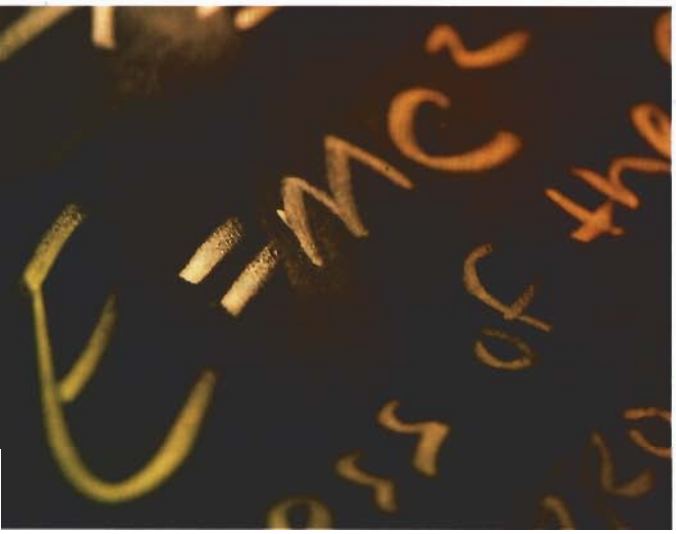
Solutions to Complex Problems



Intelligent inserting clarified

By Peter Somu and Jeff Sutton

This is the first in a series of articles designed to help *Mailing Systems Technology* readers identify and develop solutions for the complex issues and problems found in today's cutting edge document production environments. In the coming months, we'll look at intelligent insert assignment and utilization, issues and strategies surrounding database driven inserting and describe how postal manifest mailing can improve production control, increase output quality and reduce costs.

This issue's topic, Intelligent Insert Assignment, is a problem that many of our customers face in one form or another. With the proliferation of customer relationship management (CRM) and enterprise resource planning (ERP) systems, more and more customer communication documents are being generated in an "on demand" fashion. In fact, in many of today's mailing operations, multiple pockets of CRM nodes independently and autonomously generate requests for customer correspondence.

Consider, for example, a large bank or an insurance company's call center. They typically field several thousands of requests for documents from customers and prospective customers. With the use of CRM systems, these requests for documents are typically aggregated and batched for the purposes of document production. This process of consolidation or "batching" poses new challenges for a print production facility. The systems that consolidate the documents may or may not be aware of the production capabilities possible on the shop floor. Since on-demand, CRM-driven document requests are varied, the inserts typically called for are also diverse. If a final batched print stream calls for more than 10 unique inserts, and if the finishing equipment has a limitation of 10 insert pockets, the print stream cannot be finished using that equipment.

One way to deal with this "insert assignment" dilemma is to split the input print streams based on production machine insert capabilities. In splitting the incoming print streams, several factors need to be considered: the unique number of inserts, the type and number of inserters and insert capabilities, type of mailing, print stream segmentation requirements for efficient handling of print segments, the weight of individual inserts, etc.

What's So Hard About That?

In typical document composition and generation applications, the nature of the output print stream is generally known beforehand. For example, the type of output print stream to be generated (AFP, Metacode, etc.), the type of finishing equipment the document stream will be finished on, the type and quantity of external inserts that will be called into each package, etc. These and other operational parameters such as segment or batch size of each job, type of mail being produced (tri-folds, half-folds, flats, manuals, etc.) and the geographic distribution of mail can be carefully controlled during the document production process. Composition tools in the market today allow for comprehensive logic to be implemented during the print stream production process. Of particular interest to this article, is the fact that the "type" of inserts to be used within a job is known beforehand. For example, if an operation has a Series 9 inserter configured to use 10 insert pockets, the document composition process must be forced to honor this constraint.

It is often more desirable to combine many smaller print streams to form larger streams for printing and operational efficiencies, as well as postal savings advantages when distribution and volume supports 3/5 digit presort. The dilemma is when multiple document streams with different insert requirements and are merged to form a larger print stream, it is possible that the new consolidated print stream may not be machinable on a single inserter due to the total number of unique inserts required. Consider for example that Print Stream 1 requires 10 unique inserts (each ➡

document calling for one or more of the 10 inserts) and Print Stream 2 requires another 10 unique inserts. A combination of these two print streams calls for a total of 20 unique inserts. If the two original print streams were to be finished on an inserter with a maximum of 10 insert pockets, the combined print stream is no longer machinable on that inserter. The problem compounds when more than two print streams are combined, and further, when each of those print streams calls for a disparate set of inserts.

An Exhaustive Approach...

A simple but computationally intensive approach can be devised to easily solve the "insert" distribution dilemma. Consider the example defined above, with two print streams and a total of 20 unique inserts. If the resultant print stream is to be finished on a machine with a maximum of 10 pockets, then the following approach could be used to make the resultant print streams machinable:

- Combine the print streams into one print stream.
- Evaluate all possible combinations of groups of 10 inserts. Mathematically, this is 20C10 (combinations of 10 at a time from a total of 20). This evaluates to 20!/(10!)(20-10)! = 184,756 combinations.
- For each of those combinations, evaluate the number of documents that will fit into a print stream with that combination of the inserts.
- As a final step, pick the combinations that offer the largest volume distribution from the exhaustive set, which honors the insert requirements.

As one can see, the number of possibilities that have to be evaluated grows extremely large as the number of inserts and print streams is increased.

Clearly, this type of an "exhaustive search" approach is not workable in a production-type environment. The computing resources required to evaluate and identify the optimum solution will usually fall well outside the boundaries of what is possible in the average document-production environment.

A Greedy Approach...

In response to customer need, we were challenged to develop a simpler approach to insert optimization. Utilizing problem solving methodology known as a "Greedy Approach," an algorithm was developed based on the notion of assigning documents into "buckets." Each bucket is a target print stream with a set of inserts defined as an "insert pattern." Documents are examined one by one to determine if the inserts being called for that document can be satisfied by the current set of inserts assigned to the bucket. If so, the document is tagged for the bucket. Here's how it works:

For each document (di) in the combined print stream:

Is there a bucket (b1) that has ALL of di's inserts in it?

- If yes, assign bucket b1 to document di. Compute the bit pattern of inserts for this document based on bucket b1's inserts.

If not:

- Is there a bucket (b2) that can hold ALL of di's inserts in it? (i.e, is there a partially empty bucket that can accommodate this document's inserts).
- If yes, put di's inserts in bucket b2 and assign bucket b2 to document di. Compute the bit pattern of inserts for this document based on b2's inserts. Move to consider the next document.
- If not (there is no current bucket that can accommodate this document's inserts), create a brand-new bucket, b3. Put di's inserts into b3. Assign bucket b3 to di. Compute the bit pattern of inserts for this document based on b3's inserts. Move to consider the next document.
- Check to see if b1, b2 or b3 is "full" (based on a machine's maximum number of inserts). If so, mark the bucket as full.

This approach is "greedy" in the sense that it does not wait or exhaustively explore all the possibilities, but rather, it makes decisions for each step based on what seems best with the information available at that time. The algorithm's objective is to simply arrive at a quick solution, not necessarily the most optimal solution. As explained in the previous section, an optimal solution will require examining every possible combination, and for tasks such as this, a greedy approach works extremely well.

Optimizing the Effectiveness of Assignment Patterns

The above algorithm will break the original print streams into output buckets. Each output bucket has a set of inserts assigned to it and corresponds to an output defined as

Mathematical Aside

If there is a set of "m" objects and we were to choose "n" objects from the set, mathematically this problem is called "m choose n". It is representad as mCn . It is also written as C(m,n) sometimes. The formula for arriving at this is :

$$C(m,n) = \frac{m!}{(n!)(m-n)!}$$

In the example used above, the problem is to choose 10 inserts out of a total of 20 inserts.

$$C(20,10) = \frac{20!}{(10!)(20-10)!} = \frac{20!}{(10!)(10!)} = \frac{20*19*18*17*16*15*14*13*12*11}{10!} = 184756$$

The following table illustrates the number of Combinations from a total of 20 inserts:

	Number of Unique Inserts: 20
# of Insert Pockets	Total Combinations
4	C(20,4) = 4845
5	C(20,5) = 15504
6	C(20,6) = 38760
7	C(20,7) = 77520
8	C(20,8) = 125970
9	C(20,9) = 167967
10	C(20,10) = 184756

"Print Stream." At the end of the run, each bucket will have a certain number of documents assigned to it with a set of inserts. Each document in the bucket will call for a "subset" of the inserts assigned.

One easy measure of the effectiveness of this algorithm is to count the number of buckets and the average number of documents in each bucket. From a practical perspective, this should result in a manageable number of buckets, i.e., not too many output jobs to handle or track. Also, the other extreme is that each bucket should not have too many documents assigned to it, i.e., a large print stream that is not practical to produce as a single job.

To this end, a practical limit such as a "segment size" can be employed. In the algorithm above, before assigning any document to an existing bucket (and after considering the insert requirements), a further check then can be made to ensure that the bucket's size (in terms of number of pages or documents) is within a manageable, pre-defined limit.

Alternately, if the algorithm is producing too many output jobs with small document distribution, one could consider combining fewer jobs with which to start. The algorithm can then be applied to each combination of the print streams separately.

How Does It Work in the Real World?

A solution based on this algorithm was implemented for a large bank that undertook a consolidation of print requests for their call centers. Individual requests for documents were collected into batches and provided to a document composition system. The document composition system then produced several AFP print streams per day. A system based on StreamWeaver (from Pitney Bowes), utilizing custom Perl code was developed to extract the inserts for all the documents and implement the exact algorithm described in this article. The Perl program generates the buckets and the document distribution of each bucket. The insert pattern for each document is also generated by the Perl program. The StreamWeaver system then used this output to go into the documents and appropriately compose the right output print streams.

The algorithm dynamically takes into consideration the availability of insert pockets, number of pockets, insert types and weights, run length of print streams, etc. This solution is a software solution module that has clearly defined XML interfaces which enables it to be seamlessly incorporated into an existing print/finishing software system.

As a result of this solution, the customer is able to receive document inputs from a variety of CRM/ERP systems, and the results are production friendly print jobs which are optimized for the production floor. Wall Street Trader Ivan Boesky, once said "Greed is good," and when it comes to insert assignment and optimization, he was right!

Peter Somu is principle and Jeff Sutton is consulting partner with Zen Systems. Zen Systems has years of experience working with industry leaders like Pitney Bowes, Bowe Bell & Howell, Kern and Xerox in support of their most critical customers. Contact the authors at contactus@zensys.com or by phone at 504-288-6202 or 908-369-0225.