Portfolio 7

Sam Bowyer

Numerical Optimization

Theory

One-Dimensional Optimization

Multi-Dimension Optimization

Solving a Magic-Word-Grid with Simulated Annealing

We'll first check 4x4 grids and define a fitness function that counts how many words can be found in the rows and columns of the grid:

```
n = 4

install.packages("qdapDictionaries")
```

```
## Installing package into '/home/dg22309/R/x86_64-pc-linux-gnu-library/4.2'
## (as 'lib' is unspecified)
```

```
library(qdapDictionaries)
isWord  <- function(x) x %in% GradyAugmented

fitness <- function(grid){
  # Add row scores
  score = sum(isWord(apply(grid, 1, function(x) paste(unlist(x), collapse=""))))

  # Add column scores
  score = score + sum(isWord(apply(grid, 2, function(x) paste(unlist(x), collapse=""))))

  # Add diagonal scores
  score = score + isWord(paste(diag(as.matrix(grid)), collapse=""))
  score = score + isWord(paste(diag(as.matrix(rev(grid[n:1,n:1]))), collapse=""))

  return(score)
}
```

Now to generate a random grid:

```
getGrid <- function(){
  grid = data.frame(matrix(sample(LETTERS, n*n, replace=TRUE), ncol=n, nrow=n))
  colnames(grid) = 1:n
  return(grid)
}
grid = getGrid()
print(grid)
```

```
##   1 2 3 4
```

```
##   1 2 3 4
## 1 Y K F A
## 2 U M C O
## 3 R A A H
## 4 S G V Z
```

```
print(fitness(grid))
```

```
## [1] 0
```

And we also need a method by which to move from one grid to a neighboring grid (in which case we'll say two grids are neighbours if they differ by one letter):

```
getNeighbour <- function(grid){
  pos = sample(1:n, 2, rep=TRUE)
  val = grid[pos[1], pos[2]]

  newGrid = grid
  newGrid[pos[1], pos[2]] = sample(setdiff(LETTERS, val), 1)
  return(newGrid)
}
neighbour = getNeighbour(grid)
print(neighbour)
```

```
##   1 2 3 4
## 1 Y K F A
## 2 U M C V
## 3 R A A H
## 4 S G V Z
```

```
print(fitness(neighbour))
```

```
## [1] 0
```

Finally we define the simulated annealing algorithm:

```
SA <- function(s0, t0, tempUpdate, maxIter){
  best = s0
  bestFitness = fitness(s0)

  temp = t0

  iter = 1
  while(bestFitness < n & iter < maxIter){
    neighbour = getNeighbour(best)
    neighbourFitness = fitness(neighbour)
    # difference = neighbourFitness - bestFitness

    if (neighbourFitness > bestFitness) print(neighbour)

    temp = tempUpdate(temp, iter)
    rand = runif(1)

    if (neighbourFitness > bestFitness | rand < exp(-1/temp)){
      best = neighbour
      bestFitness = neighbourFitness
    }
```

```
    iter = iter + 1
  }
  print(best)
  print(bestFitness)
}

SA(grid, 100, function(t, i) t*(1-i/100), 10000)
```

```
##   1 2 3 4
## 1 M F J J
## 2 R Y I Z
## 3 R Q N S
## 4 L Y H L
## [1] 0
```