AN ALGORITHM FOR COST-MINIMIZATION OF MULTIPLE-IMAGE PRINTING

Jeffrey R. Routh, M.A., C.D.P., C.S.P.*
and
Stanford H. Rosenberg, J.D., Ph.D.**

Abstract: This work follows the work of Engstrom and Rosenberg. An efficient, iterative algorithm based on 'dynamic programming' concepts has been developed which minimizes both planning time and raw materials expended producing "gang runs" on printing press equipment. A computer program has been developed from that algorithm which will produce satisfactory results, in an acceptable period of time, for a large number of the gang run situations [which] a printing shop would experience.

Background

Engstrom and Rosenberg stated that there was no generally-accepted single technique for resolving the gang run problem; they developed an algorithm for resolving it. They stated that "by using this algorithm, an operator can write a program to run on a small table top calculator with 1K storage or a micro, mini, or general purpose computer." Since that article was published [in 1976], major changes have occurred in the economics of computing. Given that fact, Rosenberg decided to re-examine the problem. The purpose was to develop an algorithm which could be implemented cost-effectively, utilizing recently developed, state-of-the-art microcomputer technology.

Statement of the Problem

Basically, the problem is to distribute a variable number of jobs -- each of which requires a different length

---

*Associate Professor, Computer Information Systems
La Roche College; Pittsburgh, PA 15237
**Professor, Administration and Management
La Roche College; Pittsburgh, PA 15237

of runs -- over one or more plates so that the total cost for all the jobs is minimized.

Data Requirements

The following data are required:

1) the number of jobs to be printed ["total number of jobs"];
2) the length of runs for each of the different jobs ["length of runs"];
3) the number of images on each plate ["number of images per plate"];
4) the 'makeready' cost for each plate ["makeready cost per plate"];
5) the cost of paper per press-sized sheet ["paper cost"].

Algorithm Development

Thorough analysis of a general form of the problem indicates that the basic problem solution to minimize costs, in general, follows the logic stated in the algorithm below. [Note that the algorithm is not a direct extension of the algorithm developed by Engstrom and Rosenberg. The algorithm developed in the present work is the result of a complete re-analysis of the problem.] Immediately after the algorithm, an example problem is given. By studying the calculations performed at each step of the algorithm for the example problem, you will see how the logic of the algorithm is operationalized.

1) For any "given total number of plates," calculate the "total number of images" available on the plates;

   set the value of the "total number of images" to the value of the "given total number of plates" times the value of the "number of images per plate."

2) Calculate the number of "'surplus' images" available on the "given total number of plates":

   set the value of the "number of surplus images" to the value of the "total number of images" minus the value of the "total number of jobs."

3) Create a list (an array) of the "length of runs" of the different jobs.

4) Put the list (the array) of the "length of runs" of

the different jobs in descending order. I.e., the job with the highest "length of runs" is first . . . the job with the lowest "length of runs" is last.

5) Save the list (the array) created in step 4) as the "original sorted list of length of runs."

COMMENT: Steps 6) through 21) generate the first possible solution to the problem:

6) Set the value of "divisor-1" to the value of the "number of surplus images" plus one.

COMMENT: Steps 7) through 10) will make the "sorted list of length of runs" equal in length to the "total number of images" available on the "total number of plates."

7) Divide the "length of runs" of the first job [the job with the highest length of runs] by (the "number of surplus images" plus one):

set the value of the "quotient" to the value of the "length of runs of first job" divided by the value of "divisor-1."

8) If the number in "quotient" is not a whole number -- i.e., there are decimal places in the quotient -- round the number in "quotient" up to the next highest number.

9) Replace the "length of runs" of the first job (in the "sorted list of length of runs") with the "quotient" from step 7);

e.g.: if the value of "quotient" were 1250 and the value of the first "length of runs" (in the "sorted list of length of runs") were 5000, the number 1250 would replace the number 5000 as the first "length of runs" in the "sorted list of length of runs."

10) Add the number in "quotient" to the list (the array) of the "length of runs" "number of surplus images"-times;

e.g.: if the value of "quotient" were 1250 and the value of "number of surplus images" were 3,

1250 would be added to the "sorted list of length of runs" 3 times.

11) Put the new list (a second array) of the "length of runs" of the different jobs [created in steps 9) and 10)] in descending order.

This is then the "new sorted list of length of runs."

COMMENT: Steps 12) through 21) calculate the cost of this possible solution as follows:

12) Set the "total cost" to $0.00.

COMMENT: Steps 13) through 19) calculate the cost of paper for this possible solution:

13) Set "list number-1" to 0 (zero) minus the "number of images per plate":

e.g.: if the "number of images per plate" were 4, "list number-1" would be set to 0 - 4.

14) Set "counter-1" to 0.

15) Add 1 (one) to "counter-1."

16) Set "list number-2" to "list number-1" plus ("counter-1" times "number of images per plate") plus one;

COMMENT: The number in "list number-2" specifies which one of the list of "length of runs" (in the second array) will be used in the following steps;

e.g.: if "list number-2" were set to 8 in step 16), the 8th item in the "new sorted list of length of runs" (the second array) would be used in the following steps.

17) Multiply the "length of runs" specified in the "new sorted list of length of runs" (the second array) by "list number-2" times "paper cost":

"cost-1" = ["length of runs"("list number-2")] X "paper cost."

18) Calculate "total cost" as a 'running subtotal':

set the "total cost" to the "total cost" plus "cost-1."

19a) If "counter-1" is not equal to the "given total number of plates," then go to step 15);

19b) . . . if "counter-1" is equal to the "given total number of plates," then go on to step 20).

20) Calculate total makeready costs:

set "cost-2" to the "makeready cost per plate" times the "given total number of plates."

21) Calculate "total cost":

set the "total cost" to the "total cost" plus "cost-2."

22) Save the "new sorted list of length of runs" (the second array), "total cost," and the "given total number of plates" as the "lowest total cost solution."

23) . . . Calculate next possible solution . . . .

Example Problem

The data for the example problem are as follows:

NUMBER OF JOBS TO RUN:	6
NUMBER OF IMAGES PER PLATE:	4
'MAKEREADY' COST OF PLATE:	$50.00
COST OF PAPER PER SHEET:	$00.50
LENGTH OF RUNS OF JOB 1:	5,000
LENGTH OF RUNS OF JOB 2:	2,500
LENGTH OF RUNS OF JOB 3:	7,500
LENGTH OF RUNS OF JOB 4:	15,000
LENGTH OF RUNS OF JOB 5:	10,000
LENGTH OF RUNS OF JOB 6:	1,000

For the example problem, assume that the "given total number of plates" is 3. Therefore:

1) "Total number of images" = 3 X 4 = 12.

2) "Number of surplus images" = 12 - 6 = 6.

3) The original [unsorted] list of "length of runs" is:

5,000	[JOB 1]
2,500	[JOB 2]
7,500	[JOB 3]
15,000	[JOB 4]
10,000	[JOB 5]
1,000	[JOB 6]

4) The "sorted list of length of runs" [sorted in descending order] is:

15,000	[JOB 4]
10,000	[JOB 5]
7,500	[JOB 3]
5,000	[JOB 1]
2,500	[JOB 2]
1,000	[JOB 6].

6) "Divisor-1" = 6 + 1 = 7.

7) "Quotient" = 15,000 / 7 = 2,142.8571.

8) Since the "quotient" does have decimal places in it, round the "quotient" up to 2,143.

9) Replace the "length of runs" of the first job (in the "sorted list of length of runs") with the "quotient" from step 8):

15,000	->	2,143	[JOB 4]
10,000	->	10,000	[JOB 5]
7,500	->	7,500	[JOB 3]
5,000	->	5,000	[JOB 1]
2,500	->	2,500	[JOB 2]
1,000	->	1,000	[JOB 6].

10) Add "quotient" to the "sorted list of length of runs" "number of surplus images"-times:

["number of surplus images" - 6]

2,143	->	2,143	[JOB 4]	
10,000	->	10,000	[JOB 5]	
7,500	->	7,500	[JOB 3]	
5,000	->	5,000	[JOB 1]	
2,500	->	2,500	[JOB 2]	
1,000	->	1,000	[JOB 6]	
		2,143	[JOB 4]	-- once --

2,143	[JOB 4]	-- twice --
2,143	[JOB 4]	-- three times--
2,143	[JOB 4]	-- four times --
2,143	[JOB 4]	-- five times --
2,143	[JOB 4]	-- six times --

11) Put the new list of the "length of runs" of the different jobs [created in steps 9) and 10)] in descending order:

10,000	["length of runs"(1)]	[JOB 5]
7,500	["length of runs"(2)]	[JOB 3]
5,000	["length of runs"(3)]	[JOB 1]
2,500	["length of runs"(4)]	[JOB 2]
2,143	["length of runs"(5)]	[JOB 4]
2,143	["length of runs"(6)]	[JOB 4]
2,143	["length of runs"(7)]	[JOB 4]
2,143	["length of runs"(8)]	[JOB 4]
2,143	["length of runs"(9)]	[JOB 4]
2,143	["length of runs"(10)]	[JOB 4]
2,143	["length of runs"(11)]	[JOB 4]
1,000	["length of runs"(12)]	(JOB 6].

This is the "new sorted list of length of runs" (the second array).

12) "Total cost" = $0.00.

13) "List number-1" = 0 - 4 = -4.

14) "Counter-1" = 0.

15) Add 1 (one) to "counter-1": 0 + 1 = 1.

16) "List number-2" = -4 + (1 X 4) + 1 = 1.

17) Multiply the "length of runs" specified in the "new sorted list of length of runs" (the second array) by "list number-2" times "paper cost":

COMMENT: At this point:
["length of runs"(list number-2")] corresponds to ["length of runs"(1)] because "list number-2" is equal to 1;
["length of runs"(1)] is equal to 10,000.

"cost-1" = 10,000 X $0.50 = $5,000.00

18) Calculate "total cost" as a 'running subtotal':

"total cost" = $0.00 + $5,000.00 = $5,000.00

19) "Counter-1" = 1, which is less than the "given total number of plates," which is 3; repeat steps 15) through 18).

15) Add 1 (one) to "counter-1": 1 + 1 = 2.

16) "List number-2" = -4 + (2 X 4) + 1 = 5.

17) Multiply the "length of runs" specified in the "new sorted list of length of runs" (the second array) by "list number-2" times "paper cost":

COMMENT: At this point:
["length of runs"("list number-2")] corresponds to ["length of runs"(5)] because "list number-2" is equal to 5;
["length of runs"(5)] = 2,143.

"cost-1" = 2,143 x $0.50 = $1,071.50.

18) Calculate "total cost" as a 'running subtotal':

"total cost" = $5,000.00 + $1,071.50 = $6,071.50.

19) "Counter-1" = 2, which is less than the "given total number of plates," which is 3; repeat steps 15) through 18).

15) Add 1 (one) to "counter-1": 2 + 1 = 3.

16) "List number-2" = -4 + (3 X 4) + 1 = 9.

17) Multiply the "length of runs" specified in the "new sorted list of length of runs" (the second array) by "list number-2" times "paper cost":

COMMENT: At this point:
["length of runs"("list number-2")] corresponds to ["length of runs"(9)] because "list number-2" is equal to 9;
["length of runs"(9)] = 2,143.

"cost-1" = 2,143 x $0.50 = $1,071.50.

18) Calculate "total cost" as a 'running subtotal':

"total cost" = $6,071.50 + $1,071.50 = $7,143.00.

19) "Counter-1" = 3, which is equal to the "given total number of plates," which is 3; go to step 20).

20) Calculate total makeready costs:

"cost-2" = $50.00 X 3 = $150.00.

21) Calculate "total cost":

"total cost" = $7,143.00 + $150.00 = $7,293.00.

22) Save the "new sorted list of length of runs," "total cost," and "given total number of plates" as the "lowest total cost solution."

In step 22) the data for this possible solution to the problem are stored as the "lowest total cost solution." These data are then compared with the "total cost" data for the second, and succeeding, possible solutions. When a solution with a lower "total cost" is found, the data saved as the "lowest total cost solution" are updated. At the end of the process -- when the "total cost" for all possible solutions for the "given total number of plates" has been calculated -- the data saved as the "lowest total cost solution" will be the "overall lowest total cost solution" for the "given total number of plates."

The above algorithm applies only to the first of all the possible solutions for a specific "given total number of plates." Calculation of the second, and succeeding, possible solutions requires some modification to, and extension of, the algorithm.

The logic for calculating the second possible solution is as follows:

A1) Subtract one from the divisor of the "length of runs" of the first job [the job with the highest length of runs]:

set "divisor-1" to "divisor-1" minus one.

A2) Establish "divisor-2" as the divisor of the "number of runs" of the second job [the job with the second highest length of runs]:

set "divisor-1" to (the "number of surplus images" plus 2) minus "divisor-1."

A3) If "divisor-2" is not greater than or equal to 2, then go to step A1).

COMMENT: For each possible solution, every divisor must be greater than or equal to 2. A divisor of 1 does not distribute one job over more than one image on the plate.

IMPORTANT NOTE: Step A4) determines whether or not this iteration of the algorithm provides a "valid" problem solution. A "valid" solution is defined (for this problem) as one in which the sum of the values in all the "divisors" minus the "number of divisors" equals the "number of surplus images."

A4) If ("divisor-1" plus "divisor-2" minus 2) is greater than the "number of surplus images," then go to step A1);

If ("divisor-1" plus "divisor-2" minus 2) is equal to the "number of surplus images," proceed to step A5).

IMPORTANT NOTE: If ("divisor-1" plus "divisor-2" minus 2) is less than the "number of surplus images," this iteration of the algorithm does not provide a valid problem solution; if that is the case, an extension of these steps is required.

A5) Using the "original sorted list of length of runs," (the first array) repeat steps 6) through 10).

A6) [After repeating steps 6) through 10), then:] in steps 6) through 10), where '"length of runs" of the first job' occurs, substitute '"length of runs" of the second job' [the job with the second highest length of runs] for it.

A7) Repeat steps 6) through 10) again.

A8) Repeat steps 11) through 22).

To formulate a general algorithm, it is crucial to identify the repeating patterns that develop (in successive iterations of the problem-solving process). To help clearly identify those patterns, the logic required to generate a third possible solution will be detailed:

B1) Subtract one from the divisor of the "length of runs" of the first job [the job with the highest length of runs]:

set "divisor-1" to "divisor-1" minus 1.

B2) Subtract one from the divisor of the "length of runs" of the second job [the job with the second highest number of runs]:

set "divisor-2" to "divisor-2" minus 1.

B3) If "divisor-2" is not greater than or equal to 2, then go to step B1).

B4) Establish "divisor-3" as the divisor of the "number of runs" of the third job [the job with the third highest length of runs]:

set "divisor-3" to (the "number of surplus images" plus 3) minus "divisor-1" minus "divisor-2."

B5) If "divisor-3" is not greater than or equal to 2, then go to step B2).

IMPORTANT NOTE: Step B6) determines whether or not this iteration of the algorithm provides a "valid" problem solution.

B6) If ("divisor-1" plus "divisor-2" plus "divisor-3" minus 3) is greater than the "number of surplus images," then go to step B2);

If ("divisor-1" plus "divisor-2" plus "divisor-3" minus 3) is equal to the "number of surplus images," then proceed to step B7).

IMPORTANT NOTE: If ("divisor-1" plus "divisor-2" plus "divisor-3" minus 3) is less than the "number of surplus images," this iteration of the algorithm does not provide a valid problem solution; if that is the case, an extension of these steps is required.

B7) Repeat the logic of steps A5) through A7).

B8) In steps 6) through 10), where '"length of runs" of the first job' occurs, substitute '"length of runs"

of the third job' [the job with the third highest length of runs] for it.

B9) Repeat steps 6) through 10) again.

B10) Repeat steps 11) through 22).

The logic detailed so far must be further extended to generate every possible combination of values for the "divisors" that satisfies the formula:

("divisor-1" minus one) [. . . plus ("divisor-2" minus one) [. . . plus "divisor-("number of jobs") minus one)]] equals the "number of surplus images."

Performance Tests of the Algorithm

The algorithm is still not elaborated completely enough to write a 'finished' computer program [directly from it] which accounts for all considerations implied in the original problem. But it is a correct algorithm from which a computer program can be developed. A test program based on this algorithm was written in Microsoft Corp.'s compiler version of the BASIC programming language. When that program was executed on an IBM-PC computer, a serious problem with it became apparent rather quickly: as the number of jobs increases, and/or as the number of images per plate increases, it takes proportionately longer and longer to generate the solution. Unfortunately, this is inherent with problem solutions based on 'dynamic programming' concepts.

In test runs of the program, the number of possible solutions actually tested to determine if they were "valid" solutions -- solutions which satisfied the formula given above -- was 63. The number of valid possible solutions found was 32. But, of course, simply testing all the possible solutions for 3 plates does not generate the final problem solution. The problem is to find the lowest possible total cost for all jobs -- and finding it requires that all solutions be checked for 2 plates, for 4 plates, for 5 plates, . . . and so on.

For 4 plates the number of possible solutions tested for validity was 847. The number of valid possible solutions was 382. For 5 plates, the total number of possible solutions tested for validity was 6,475 and the number of valid possible solutions was 2,380.

To summarize: in the logic developed so far, each of the possible solutions must be checked to determine if it is

a valid solution. Moreover, as you would expect, as the number of possible solutions increases, the number of valid solutions (generally) also increases in absolute terms.

The logic for each valid solution requires: 1) dividing some of the values of the "length of runs" in the "list of length of runs"; 2) modifying the "length of runs" of some of the jobs [those that were divided] in the "list of length of runs"; 3) adding values to the "list of length of runs"; 4) sorting the new "list of length of runs" into descending order; and 5) finding the cost of that solution. Each of those operations -- particularly the sort operation-- is time consuming.

In short, it was found that the algorithm is inefficient. Even when its procedures are performed by a computer it will not generate the desired result in an acceptable period of time. That is true, at least, for many of the gang run situations [which] a printing shop would experience.

Modifications to the Algorithm

Further analysis indicated several modifications which could be made to the logic of the algorithm. Those modifications substantially reduce the number of possible solutions which have to be tested for validity while the algorithm still generates a 'satisfactory' result.

For example, an analysis of a 'worst possible case' scenario suggested changes which reduced the number of possible solutions to be checked.

A 'worst possible solution' would be: simply make as many plates as there are jobs. For each job, make a plate with as many "up" as there are images on the plate. At least that divides the "length of runs" of each job by the "number of images" per plate. If paper is relatively expensive per sheet and the makeready cost is relatively low per plate, that approach (presumably) results in an overall savings due to reduced paper cost. Of course, the savings in paper cost would be offset to some greater or lesser degree by the makeready cost for each additional plate.

But of more importance to the present work -- in terms of the development of the algorithm -- the 'worst possible case' dictated a revised procedure for setting the initial values of the "divisors." The advantage of the revised procedure is that the algorithm does not test possible solutions for which any "divisor" has a value higher than the number of possible solutions tested for validity.

Steps 1) through 5) of the logic for generating the first possible solution for any "given total number of plates" remain the same as described above. But step 6) is divided into two steps as follows:

6a) Set "divisor-1" to the "number of surplus images" plus one.

6b) If "divisor-1" is greater than the "number of images per plate," set "divisor-1" equal to the "number of images per plate."

Steps 7) through 22) remain the same.

The logic for calculating the second possible solution is as follows:

C1) Subtract one from the divisor of the "length of runs" of the first job [the job with the highest length of runs]:

set "divisor-1" to "divisor-1" minus one.

C2) Establish "divisor-2" as the divisor of the "number of runs" of the second job [the job with the second highest length of runs]:

set "divisor-2" to the ("number of surplus images" plus 2) minus "divisor-1."

C3) If "divisor-2" is greater than the "number of images per plate" set "divisor-2" equal to the "number of images per plate."

C4) If "divisor-2" is not greater than or equal to 2, then go to step C1).

IMPORTANT NOTE: Step C5) determines whether or not this iteration of the algorithm provides a "valid" problem solution. A "valid" solution is defined (for this problem) as one in which the sum of the values in all the "divisors" minus the "number of divisors" equals the "number of surplus images."

C5) If ("divisor-1" plus "divisor-2" minus 2) is greater than the "number of surplus images," then go to step C1);

If ("divisor-1" plus "divisor-2" minus 2) is equal to the "number of surplus images," proceed to step C6).

IMPORTANT NOTE: If ("divisor-1" plus "divisor-2" minus 2) is less than the "number of surplus images," this iteration of the algorithm does not provide a valid problem solution; if that is the case, an extension of these steps is required.

C6) Go to step A5).

Those modifications dramatically reduce the number of possible solutions tested. As with the original algorithm, a test program was written from this algorithm. It produced the following results: for 3 plates, the number of possible solutions tested for validity was 51; the number of valid possible solutions was 24. For 5 plates, the number of possible solutions was 1013; the number of valid possible solutions was 95. Compare that to the figures for the original algorithm: for 5 plates, the number of possible solutions tested for validity was 6,475 and the number of valid possible solutions was 2,380.

To assure that both the algorithm and program work correctly, a test was run with the "given total number of plates" set to 6 -- the 'worst possible case' solution for the example problem. The number of valid possible solutions was 1 -- exactly correct.

As suggested above, several additional modifications have been made to the logic of the algorithm(s) which further reduce the number of possible solutions which are tested for validity.

As work continued on the problem and additional changes were made to the logic of the algorithm, test programs were written from the different versions of the algorithm. Those test programs were run to determine whether or not the changes made might actually eliminate the best possible solution from consideration -- obviously an unacceptable effect.

The acid test was to compare the "overall lowest total cost solution" generated by the computer program with the best lowest total cost solution an 'experienced printing professional' could calculate 'by hand.' This was done with

numerous different sets of data. Data sets were tested which had fewer "number of jobs" than "number of images per plate." Data sets were tested in which the "number of jobs" was greater than the "number of images per plate." And data sets with equal "number of jobs" and "number of images per plate" were tested. In every case, the computer program equaled or bettered the solution calculated by hand.

Just as important, the computer program developed from the algorithm does produce 'satisfactory' results in an acceptable period of time (for a large number of the gang run situations [which] a printing shop would experience).

Two 'benchmark' runs that were done to evaluate the performance of both the algorithm and the program developed from it follow:

First, for the example problem shown previously, the program produced the overall lowest total cost solution in 39 seconds, including the time that it took to key-in the data. The actual run time the program took to calculate the solution was 6 seconds.

Before giving the data concerning the performance of the second benchmark run, it needs to be pointed out again that the algorithm employs an iterative, 'dynamic programming' approach to produce the solution. Because of that, as the number of jobs increases, and/or as the number of images per plate increases, the program does take longer and longer to generate the solution.

For the second benchmark the number of jobs was 8, the number of images per plate was 8, the makeready cost per plate was $200.00, and the cost of paper per sheet was $0.35. The program took 17 minutes and 33 seconds to produce its lowest total cost solution; the experienced printing professional took about 16 minutes to produce his lowest total cost solution. However, the solution produced by the program was $4,387.20, the solution produced by hand was $4,455.25 -- a difference of $68.05.

After careful evaluation of the performance of the program based on the agorithm developed above, it may be concluded that the savings which can be achieved by using it are significant enough to warrant its use. That conclusion applies even for gang run situations in which higher numbers of jobs and/or higher numbers of images per plate do increase the run time of the program.

One final comment must be made: the work done indicates that it is imperative that the program be written in a programming language that can be compiled: COBOL, Pascal, Compiler BASIC, etc. If the program is written using the interpreter version of BASIC that is built into most microcomputers on the market today, it will not execute quickly enough to produce the lowest cost solution in an acceptable period of time.

Literature Cited

Engstrom, Patricia M. and Rosenberg, Stanford H. 1976. "A Heuristic Approach to Cost-Minimization of Multiple-Image Printing," TAGA Proceedings 1976, pp. 323-30.