SISA algorithm for Cover printing problem

Minelli Giovanni

September 13, 2021

1 Introduction

The use of approaches which combines linear programming algorithms with metaheuristics is not new for problems which allow a huge space of solutions. In particular there are many examples in the literature of applications of the simplex algorithm with higher-level optimizations obtaining good results even exploring only partially the search space [2][5][6].
The subject of this project has been the evaluation of Simulated Annehaling metaheuristic combined with the two phase variant of the SImplex algorithm. The first was intended to be used for the exploration of the search space and exploitation of possible solutions while the second given of those trial point by the SA algorithm was able to calculate the relative cost using them as solutions to the main problem. In particular the problem chosen for the evaluation is a normal variant of the Cover printing problem.

1.1 Description of the problem

The cover printing problem is an NP-hard problem [1] arising in a printing shop [1—15].

Let $M = 1,...,$ m be a set of different book covers (or advertisements, labels, tracts, etc.) of equal size, and suppose that d_i copies are to be printed of cover i, for $i \in M$. Suppose that for each print an unlimited number of identical plates can be made, and that an impression grid also called master or template can accommodate a specified number of t plates. The printing process is:

1. Compose an impression grid of t plates (some of them can be identical), and make a certain number of imprints with it. Each imprint produces one large printed sheet of paper which, once properly cut into t parts, yields t copies.

2. Repeat step 1 until all the required copies are made.

The printing cost comes from two sources: a per impression cost C_1, and a fixed cost C_2 for composing one impression grid (or grid, for short). Thus, the problem consists in determining the number of grids, the composition of each grid (which plates?), and the number of imprints made with each grid, so as to fulfill the copies' requirement at minimum total cost.

1.2 Matematical formulation

Recall M = 1,..., m and let N = 1,..., n. Namely,

$$n = \binom{m+t-1}{t}$$

is the total number of distinct grids. Consider the integer, no-negative m-by-n matrix $a_{i,j}$ whose columns are all pairwise distinct, each column corresponding to a possible impression grid.

For $(i, j)M \times N$ the number of plates of cover i in grid j is represented by $a_{i,j}$. Obviously

$$\sum_{i=1}^{m} a_{ij} = t, \text{ for } j \in N.$$

Thus the cover printing problem — also referred to as advertisement printing or label printing problem or job splitting problem — can be formulated as one of integer nonlinear optimization:

$$P = \begin{cases} \text{(min)} & C_1 \sum_{j \in N} x_j + C_2 \sum_{j \in N} y_j & & \\ \text{subject to} & \sum_{j \in N} x_j \, a_{ij} \geq d_i & i \in M, & (1) \\ & x_j(1 - y_j) = 0 & j \in N, & (2) \\ & x_j \geq 0 \text{ and integer} & j \in N, & (3) \\ & y_j = 0 \text{ or } 1 & j \in N. & (4) \end{cases}$$

2 Simulated annehaling

The main focus points of the SA algorithm are:

- **the terminating condition**: when to terminate the execution.
- **the annealing condition**: when to decrease the temperature.
- **cooling schedule**: how much is the temperature to be decreased.
- **the search process**: how explore the the space of possible solutions.

Mainly the structure of the algorithm and the parameters have been taken by literature [4] [3] and then adapted to the problem. Later on tuned and refined with tests on the fly. The final structure is:

Algorithm 1 SISA

```
1: procedure SISA_OPTIMIZER
2:     temp ← MAX_TEMP
3:     initial_cover ← create_sol(dimension)
4:     S_best, S_iter ← (initial_cover, Simplex(initial_cover))
5:     while temp > MIN_TEMP do
6:         initial_cover ← create_sol(len(S_iter.cover_solution))
7:         S_iter ← (initial_cover, Simplex(initial_cover)
8:         stat_iter ← value_cost(S_best)
9:         for range(N_ITER) do
10:            S ← ▷ use S_iter as starting point to calculate a new solution at
    each cycle exploring the neighbourhood and calcuate the simplex over that
11:            if value_cost(S_iter) > S then
12:                S_iter ← S
13:                if value_cost(S_best) > S then
14:                    S_best ← S
15:            else
16:                S_iter ← resize(S_iter)
17:        temp ← update_temp(stat_iter-value_cost(S_best), temp)
```

The temperature is a parameter in simulated annealing that usually affects two aspects of the algorithm: the distance of a possible solution point from the next and the probability during a cycle of search, of accepting a solution even if not optimal. The temperature is fixed at a value dependant by the initial solution and the minimum bound is 100 times smaller of it.
The search proceed with a cycle of `n` iterations at the end of which the temperature is updated dependently by the value itself and the improvement in the solution found during the `n` iterations. This value of improvement acquire a bigger weight with the decrement of the temperature and the more is high the more it has influence on the decrease of the temperature itself.
Inside the iterations the standard SA behaviour has been coded. A new solution is picked at each iteration exploring a small neighbourhood around the solution of the previous iteration. After it's evaluation, it's acceptance can be decided by it's optimality or by a random factor influenced by the value of the temperature of the algorithm. In the negative case a procedure to exploit a solution outside of the current search area decide to enlarge or not the search space, adding a column to the matrix solution of the iteration.

2.0.1 Neighbourhood

The function to explore the neighbourhood is strictly dependant to the optimality of the solution: if the matrix is large, and some columns are not used (the zeros in the simplex solution indicate that such corresponding column is not used) and the solution is very good respect to the optimal one there is a good probability of resize the solution removing a single column. Indipendently from that the neighbourhood of the current solution is successively explored perturbing the values inside the matrix, of course still maintaining the properties of feasibility.

2.1 Resize

This procedure is a bit more articulate since this isn't a default default behaviour of the algorithm. The need I wanted to satisfy was to compensate the negative resize of the neighbourhood exploration end eventually escape a dip in the search space where no improvement can be made but still leave options to descend in a new optimum. To decide in which cases such action is necessary or not, have been considered the following parameters: the temperature, the value of the solution discarded in comparison to the best one of the current cycle of iterations, and the number of columns in the matrix solution. The following intuitions guided the choice of implementation:

- With high temperature give much more importance to the value of distance from optimum, but keep the probability down even if there is a bad (high difference) value of optimality: high value ← mid P
- With low temperature, since you are supposed to enlarge the search space gradually the probability increase: high value ← high P
- Since the probability of remove columns (in the neighbourhood function) is just influenced by the optimality and not by the temperature, the action probability in resize method can increase with the lowering of the temperature even if the optimality value is good with the certainty that the matrix will decrease anyway if the search process is in a good position. This allow to search in a bigger space even not escaping from a valley.

The possible combinations of situations can be described with the following distributions of probability (high P = it's very probable that will be applied a resize procedure to the solution matrix)

high temp, low cost, few cols = 0.3 % low-mid
high temp, low cost, many cols = 0.1 % low-low
high temp, high cost, few cols = 0.5 % mid
high temp, high cost, many cols = 0.01
low temp, low cost, few cols = 0.5 % low-high
low temp, low cost, many cols = 0.2 % low-mid
low temp, high cost, few cols = 0.9 % high
low temp, high cost, many cols = 0.05

3 Simplex

For the simplex part were analyzed different techniques and flavour of implementation before finally choose the Two phase variant of the algorithm. In particular from code tests and public documentation has been observed that the Revised version is more efficient than simple Simplex and that the Two-phase method compared with the Big-M method doesn't have significant differences in terms of performance. In practice, however, most computer codes utilizes the two-phased method. The reasons are that the inclusion of the big number M may cause round-off error and other computational difficulties.
To be noted that some rounding problem were encountered even in my implementation and therefore a rounding with a precision of $1x10^{-12}$ have been

applied at each tableau formation step.

Some details of implementation about the simplex:

- If the problem is unfeasible it will be recognized at the end of phase1
- Phase2 is needed to verify the optimality or recognize an unbounded situation

The procedure works as follow:

- A BFS of the simplex is a basic feasible solution of the linear program (*e.g.* maximize c^Tx subject to Ax=b, x≥0)
- A feasible solution is $x \in R^n$ for which there exists an m-element set $S \subseteq \aleph$ such that
  1. The (square) matrix A is nonsingular, *i.e.*, the columns indexed by B are linearly independent.
  2. x_j=0 for all $j \notin S$.
- If such a S is fixed, we call the variables x_j with j ∈ S the basic variables, while the remaining variables are called nonbasic.
- If during phase1, in maximization, Z (the objective) value become strictly negative (< 0) the problem is infeasible (minimization/positive)
- If at end of phase1, artificial variables are basic the solution is degenerate but valid
- Degeneracy happens when the equations in a tableau do not permit any increment of the selected nonbasic variable, and it may actually be impossible to increase the objective function value Z in a single pivot step.

Particular attention has been dedicated also to the possible problems derived by the resolution process of the LP algorithm: unfeasible cases or degenerancy during the solution search. These cases are well captured in the code and handled, moreover cases in which the simplex steps doesn't take to an improvement can be easily seen in the plot representing the objective value collected at each step during the execution.

4 Visual representation

For the representation of the solutions each best solution found has been stored with the value of the temperature at the moment of discovery. All the solutions are then showed in a 3 dimensional plot where the different sizes of the solutions are put in relation with the corresponding value of optimality, and the moment of discovery determined by the temperature. Since the temperature has been used also as indicator of progression in the optimization process, it has been represented both spatially and more intuitively with the aid of the colormap.
As mentioned before, there exist also a function to represent graphically the simplex resolution process, to evaluate the behaviour and spot eventual cases of degenerancy.

5 Results

The tests and evaluation has been forwarded on [8] dataset, obtaining good results of optimality with the current setting of parameters.

	TIME (m)			**OPTIMALITY (%)**			
N°	1°	2°	3°	1°	2°	3°	**≃AVG**
1	00:23.90	00:20.95	00:21.67	100,00%	100,00%	100,00%	100%
2	00:34.69	00:34.10	00:36.94	100,00%	100,00%	100,00%	100%
3	00:52.09	01:05.42	00:55.87	100,00%	99,35%	99,30%	100%
4	03:34.39	03:44.37	02:35.14	100,00%	100,00%	100,00%	100%
5	08:14.99	08:10.77	08:55.91	95,01%	93,07%	95,01%	94%
6	12:30.77	15:21.64	17:56.27	92,17%	93,77%	93,48%	93%
7	03:31.47	03:49.65	03:40.34	100,00%	96,55%	96,04%	98%
8	10:33.51	20:47.00	17:57.62	97,66%	93,56%	94,06%	95%
9	26:01.68	23:59.61	18:51.87	90,13%	95,09%	96,25%	94%
13	93:28.29	101:55.09	87:42.65	89,94%	93,58%	91,19%	92%
14	194:09.80	204:05.54	242:53.35	91,03%	89,65%	92,04%	91%

Table 1: Results on dataset computed with a CPU i5 3.6 GHz 4 core

References

[1] A. Ekici, O. Ergun, P. Keskinocak, and M.G. Lagoudakis, Optimal Job Splitting on a Multi-Slot Machine with Applications in the Printing Industry, Naval Research Logistics 57 (2010) 237–251.

[2] https://onlinelibrary.wiley.com/doi/epdf/10.1002/atr.1183

[3] https://www.researchgate.net/publication/250772567

[4] https://www.researchgate.net/publication/2619136

[5] https://web.mit.edu/15.053/www/AMP-Chapter-13.pdf

[6] https://www.hindawi.com/journals/mpe/2018/6193649/

[7] https://paginas.fe.up.pt/∼sfeyo/Docs_SFA_Publica_Conferences/SFA_JP_19960101_CCE_The%20Simplex-Simulated%20Annealing.pdf

[8] Cover printing instances dataset
https://www.matcuer.unam.mx/∼davidr/cpp.html

[9] Simplex calculator
http://simplex.tode.cz/en/steps

[10] Simplex - Big M method
https://www.dam.brown.edu/people/huiwang/classes/am121/Archive/big_M_121_c.pdf

[11] http://www.maths.qmul.ac.uk/∼ffischer/teaching/opt/notes/notes8.pdf

[12] https://www3.nd.edu/∼dgalvin1/30210/30210_F07/presentations/simplex_full.pdf