) Describe the travelling salesperson problem (TSP) and explain why it is important in computer science and operations research.

You may aid your answers with information from the research papers presented in class.

Problem 9 ..... *(10%)*

Consider a continuous (real-valued) GA for learning strings such as the one that we have implemented in class. Assume that the target string to be learned contains characters in the following alphabet:

```
alphabet = "abcdefghijklmnopqrstuvwxyz0123456789.,;:?!_+-*/ " :: String
```

We can represent a character in the alphabet by its index. A target string of length n is therefore a list of n variables constrained to the range [`pLo`, `pHi`]:

```
pLo = 0.0 :: Double
pHi = fromIntegral $ length alphabet − 1 :: Double
```

Furthermore, assume that genes represent characters and are encoded as real-valued numbers in the range [`gLo`, `gHi`]:

```
gLo = 0.0 :: Double
gHi = 1.0 :: Double
```

(a) Assuming `gLo`=0.0 and `gHi` =1.0, implement the following functions, one for normalisation of variables in the range [`pLo`,`pHi`] to the range [`gLo`,`gHi`] and the other for the corresponding denormalisation:

```
-- Normalise a variable in range [pLo, pHi] to range [0.0, 1.0]
normalise :: Double → Double
normalise p =

-- Denormalise a variable in range [0.0, 1.0] to range [pLo, pHi]
denormalise :: Double → Double
denormalise pNorm =
```

(b) With the aid of the function `elemCost` below or otherwise, implement a function `stringCost` that returns the total number of unequal characters, compared character by character, of two strings. You may assume that the strings have equal length.

```
elemCost :: Eq a ⇒ a → a → Int
elemCost a b | a == b = 0
             | otherwise = 1

stringCost :: String → String → Int
stringCost s1 s2 =
```

(c) Suppose you want to optimise the following function using the continuous GA:

$$f(x, y) = x \sin 4x + 1.1y \sin 2y, \quad -10 \leq x, y \leq 10 \tag{1}$$

- How many genes would you need in each chromosome?
- Which values would you use for `pLo`, `pHi`, `gLo`, `gHi`?

Justify your answers.

Problem 10 ..... *(8%)*

Consider the following code extract from a binary GA for learning character strings such as the one that we have implemented in class:

```
-- Sum the number of unequal bits, bit by bit, of two chromosomes
chromCost :: Chromosome → Chromosome → Int

-- Define a type for (cost, chromosome) pairs
type ChromCost = (Int, Chromosome)

-- Generate (cost,chromosome) pair
chromCostPair :: Chromosome → Chromosome → ChromCost
chromCostPair target c = (chromCost target c, c)

-- Some test data
g1,g2,g3,g4 :: Gene
c1,c2,c3,c4,t :: Chromosome
p :: Population
g1 = [1,0,1,0,1,0]; g2 = [0,1,0,1,0,1]; g3 = [0,0,0,0,0,1]; g4 = [1,1,1,1,1,1]
c1 = [g1,g2]; c2 = [g3,g4]; c3 = [g1,g3]; c4 = [g2,g4]
```

(a) Suggest suitable types for bits, genes, chromosomes, and a population:

```
type Bit        =
type Gene       =
type Chromosome =
type Population =
```

(b) Implement a function `evalPop` with following type signature that evaluates each of the chromosomes in a population with respect to a target chromosome and returns a list of (cost, chromosome) pairs:

```
-- Evaluate a population and store in a list of (cost,chromosome) pairs
evalPop :: Chromosome → Population → [ChromCost]
evalPop target pop =
```

(c) An example function evaluation in `ghci` is given below:

```
*Exam2016binGA> evalPop t p
[(10,[[1,0,1,0,1,0],[0,1,0,1,0,1]]),
 (3, [[0,0,0,0,0,1],[1,1,1,1,1,1]]),
 (12,[[1,0,1,0,1,0],[0,0,0,0,0,1]]),
 (1, [[0,1,0,1,0,1],[1,1,1,1,1,1]])]
```

Based on this output, deduce the target chromosome `t`.

Problem 11 ..... *(9%)*

Consider an integer-based GA for solving the traveling salesperson problem (TSP) such as the one that we have implemented in class. We may define some useful types as follows:

```
type CityID     = Int                -- city id
type Position   = (Double, Double)   -- city position (x, y)
type City       = (CityID, Position) -- (city id, x-coordinate, y-coordinate)
type Gene       = CityID             -- integer
type Chromosome = [Gene]             -- list of genes
type Population = [Chromosome]       -- list of chromosomes
```

(a) Implement a function for single-point `crossover` with the following type signature:

```
-- Single point crossover
crossover :: Int → Chromosome → Chromosome → Population
crossover cp ma pa =
```

The function should *not* be specialised for TSPs but implement the ordinary textbook definition of single-point crossover.

(b) For the chromosomes `ma` and `pa` below, perform mating (that is, determine the two offspring chromosomes `o1` and `o2`) by using single-point crossover with a crossover point between bits 3 and 4:

```
ma = [1..10] :: [Chromosome]
pa = reverse ma :: [Chromosome]
```

(c) Using ordinary crossover operations such as that above has an undesirable consequence for chromosomes encoding TSP tours. Explain what this undesirable consequence is, and how you can avoid it.