Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

AMMM-Project

Algorithmic Methods for Mathematical Models

Authors: Gonzalo Solera - Meysam Zamani Forooshani

Master in Innovation and Research in Informatics.

Barcelona - December 17, 2019

1 Problem formulation and linear model

In all the variations of the problem (defined below) we are asked to find an ordering of n cars to be produced in an assembly line. These cars are from different classes, and we want the ordering of cars to be one such that some constraint about the sequence of classes in the assembly line is satisfied or minimized, depending on the variation of the problem.

For a given ordering of the n cars to be produced, let $(c_1, c_2, \ldots, c_n)$ be the sequence of car classes such that c_p is the car class of the p-th car to be produced. Let $\mathcal{C} = \cup_{p \in [1..n]} c_p$. Each class $c \in \mathcal{C}$ consists in a set of options $O(c)$. Let $\mathcal{O} = \cup_{c \in \mathcal{C}} O(c)$.

1.1 Variation A

In this variation we are asked to find a sequence of car classes $(c_1, c_2, \ldots, c_n)$ such that for each option $o \in \mathcal{O}$ and index $p \in [1..n]$ the following constraint is satisfied:

$$\sum_{p'=p}^{\min(p+k_o-1,n)} f(o, c_{p'}) \leq m_o \qquad f(o,c) = \begin{cases} 1 & \text{if } o \in O(c) \\ 0 & \text{if } o \notin O(c) \end{cases} \tag{1}$$

For given values of k_o and m_o strictly greater than 0.

For the ILP formulation, we will use the variables $isClassAt_{c,p} \in \{0,1\}$ for every $c \in \mathcal{C}$ and $p \in [1..n]$, and $isOptionAt_{o,p} \in \{0,1\}$ for every $o \in \mathcal{O}$ and $p \in [1..n]$. First, we will represent the need of including all the cars of each class in the ordering with the following equalities:

$$\forall_{c \in \mathcal{C}} \sum_{p=1}^{n} isClassAt_{c,p} = \sum_{c_i \in C | c_i = c} 1 \tag{2}$$

And we make sure that the solution of the ILP problem encodes a valid ordering with the following equalities:

$$\forall_{p \in [1..n]} \sum_{c \in \mathcal{C}} isClassAt_{c,p} = 1 \tag{3}$$

We also need the variables $isOptionAt_{o,p}$ to reflect the state of the variables $isClassAt_{c,p}$, and we accomplish it with the following inequalities:

$$\forall_{p \in [1..n]} \forall_{c \in \mathcal{C}} \forall_{o \in O(c)} isClassAt_{c,p} \leq isOptionAt_{o,p} \tag{4}$$

We translate the constraints from (1) into the following inequalities:

$$\forall_{o \in \mathcal{O}} \forall_{w \in [1..n-k_o+1]} \sum_{p \in [w..w+k_o-1]} isOptionAt_{o,p} \leq m_o \tag{5}$$

Since we are just interested in finding a valid ordering, in case it exists, a constant value for the objective function suffices.

1.2 Variation B

In this variation we are asked to find a sequence of car classes $(c_1, c_2, \ldots, c_n)$ such that the following expression is minimized:

$$\sum_{p \in [2..n-1]} f(c_p, c_{p-1}) \cdot f(c_p, c_{p+1}) \qquad f(c, c') = \begin{cases} 0 & \text{if } c = c' \\ 1 & \text{if } c \neq c' \end{cases} \tag{6}$$

For the ILP formulation, we can reuse the same variables from the previous variation. Furthermore, we can also use the equalities from (2), (3), (4) and (5).

Finally, we encode the expression to be minimized from (6) into the following objective function:

$$\sum_{p \in [2..n-1]} \sum_{c \in \mathcal{C}} (isClassAt_{c,p} \wedge \overline{isClassAT_{c,p-1}} \wedge \overline{isClassAt_{c,p+1}})$$

1.3 Variation C

In this variation we are asked to find a sequence of car classes $(c_1, c_2, \ldots, c_n)$ such that the following expression is minimized:

$$\sum_{o \in \mathcal{O}} \sum_{p \in [1..n]} g\left(o, \sum_{p'=p}^{\min(p+k_o-1, n)} f(o, c_{p'})\right) \qquad f(o, c) = \begin{cases} 1 & \text{if } o \in O(c) \\ 0 & \text{if } o \notin O(c) \end{cases} \tag{7}$$

$$g(o, x) = \begin{cases} 1 & \text{if } x > m_o \\ 0 & \text{if } x \leq m_o \end{cases}$$

For given values of k_o and m_o strictly greater than 0.

For the ILP formulation, we can reuse the same variables from the previous variations. Furthermore, we can also use the equalities from (2), (3) and (4).

Finally, we encode the expression to be minimized from (7) into the following objective function:

$$\sum_{o \in \mathcal{O}} \sum_{w \in [1..n-k_o+1]} \neg \left(\left(\sum_{p \in [w..w+k_o-1]} isOptionAt_{o,p} \right) \leq m_o \right)$$

2 Greedy and GRASP algorithms

For the following definitions we will use some of the notation from the previous section.

2.1 Greedy algorithm

The greedy algorithm that we propose builds the sequence choosing the immediately next class of the sequence, one class at a time until the entire sequence is determined. At each step p we will consider all the classes that still have cars to be added in the ordering, and we will get the load of the most loaded option $o \in O(c)$ of each class c, using the following definition:

$$load(o) = \left(\left[\sum_{p'=\max(1,p-k_o+1)}^{p-1} f(o, c_{p'}) \right] + 1 \right) / m_o \quad f(o,c) = \begin{cases} 1 & \text{if } o \in O(c) \\ 0 & \text{if } o \notin O(c) \end{cases}$$

The load of an option measures how full the current window of that option is. And it's important that the highest load is kept low because if it becomes greater than 1 then the sequence won't satisfy the constraint (1).

Then, for every class c that still has cars to be included in the ordering, we define $q(c)$ as follows:

$$q(c) = \max(\max_{o \in O(c)} (load(o)), \tau \cdot g(c_{p-1}, c_{p-2}) \cdot g(c_{p-1}, c)) \quad g(c, c') = \begin{cases} 0 & \text{if } c = c' \\ 1 & \text{if } c \neq c' \end{cases}$$

For some parameter τ and ignoring the second parameter of the max function when $p \leq 2$. In case of ties, the class with smallest maximum load will be considered better. Conceptually, we are using the already mentioned maximum load, and we are adding an extra penalization of τ if adding this class into the sequence increases the number of changes in the sequence. So the algorithm will try to select a class that doesn't increase the number of changes if the maximum option load is under the threshold τ. Otherwise, it will choose the class that keeps the maximum load as small as possible ignoring whether it increases the number of changes or not, since the constraint (1) is more important than minimizing the objective function.

Hence, the greedy algorithm would be as follows:

Algorithm 1: Greedy($\mathcal{C}$, τ)

```
foreach p ∈ [1..n] do
    c_p = arg min_{c∈C} q(c)
    if q(c_p) > 1 then
        /* Unable to find a feasible solution */
        return ∅
    if c_p.remaining_cars() == 0 then
        C.remove(c_p)
return (c_1, c_2, ..., c_n)
```

The execution of the greedy algorithm using $\tau = 0.7$, assuming that it has already chosen $c_1 = 1$, $c_2 = 1$ and $c_3 = 2$ using the dataset from "projectAB.dat"

would be:

class	$q(c_4)$	$q(c_5)$
1	1	0.7
2	2	2
3	1	2
4	1	0.7
5	1	2
6	1	2
7	2	2

2.2 GRASP algorithm

The GRASP algorithm that we propose is an straightforward adaptation of the previous greedy algorithm:

Algorithm 2: GRASP($\mathcal{C}$, τ, α)

foreach $p \in [1..n]$ **do**
  $Q = \{c \in \mathcal{C} | q(c) \leq 1\}$
  if $Q == \emptyset$ **then**
    `/* Unable to find a feasible solution */`
    return $\emptyset$
  $q^{\min} = \min_{c \in Q} q(c)$
  $q^{\max} = \max_{c \in Q} q(c)$
  $Q' = \{c \in Q | q(c) \leq q^{\min} + (q^{\max} - q^{\min})\alpha\}$
  c_p = random_element(Q')
  if c_p.remaining_cars() == 0 **then**
    $\mathcal{C}$.remove(c_p)
return $(c_1, c_2, \ldots, c_n)$

The execution of the GRASP algorithm using $\tau = 0.7$ and $\alpha = 0.5$, assuming that it has already chosen $c_1 = 1$, $c_2 = 1$ and $c_3 = 2$ using the dataset from "projectAB.dat" would be:

class	$q(c_4)$	RCL_4	$q(c_5)$	RCL_5
1	1	✓	0.7	✓
2	2	-	2	-
3	1	✓	2	-
4	1	✓	0.7	✓
5	1	✓	2	-
6	1	✓	2	-
7	2	-	2	-