Packing Boxes with Score Constraints

Asyl L. Hawa

September 7, 2017

1 Example

1.1 Creating problem instance

In this example, we will attempt to find a feasible alignment of nine boxes, with score widths ranging between 1mm and 70mm. The minimum score separation constraint, or "threshold", is the minimum distance allowed between two scores from different boxes, and will be set at 70mm, the industry standard.

Initially, we create 18 random values between 1 and 70, which will be our score widths. We also need to add an extra "dominating box" with two scores of widths 70mm, which will be our "dominating scores", giving us a total of $n = 10$ boxes and 20 scores. These dominating scores will eventually be discarded.

We then sort the scores in non-decreasing order, and place them in a vector. The scores will now be addressed by their indices throughout the rest of the algorithm (see Table 1).

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Scores	3	4	12	22	25	35	36	37	38	45	49	54	54	55	55	64	65	66	70	70

Table 1: allScores vector.

Since every box consists of two scores, each score is randomly assigned a "mate", so that one score (the smaller score) represents the left-hand side of the box, and the other score represents the right-hand side of the box. Therefore, each pair of mates represents a box. The only exceptions to the random allocation of mates are the dominating scores - they must be assigned to one another. We can then assign measurements to each pair of mates to represent the width of each individual box. Note that we do not assign a box width to the dominating box, as it will not be used in the final solution.

Box	Scores		Box Width
1	0	6	314
2	1	10	372
3	2	16	297
4	3	11	220
5	4	17	959
6	5	9	738
7	7	12	622
8	8	15	635
9	13	14	859
10	18	19	0

Table 2: Mates.

Figure 1 shows the individual boxes with their respective score widths and box number. The box number will be used in the final solution to show the order in which the boxes should be placed.

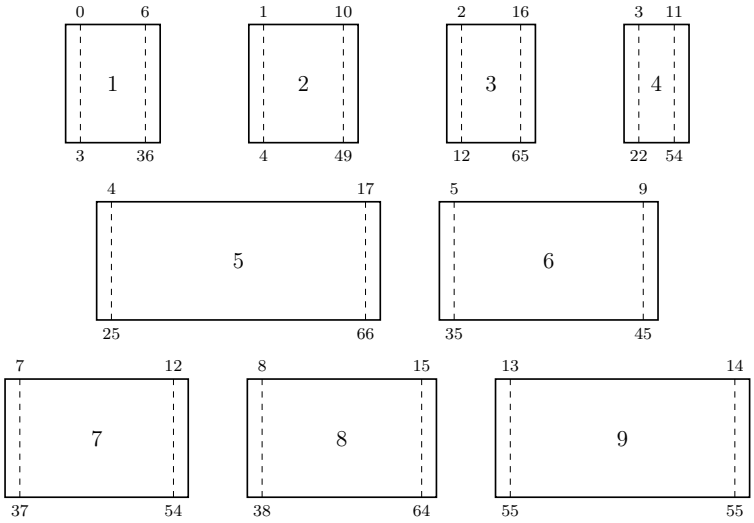

Figure 1: Individual boxes.

We then create an adjacency matrix, which contains information regarding which pairs of scores meet the minimum score separation constraint, as well as which pairs of scores are mates.

We can also depict this problem graphically (Figure 2) by representing each score as a vertex with an assigned value corresponding to the score width. A red edge between two vertices shows that the scores meet the minimum score separation constraint, and can be feasibly matched, and a blue edge between two vertices shows that the corresponding scores are mates, and therefore represent a box.

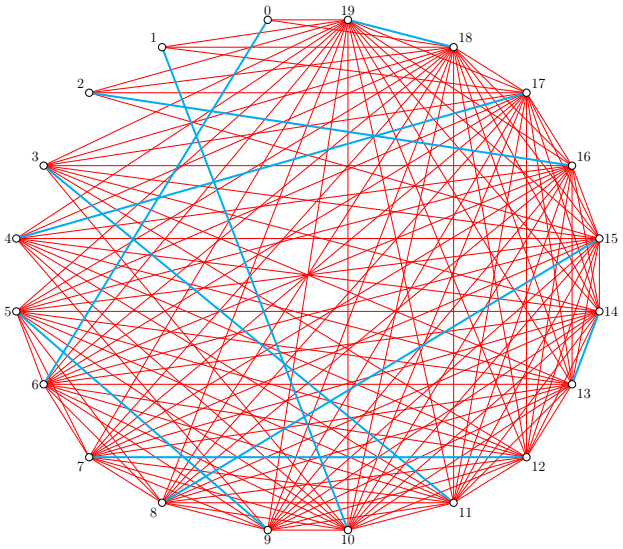

Figure 2: Threshold graph.

1.2 Matching scores

Once we have created the problem instance, we then have to attempt to match the scores together. If two scores can be matched, it means that the boxes can be placed next to each other, and the

total width of the two scores from the boxes is equal to or exceeds the threshold, and can therefore be scored using the knives in the scoring machine.

We already have a list of all the scores that any one score is adjacent to. For each score u, we now have to choose just one score v that score u will be matched with. Starting from the smallest score, we use the adjacent matrix to match each score to the largest possible score available. If a score u cannot be matched with the largest possible score v, due to v being u's mate, we instead match u with the second largest score available. As we can see in Table 3, score 7 was unable to be matched with the largest possible score, 12, as they are mates. Instead, 7 was matched with the next largest score, 11.

Score	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Match	19	18	17	16	15	14	13	11	12	10	9	7	8	6	5	4	3	2	1	0

Table 3: Match list.

Once the matching algorithm is complete, each score should be associated with exactly two other scores: its mate, and its match. For the instance to be feasible, all of the scores must be matched. If there are one or more scores that are unmatched, the instance is immediately infeasible.

Graphically, as depicted in Figure 3, if the instance is feasible, each vertex will be adjacent to two other vertices via two edges of different colours, one red edge, and one blue edge. The resulting graph is a 2-regular graph.

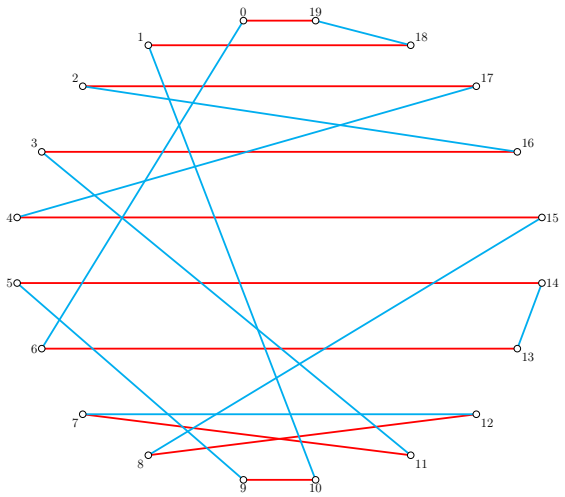

Figure 3: Match graph.

1.3 Mate-Induced Structure

Now that we have a list of matched scores, we can now attempt to create an *alternating cycle*, which is a cycle consisting of edges of alternating colours. In other words, the cycle will consist of an edge between scores that are mates, followed by an edge between scores that are matched. Therefore, each score will either be preceded by a mate and followed by a match, or vice versa. The cycles are easily formed using the mate list and match list, and are collectively called the mate-induced structure.

Cycle 1	0	6	13	14	15	9	10	1	18	19
Cycle 2	2	16	3	11	7	12	8	15	4	17

Table 4: Mate-induced structure consisting of two cycles.

If the mate-induced structure consists of only one cycle, then the instance is instantly feasible. We simply remove the dominating vertices and obtain a feasible alignment of the boxes. In this example we have two cycles, and so we must find a way to connect, or "patch", these two cycles together to create a single cycle.

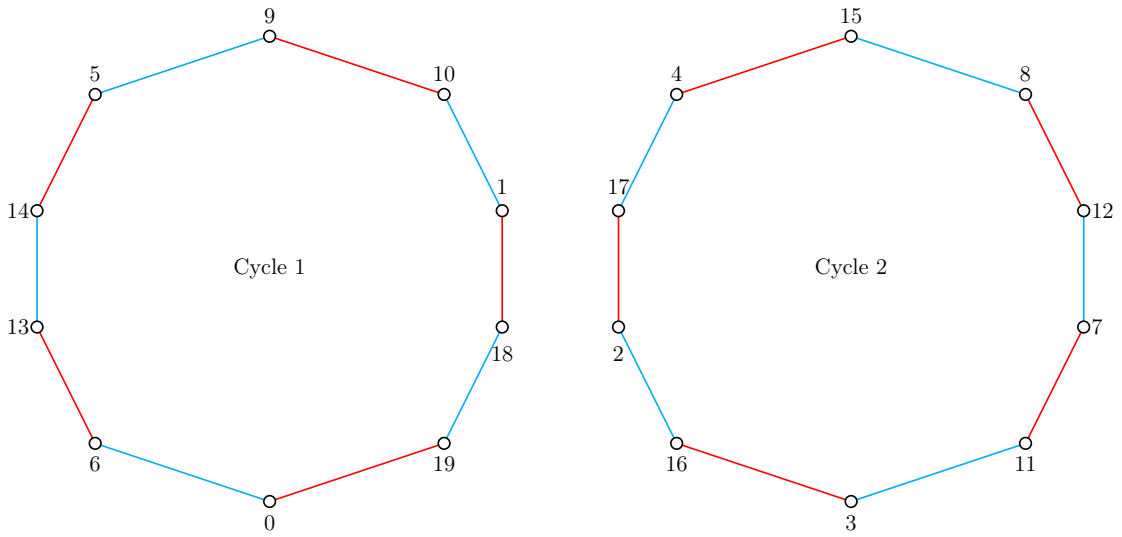

Figure 4: MIS.

1.4 Connecting cycles

In order to join these two cycles together to create a single cycle, we must find scores from different cycles that are able to be matched. If we are able to find such scores, then we simply remove the old matching edges, and add the new ones. In this case, we can use the edges $\{1, 18\}$ and $\{2, 17\}$, which are from different cycles. Score 1 can be feasibly matched with score 17, and score 2 can be feasibly matched with score 18. Therefore, we remove the original edges $\{1, 18\}$, $\{2, 17\}$, and create two new edges, $\{1, 17\}$ and $\{2, 18\}$. The two cycles have now been connect to create a single cycle, as shown in Figure 5.

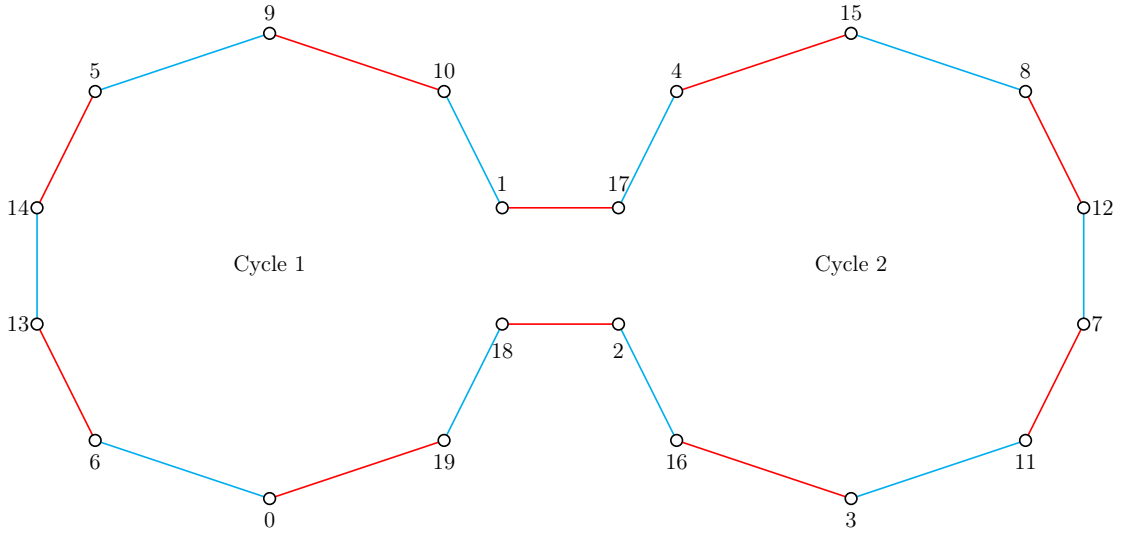

Figure 5: Patching MIS blue = mates, red = matching, explain dotted lines.

Now that we have a single alternating cycle containing all of the scores, we can remove the dominating scores, and obtain a feasible solution to the problem. The final feasible alignment of the boxes is shown in Figure 6.

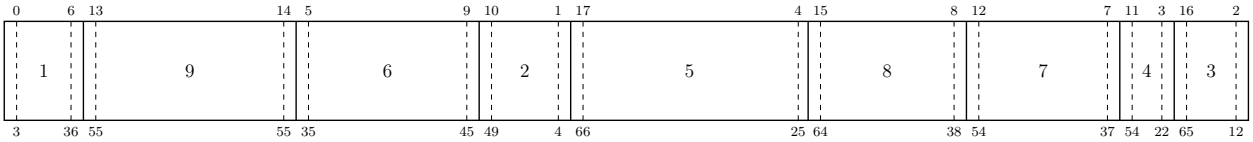

Figure 6: Final feasible alignment of boxes.

2 Output

Becker's original algorithm was designed to output a statement of feasibility or infeasibility for a large number of instances. This is useful to test the efficiency of the algorithm, but it does not allow a user to obtain an explicit solution for any given instance. In Becker's algorithm, he determines whether the cycles of the mate-induced structure can be connected, and if so, outputs a statement of feasibility - his program does not actually proceed to connect the cycles together. I have extended the program, and created functions that connect the cycles together, removes the dominating scores, and outputs the scores and box numbers in the order that will produce a feasible alignment of the boxes. The output corresponding to the above example is as follows:

```
Complete Path:  0 6 13 14 5 9 10 1 17 4 15 8 12 7 11 3 16 2
Order of Boxes:  1 9 6 -2 -5 -8 -7 -4 -3
Total Length of Path:  5043 millimeters.
```

If the box is aligned in its original form, i.e. with the smallest score on the left-hand side, the box number in the `Order of Boxes` output is positive. If the box needs to be rotated in the alignment, i.e. with the smallest score on the right-hand side, the box number is negative. This immediately provides the user with the information required to form a feasible alignment of the boxes.

3 Improvements

- Becker used arrays and set a specific size for each one - wastes memory and is inefficient. Instead I used vectors that could be expanded as required.
- Becker's code only evaluates whether a specific instance is feasible or infeasible. My code now outputs a solution to an instance in full, including the following:
  - Outputs final solution with score indices in order.
  - Outputs final solution with score widths in millimeters in order.
  - Outputs box widths in millimeters in order.
  - Outputs box number in order: if the number is positive, then the box's orientation is "regular" (the smallest score width is on the left-hand side), and if the number is negative, then the box's orientation is "rotated" (the smallest score is on the right-hand side).
  - Outputs the total length of the boxes in millimeters.
- All methods in the algorithm are now in separate functions, which makes the code easier to read and edit.

- The code determines whether only one T-cycle is required to connect all the cycles together, or whether multiple T-cycles are required.
- Box widths have now been added - each box has a width in millimeters.
- User input - users can create a simple text file containing score width, mates and box widths to use with the program.

4 Things to find out/do

- What are the industry standard strip lengths?
- Goulimis $n = 10$, is this true? Are there cases where more boxes are needed?
- Current speed of scoring in industry
- What happens before/after scoring in industry?
- Output to csv file?
- Calculate complexity
- Martello and Toth (1990b) MTP algorithm, adapt with additional constraints
- Exemptions on threshold, what if the instance in infeasible due to one box, can that box be held back and attempt to use in the next instance? Or can it still be used, but scoring run slower?
- How many scoring knives are used?
  - If only one set of scoring knives is used, then are the boxes placed on a conveyor belt? Does this affect the time?
  - If there are a specific number of scoring knives used, should we look at how many boxes are on each strip? I.e. if there are 4 scoring knives, then there should be only 4 scores on the strip, even if there is space available left on the strip?