SAT: Box Wrappig

Arnau Abella
Universitat Politècnica de Catalunya

June 19, 2020

1 Variables and Constraints

The following constants are used in the box wrapping problem:

- $1 \leq W \leq 11$: width of the roll.
- $1 \leq L \leq (\sum_{b \in B} h_b)$: maximum length of the roll.
- $1 \leq B \leq 13$: total number of boxes.
- $w_{b \in B}$: width of the b-th box.
- $h_{b \in B}$: height of the b-th box.

The following variables are used in the problem:

- tl_{bij} where $i \in W, j \in L, b \in B$: box represented by its top-left coordinate.
- a_{bij} where $i \in W, j \in L, b \in B$: area of each box. Overlapping of areas is not allowed.
- r_b where $b \in B$: rotation of each box. *True* means the box is rotated.

The following constraints are used in the problem:

- **Exactly One**. Each box must appear once and only once in the paper roll.

$$\bigwedge_{b=1}^{B} ((\sum_{i=1}^{W} \sum_{j=1}^{L} tl_{bij}) = 1) \tag{1}$$

- **Bounds**. Each box must be inside the bounds of the roll.

$$
\begin{aligned}
&\forall i_{\in W} \forall j_{\in L} \bigwedge_{b=1}^{B} \neg tl_{bij} && w_b = h_b, i + w_b > W, j + h_b > L \\
&\forall i_{\in W} \forall j_{\in L} \bigwedge_{b=1}^{B} (\neg tl_{bij} \wedge r_b) \wedge (\neg tl_{bij} \vee \neg r_b) && w_b \neq h_b, i + w_b > W, j + h_b > L
\end{aligned} \tag{2}
$$

- **Overlapping**. This constrant is divided in two constraints:
  - Clauses to represent the **area** of each box.

$$
\begin{aligned}
&\forall i_{\in W} \forall j_{\in L} \bigwedge_{b=1}^{B} (\neg tl_{bij} \vee a_{bij}) \wedge \neg r_b && w_b = h_b \\
&\forall i_{\in W} \forall j_{\in L} \bigwedge_{b=1}^{B} (\neg tl_{bij} \vee a_{bij} \vee r_b) \wedge (\neg tl_{bij} \vee a_{bij} \vee \neg r_b) && w_b \neq h_b
\end{aligned} \tag{3}
$$

  - **At most one constraint** of these areas to prevent the overlapping.

$$
\forall b_{\in B} \sum_{i=1}^{W} \sum_{j=1}^{L} a_{bij} \leq 1 \tag{4}
$$

  Notice, this constraint is encoded using the *at most one logarithmic encoding*.

- (Optimization) By symmetry, placing the box on the left-half side is the same as placing the box in the right-half side. This constraint forces the biggest box to be on the left-half side, in the $(0, 0)$ coordinate.

$$
tl_{b00} \wedge (\bigwedge_{i=1,j=1}^{W,L} \neg tl_{bij}) \quad b \in B \tag{5}
$$

2 Implementation

This implementation is based on mios, a minisat-based CDCL SAT solver written in purely Haskell [1]. It is one of the few, open-source, SAT solvers written in Haskell with high performance.

The implementation is split in two files:

- `app/Main.hs`: main loop, at each iteration the objective function is minimized.
- `src/SAT.hs`: implementation of the constraints, clauses and custom encoding.

Let's have a look at the encoding of the constraints:

- Constraint 1

```
-- | Boxes must be placed exactly once in the paper roll.
exactlyOneClauses :: SAT
exactlyOneClauses =
  applyForAllBoxes_ $ \(b, _, coords) ->
    exactlyOne =<< traverse (`getTL` b) coords

exactlyOne :: [Variable] -> SAT
exactlyOne xs = do
  atLeastOne xs
  atMostOne xs

-- | At Least One Constraint
atLeastOne :: [Variable] -> SAT
atLeastOne = addClause . conjunctionOf
```

```
-- At Most One Constraint (Logarithmic Encoding)
atMostOneLogarithmic :: [Variable] -> SAT
atMostOneLogarithmic xs = do
  ys <- getNewVars m
  traverse_ (addClause . getClause) [ (x,y) | x <- zip [0..]
  ↪  xs, y <- zip [0..] ys]

    where
      n = length xs
      m = ceiling $ logBase @Double 2 $ fromIntegral n

      getClause :: ((Int, Variable), (Int, Variable)) -> Clause
      getClause ((i, x),(j, y))
        | testBit j i = neg x \/ y
        | otherwise   = neg x \/ neg y
```

- Constraint 2

```
-- | Boxes must be placed inside the paper roll.
insideTheBounds :: SAT
insideTheBounds =
  void $ applyForAllBoxes $ \(b, box, allCoords) ->
    forM_ allCoords $ \coord -> do
      tl  <- getTL coord b
      rot <- getRot b
      addBoundingClause coord box rot tl
  where
    addBoundingClause coord box rot tl
      | isSquare box =
        whenM (not <$> inside coord box) $ addClause (neg tl \/
        ↪  emptyClause)
      | otherwise = do
         whenM (not <$> inside coord box)        $ addClause
         ↪  (neg tl \/ rot)
         whenM (not <$> insideRotated coord box) $ addClause
         ↪  (neg tl \/ neg rot)
```

- Constraint 3

```
overlappingVariables :: SAT
overlappingVariables = do
  S{..} <- get
  sequence_ [ addCellsForEachBoxAndCoordinates (x,y) (b, box)
                | (b, box) <- zip [0..] _boxes
                , x <- [0.._w - 1]
                , y <- [0.._maxLength -1]]
  where
    addCellsForEachBoxAndCoordinates (x,y) (b, box)
      | isSquare box = do
          addCellsNoRotation (x,y) (b,box)
          -- Only once
          when (x == 0 && y == 0) $ do
            rot <- getRot b
            addClause (neg rot \/ emptyClause)

      | otherwise = do
          addCellsNoRotated (x,y) (b,box)
          addCellsRotated   (x,y) (b,box)
```

- Constraint 4

```
amoOverlapping :: SAT
amoOverlapping = do
  coords <- paperRollCoords
  nboxes <- uses boxes length
  forM_ coords $ \(x,y) -> do
    bxsVars <- traverse (getCell (x,y)) [0..nboxes-1]
    atMostOne bxsVars
```

- Constraint 5

```
biggestBoxTopLeft :: SAT
biggestBoxTopLeft = do
  addClause =<< singleClause <$> getTL (0,0) 0 -- boxes are
  ↪  sorted in decreasing ord.
  allCoords <- paperRollCoords
  let p (x,y) = x > 0 || y > 0
      allCoordsExceptTL = filter p allCoords
  tls <- traverse (`getTL` 0) allCoordsExceptTL
  traverse_ (addClause . singleClause . neg) tls
```

References

[1] Narazaki Shuji. A Minisat-based CDCL SAT solver in Haskell. urlhttps://hackage.haskell.org/package/mios-1.6.2