Repairing solutions
Enrico Schumann
es@enricoschumann.net

1 Introduction

There are several approaches for including constraints into heuristics, see Chapter 12 of Gilli et al. [2011]. The notes in this vignette give examples for simple repair mechanisms. These can be called in DEopt, GAopt and PSopt through the repair function; in LSopt/TAopt, they could be included in the neighbourhood function.

> set.seed(112233)
> options(digits = 3)

2 Upper and lower limits

Suppose the solution \(x\) is to satisfy \(\text{all}(x \geq lo)\) and \(\text{all}(x \leq up)\), with \(lo\) and \(up\) being vectors of length(\(x\)).

2.1 Setting values to the boundaries

One strategy is to replace elements of \(x\) that violate a constraint with the boundary value. Such a repair function can be implemented very concisely. An example:

> up <- rep(1, 4L)
> lo <- rep(0, 4L)
> x <- rnorm(4L)
> x

[1]  2.127 -0.380  0.167  1.600

Three of the elements of \(x\) actually violate the constraints.

> repair1a <- function(x, up, lo)
  pmin(up, pmax(lo, x))
> x

[1]  2.127 -0.380  0.167  1.600

> repair1a(x, up, lo)

[1]  1.000  0.000  0.167  1.000

We see that indeed all values greater than 1 are replaced with 1, and those smaller than 0 become 0. Two other possibilities that achieve the same result:

> repair1b <- function(x, up, lo)
  {
    ii <- x > up
    x[ii] <- up[ii]
    ii <- x < lo
    x[ii] <- lo[ii]
  
    > repair1b(x, up, lo)

    [1]  1.000  1.000  0.167  1.000
The function `repair1c` uses the ‘trick’ that
\[
\text{pmax}(x, y) = \frac{x + y}{2} + \frac{|x - y|}{2},
\]
\[
\text{pmin}(x, y) = \frac{x + y}{2} - \frac{|x - y|}{2}.
\]

The third of these functions would also work on matrices if `up` or `lo` were scalars.

```r
> X <- array(rnorm(25L), dim = c(5L, 5L))
> X
[1,] 0.1962 0.434 -2.155 -1.588 1.000
[2,] 0.2284 1.231  0.975  0.975 1.818
[ reached getOption("max.print") -- omitted 3 rows ]
```
The speedup comes at a price, of course, since there is no checking (e.g., for \texttt{NA} values) in \texttt{repair1b} and \texttt{repair1c}. We could also define new functions \texttt{pmin2} and \texttt{pmax2}.

```r
> pmax2 <- function(x1, x2)
  ((x1 + x2) + abs(x1 - x2)) / 2
> pmin2 <- function(x1, x2)
  ((x1 + x2) - abs(x1 - x2)) / 2
```

A test follows.

```r
> x1 <- rnorm(100L)
> x2 <- rnorm(100L)
> t1 <- system.time(for (i in strials) z1 <- pmax(x1,x2) )
> t2 <- system.time(for (i in strials) z2 <- pmax2(x1,x2))
> t1[[3L]]/t2[[3L]] ## speedup
[1] 3.55
> all.equal(z1, z2)
[1] TRUE
```

```r
> t1 <- system.time(for (i in strials) z1 <- pmin(x1,x2) )
> t2 <- system.time(for (i in strials) z2 <- pmin2(x1,x2))
> t1[[3L]]/t2[[3L]] ## speedup
[1] 3.55
> all.equal(z1, z2)
[1] TRUE
```

One downside of this repair mechanism is that a solution may quickly become stuck at the boundaries (but of course, in some cases this is exactly what we want).

### 2.2 Reflecting values into the feasible range

The function \texttt{repair2} reflects a value that is too large or too small around the boundary. It restricts the change in a variable \(x[i]\) to the range \(up[i] - lo[i]\).

```r
> repair2 <- function(x, up, lo)
  {
    done <- TRUE
    e <- sum(x - up + abs(x - up) + lo - x + abs(lo - x))
    if (e > 1e-12)
      done <- FALSE
    r <- up - lo
    while (!done) {
      adjU <- x - up
      adjU <- adjU + abs(adjU)
      ...}
  }
```

adjU <- adjU + r - abs(adjU - r)

adjL <- lo - x
adjL <- adjL + abs(adjL)
adjL <- adjL + r - abs(adjL - r)

x <- x - (adjU - adjL)/2
e <- sum(x - up + abs(x - up) + lo - x + abs(lo - x))
if (e < 1e-12)
    done <- TRUE

x

> x
[1] 2.127 -0.380 0.167 1.600

> repair2(x, up, lo)
[1] 0.873 0.380 0.167 0.600

> system.time(for (i in strials) y4 <- repair2(x,up,lo))
user  system elapsed
0.108  0.000  0.104

2.3 Adjusting a cardinality limit

Let \( x \) be a logical vector.

> T <- 20L
> x <- logical(T)
> x[runif(T) < 0.4] <- TRUE
> x
[1] FALSE TRUE TRUE FALSE TRUE TRUE FALSE FALSE FALSE FALSE [ reached getOption("max.print") -- omitted 10 entries ]

Suppose we want to impose a minimum and maximum cardinality, \( k_{\text{min}} \) and \( k_{\text{max}} \).

> kmax <- 5L
> kmin <- 3L

We could use an approach like the following (for the definition of \texttt{resample}, see \texttt{?sample}):  

> resample <- function(x, ...) x[sample.int(length(x), ...)]
> repairK <- function(x, kmax, kmin) {
    sx <- sum(x)
    if (sx > kmax) {
        i <- resample(which(x), sx - kmax)
        x[i] <- FALSE
    } else if (sx < kmin) {
        i <- resample(which(!x), kmin - sx)
        x[i] <- TRUE
    }
}
printK <- function(x)
    cat(paste(ifelse(x, "o", "."), collapse = ""),
        "-- cardinality", sum(x), "\n")

For kmax:

```r
> for (i in 1:10) {
    if (i==1L) printK(x)
    x1 <- repairK(x, kmax, kmin)
    printK(x1)
}
```

```
.oo.oo.......oo...o.o -- cardinality 8
.oo.oo.........oo...... -- cardinality 5
.o...o........o....o -- cardinality 5
.o...o........oo...... -- cardinality 5
.o...o........o...o... -- cardinality 5
....oo.........oo...... -- cardinality 5
.o...o........o....o -- cardinality 5
.o...o........oo...... -- cardinality 5
.o...o........o...o... -- cardinality 5
```

For kmin:

```r
> x <- logical(T); x[10L] <- TRUE
> for (i in 1:10) {
    if (i==1L) printK(x)
    x1 <- repairK(x, kmax, kmin)
    printK(x1)
}
```

```
.........o............. -- cardinality 1
.........oo.o......... -- cardinality 3
........o............ -- cardinality 3
........o............ -- cardinality 3
........o............ -- cardinality 3
........o............ -- cardinality 3
........o............ -- cardinality 3
........o............ -- cardinality 3
........o............ -- cardinality 3
```

References