534-63

N88-15635 116736

28P

1987

NASA/ASEE Summer Faculty Fellowship Program

Marshall Space Flight Center
The University of Alabama in Huntsville

Expert System Development for Commonality Analysis in Space Programs

Prepared by:	Dorian P. Yeager
Academic Rank:	Associate Professor
University and Department:	The University of Alabama Computer Science
NASA/MSFC:	
Laboratory:	Systems Analysis and Integration
Division:	Space Station Systems
Branch:	Systems Integration
NASA Colleague:	L. Dale Thomas
Date:	August 7, 1987
Contract No.	The University of Alabama in Huntsville NGT-01-008-021

ABSTRACT

The work represented by this report is a combination of foundational mathematics and software design. A mathematical model of the Commonality Analysis problem was developed and some important properties were discovered. The complexity of the problem is described herein and techniques, both deterministic and heuristic, for reducing that complexity are presented. Weaknesses are pointed out in the existing software (System Commonality Analysis Tool) and several improvements are recommended. It is recommended that: (1) an expert system for guiding the design of new databases be developed; (2) a distributed knowledge base be created and maintained for the purpose of encoding the commonality relationships between design items in commonality databases; (3) a software module be produced which automatically generates commonality alternative sets from commonality databases using the knowledge associated with those databases; and (4) a more complete commonality analysis module be written which is capable of generating any type of feasible solution.

ACKNOWLEDGEMENTS

The author wishes to thank NASA and ASEE for their support of a fine program. My experiences here have been extremely rewarding. Thanks also to Gerald Karr and Ernestine Cothran for their capable administration and their personal involvement in the program.

Special thanks go to my NASA colleague, L. Dale Thomas, without whose help the fundamental nature of the problem would not have been evident.

Thanks go also to Charlie Cothran and the crew at EL83, who made me feel welcome and comfortable from the beginning. You have an outstanding team.

INTRODUCTION AND OBJECTIVES

The purpose of this work is to assess the feasibility of an artificially intelligent software tool to aid in the process of identification of commonality alternatives. Commonality is the degree to which two or more end items share common characteristics. A high degree of commonality is to be desired as an engineering design criterion for obvious economic reasons. Commonality analysis attempts to enhance commonality by choosing a set of end items which spans all the needed functionality of a larger set, and choosing that set which represents a minimum cost according to some previously agreed-upon cost measure. The recommendation which is inferred by such a minimum-cost set of items is that only those items be implemented and that the functionality of the remaining items be achieved by a systematic substitution of additional copies of the items in the implementation set.

The commonality analysis process necessarily involves three key activities:

1. The gathering and organization of data.
2. Identification of commonality alternatives.
3. Evaluation of alternatives.

Automation by software is a desirable goal in all three areas. What this report recommends is the development of an integrated set of software packages which interacts with existing software to solve a variety of problems in commonality analysis.

CRITIQUE OF EXISTING SOFTWARE.

The Systems Commonality Analysis Tool (SCAT) is presently the only available tool for automating the above process. SCAT provides limited assistance in all three areas; however, definite improvements can be made, as explained below.

1. For the gathering and organization of data, SCAT provides a front end to a commercial database management system (DBMS). Through SCAT, one may create and modify commonality databases, and for sophisticated database functions one may enter the DBMS proper from within SCAT. SCAT assumes that certain attributes (the so-called "generic" attributes) apply to all databases. There are two sets of such generic attributes - one set for hardware items and one set for software components. The generic attributes are simply those attributes which are directly relevant to SCAT´s Life Cycle Cost (LCC) analysis of the item. One advantage of requiring data to be entered via the SCAT front end is that the user is constrained to always include these generic attributes.

There are two crucial ingredients missing from the above data gathering strategy. Firstly, SCAT gives almost no guidance concerning the selection, naming, and use of other attributes besides those specifically needed for its analysis. These other attributes, dubbed "discriminating" attributes, are chosen by the database administrator, based on his expert knowledge of the items in the database. A predictable consequence of this lack of guidance is that similar data will be encoded in dissimilar fashion. For example, one database administrator will create a new database which incorporates an attribute named LIQUID which takes on values "Y" or "N", the first value indicating liquid and the second gas. Another database administrator may create a database which incorporates similar items with similar properties, but will use a different name and different values for his attributes. For example, he may use the name TYPE with values "LIQUID" and "GAS".

The second missing ingredient is the information needed to form groups of commonality alternatives for the analysis process. The database creator possesses essential knowledge about which items may be considered common. The first way that such knowledge is brought to bear on the problem is that a set of sorting criteria is communicated to SCAT, whereupon SCAT sorts the data as specified. The hope is that when SCAT or the DBMS displays or prints the data, groups of common items will coalesce. The second kind of knowledge is that needed for selecting commonality alternatives from a sorted set of records. The SCAT user needs to possess the ability to scan the sorted data and pick out groups of common items. Thus there is "sorting" knowledge and there is "grouping" knowledge. The SCAT paradigm indicates that a sequence of sorting and grouping operations will identify one or more sets of items which qualify for comparative LCC analysis.

It must be said here that certain kinds of commonality options do not fit neatly into the SCAT paradigm. SCAT does not provide facilities to aid in identifying ways of "extending" the function of an item, nor is it capable of automatically doing a componentwise breakdown analysis of a set of complex alternatives. Such sophisticated techniques are aided by a tool like SCAT, but SCAT does not provide a framework to support them.

Putting aside for now the idea of developing a completely general tool, a more modest goal is to somehow automate the "sorting and grouping" technique. In order to do this, it is necessary to capture the knowledge needed in the sorting and grouping process. This knowledge may be the most important kind of "data" available. It is certainly the most difficult to capture. SCAT provides no help in capturing such knowledge.

2. For the identification of commonality alternatives, as indicated above, SCAT provides only the standard database sorting, marking, and subsetting functions. No guidance is given regarding what is a "good" sort criterion, or what are "good" criteria for displaying, marking, and saving subsets of a database. Indeed, unless expert knowledge is available, no software product can provide such guidance. Thus an improvement in this area requires an improvement in the facilities available for data gathering.

We have a definite advantage with knowledge of this form, however. It can be easily encoded. A sort operation is encoded as a series of (key,direction) pairs. For example, {(TYPE, Ascending), (VOLTAGE, Ascending), (DIAMETER, Descending)}. A subsetting operation is encoded as a relational expression involving the attributes of the database. An example is 8.6 LT VOL AND VOL LT 12.8. Finally, a grouping operation is encoded as a relational expression involving the attributes of two or more records of the database. An example of this is ABS(VOL(1)-VOL(2)) < 0.5. The third type of operation is especially interesting, since it is not an operation directly supported by standard DBMS´s.

It must be said here that knowledge of this sort cannot be gathered once and for all. The content and meaning of the data determine the content of the knowledge. As data is entered and as new databases are built, the knowledge needed to analyze that data for commonality alternatives will change. It is impractical, also, to require that all such knowledge reside in a central place such as a single file of data or a single program. This sort of knowledge belongs with the data itself. The knowledge relevant to a piece of data must be physically and logically associated with that data.

Once the knowledge is gathered and made available as an integral component of a given commonality database, the selection of alternatives in that database may be automated with the use of a "shell" program which reads both data and knowledge from the database and produces as its output a series of database subsets representing proposed sets of commonality alternatives. Each such set would be presented to the user for closer scrutiny. An opportunity would then be presented for the user to approve or disapprove of these choices. In case a given choice of commonality alternatives was too restrictive or not restrictive enough, the system would prompt the user for additional knowledge which might help to avoid repeating the error.

3. SCAT provides a sophisticated resource for evaluating commonality alternatives, once such alternatives have been identified. The SCAT user presents a subset database which he or she has identified as a set of potentially common items, and SCAT provides a comparative study of the LCC differences between producing each item as an individually designed and produced component (the "unique" option) and producing a single item from the set of items to serve its own function as well as those of all other items in the set. If there are n items, SCAT computes n+1 LCC estimates: one for each item, assuming it is chosen as the common item, plus one for the unique option. It then sorts on the computed life cycle costs, and displays the sorted data.

The problem with the above approach is that it makes two basic assumptions that may not in general be valid. The first is that every item in a set of commonality alternatives can be substituted for every other item. The second is that either (a) there will always be a single, optimal common item, and the most economical alternative is to replace all items by that common item, or (b) it is cheaper to produce all items separately, i.e. to choose the unique option.

Now it is doubtful that the designers of SCAT really believed the above assumptions. Indeed, if the SCAT user is aware that those assumptions are not always valid, he may still make considerable progress by using SCAT repeatedly and/or throwing away unnecessary information. But SCAT leads its user into false assumptions.

In fact, there are often asymmetric constraints that allow, say, a larger device to be substituted for a smaller one, but will not allow the smaller device to be used in the place of the larger one. The only feature of SCAT that has bearing on this situation is its use of what are called "critical" attributes. A critical attribute is one whose value must never be diminished in a substitution. For example, if diameter is critical then when substituting item A with diameter 5 for item B with diameter 7 we are obliged to use two of item A. In practice, substitution of multiple copies of one item for a single copy of another is not always feasible. The result is that commonality is not always a symmetric relationship. There is no room in the SCAT model for asymmetric commonality relations. In order to handle a case like this, the user typically has to perform a standard SCAT analysis and ignore certain alternatives.

Also, it may often happen that the best commonality solution does not present a single item to be substituted for all other items, but instead requires keeping some items, discontinuing development on other items, and making selected substitutions of items in the first set for items in the second. This type of solution is not only beyond the scope of SCAT, but cannot easily be solved even with repeated applications of SCAT. The extreme complexity of such a solution, even in cases involving relatively few alternatives, would cause a solution by repeated SCAT analyses to require months to complete. It is a moot point that such a solution is possible with repeated applications of SCAT, not only because of the potentially prohibitive amounts of time required, but also because SCAT does not present to the user an interface that suggests such solutions are possible, nor does it offer any features to simplify the extremely complex process of arriving at a general solution.

MATHEMATICAL CONSIDERATIONS

The complexity of a general solution to the commonality problem is immense. Furthermore, there is no significant body of knowledge available in technical and scientific literature which can be drawn upon to guide the solution process. Therefore a large part of the work represented by this report was foundational in nature. Due to the creative, mathematical nature of that work, it was felt that the most appropriate forum for its presentation was in an applied mathematics journal. A complete mathematical formulation of the problem is to be found in a paper (Yeager, 1987) submitted by the author to the journal, Operations Research. In that article some foundations are laid for an orderly assault on the general problem. The details of the paper are omitted from the report, but preprints are available from the author. An illustrative summary of the major results is presented below.

The data in a database is a collection of records describing a set $A = \{a_1, a_2, \ldots, a_n\}$ of items. The items may be valves, pumps, circuit boards, or anything for which sufficient data is available for analysis. There is a set of attribute functions defined on A, which represents a set of values associated with the items. Some example attributes are weight, density, volume, composition, and power consumption. A Life-Cycle-Cost (LCC) estimate on a given item requires that certain attributes apply to that item. The SCAT program requires that data on hardware items include 12 generic attributes, 11 of which have direct bearing on the LCC analysis.

Let us begin with an illustration of the magnitude of the mathematical problem and the complexity of a potential solution. There are two sources of complexity - one is the sophistication of the LCC formula itself, and another is the complexity of the algorithm one uses to select which items to retain and which to replace. The SCAT program does a thorough treatment of the first area and pays little attention to the second. In what follows we will attempt a preliminary investigation of that second question.

A solution to the commonality analysis problem has two components: (1) a partition of the set A into smaller subsets, and (2) a set of representatives of the subsets of the partition. For example, for n = 6 we may propose the following as a solution:

Partition: $\{\{a_1, a_2, a_3\}, \{a_4\}, \{a_5, a_6\}\}$.
Set of representatives: $\{a_3, a_4, a_5\}$.

The above "proposed solution" stipulates that we produce only items a_3, a_4, and a_5, that a_3 replace a_1 and a_2, and that a_5 replace a_6. A proposed solution "works", i.e. is a true solution to the problem, if the substitution strategy it advocates yields a minimum cost according to some agreed-upon scheme for assigning costs to proposed solutions.

To gain an appreciation for the complexity of the commonality analysis problem, consider that the number of possible solutions of the above type is given by the formula

$$\sum_{i=0}^{n-1} C_{n,i}(n-i)^i$$

where $C_{n,i}$ is the number of combinations of n things, taken i at a time.

The size of this number is on the same order as n!. The following table investigates its behavior for some small values of n.

n	2^n	$\sum_{i=0}^{n-1} C_{n,i}(n-i)^i$	n!
1	2	1	1
2	4	3	2
3	8	10	6
4	16	41	24
5	32	196	120
6	64	1057	720
7	128	6322	5040
8	256	41393	40320
9	512	293608	362880
10	1024	2237921	3628800
11	2048	1.821010E+07	3.991680E+07
12	4096	1.573291E+08	4.790016E+08
13	8192	1.436630E+09	6.227021E+09
14	16384	1.381086E+10	8.717829E+10
15	32768	1.393056E+11	1.307674E+12
16	65536	1.469959E+12	2.092279E+13
17	131072	1.618459E+13	3.556874E+14
18	262144	1.855042E+14	6.402374E+15
19	524288	2.208842E+15	1.216451E+17
20	1048576	2.727262E+16	2.432902E+18

Thus a computer with the capability to generate a million potential solutions per second (a very powerful computer indeed) would require about 865 years to generate all solutions for a set of 20 items.

Thus a "brute force" approach consisting of an algorithm to enumerate all possible solutions and choose one with the smallest associated cost would be impractical for values of n much greater than 10. The interesting thing is that the nature of commonality problems is that very seldom will one have more than ten to twenty candidates available for comparison, so that the "brute force" technique is not to be completely discounted. It must be applied very judiciously, however, with full knowledge of its high degree of complexity.

Fortunately, there are quite effective ways of "paring down" the size of the solution space. The simplest of these is the feasibility relation. There are two processes involved in the generation of a potential commonality solution: (a) choose a partition, and (b) choose a representative from each set of the partition. The feasibility relation constrains us in the number of ways we may choose such a representative.

The feasibility relation tells us when a given item may be realistically substituted for another. It may be quite simple, stating for example that item a_i may be substituted for item a_j only if a_i is "larger" in some sense than a_j. Or it may be quite complex, calling into play such attributes as chemical composition, weight, diameter - literally hundreds of possible factors.

The best situation is that in which the feasibility relation linearly orders the set of candidates. In that case there is only one choice for a representative of a given subset, i.e. the only item in that subset which is substitutable for every other item in that set. In this situation we can reduce the size of the solution space to the number of partitions of a set with n elements. Unfortunately, that too is a very large number even for relatively small n. We proceed, then, to develop a class of techniques for significantly reducing the size of the solution space. These techniques concentrate on reducing the number of partitions which must be examined.

In order to prevent our formulas from becoming too unwieldy and obscuring the essential nature of the problem, we will make some simplifying assumptions about the LCC formula. The primary simplification will be to assume linearity. In particular, we will use the following abstract formulation of the LCC cost of an item. Our LCC formula requires only three attributes. For item a_i, we will call these attributes d_i, q_i, and k_i. d_i is the design, development, test and engineering cost of producing the item. q_i is the quantity, i.e. the number of copies of a_i which will be needed. Finally, k_i is the per-unit cost of producing, deploying, operating, and maintaining the item. Many factors go into computing the per-unit cost attribute, such as weight, volume, density, energy consumption, mean time between failure, and expected service life of the space system. Actually k_i is the total of all cost factors which are directly proportional to the number q_i of items needed. For our purposes it suffices to assume they are precomputed and stored as a single attribute. We use the following as a simple first approximation to the cost of the item:

$$c_i = d_i + k_i q_i$$

The natural interpretation we now give to the cost of implementing the functionality of all items in a set K of items by substituting item a_i for every other item in K is

$$d_i + k_i \sum_{x \varepsilon K} q_x$$

We will refer to the above as the <u>linear</u> <u>cost</u> <u>function</u>. In contrast, the SCAT formula is a more complex sum of terms, most of which are either constant or linear in q_i. An exception is the PROD term, the production costs incurred in producing q_i copies of item a_i. PROD is nonlinear in q_i, but it is constant if $q_i = 1$ and approaches linearity in q_i as the "Learning Curve" parameter approaches 100%. (The Learning Curve is a user-adjustable system default in SCAT, assumed to be the same for all items in a given analysis.) When we say we are assuming linearity in q_i, then, we deviate slightly from the SCAT model. What we say about a solution using the above formula can be carried over into the general SCAT formula analysis only in a heuristic sense. We can be certain that in passing to the more general SCAT formula we will be introducing more, not less, complexity. What we get out of using the above formula, then, is a mathematical model which represents the simplest model that we may hope to obtain. Much can be said about a general solution to the problem without so constricting the form of the cost function. But the more complex the cost function the less can be said about a general solution.

The quantities d_i, k_i, and q_i are constants for a given item a_i, and we assume $q_i > 0$ for all i. The quantity q_i represents the total needed number of items for the period over which our cost projections are valid.

The linear cost function is well-behaved in a very important sense. It can be proven that if the feasibility relation makes no constraints on item substitutions (i.e. if every item may be substituted for every other item), and if the linear cost function is being used, then the minimum-cost solution will always be a SCAT-type solution. In particular, there will always be a single item which, when substituted for every other item, yields a minimum cost. There are only two reasons, then, to doubt the SCAT recommendations. One is that not always is it feasible to allow every item to substitute for every other item. Another is that the SCAT formula is not linear. The SCAT formula may be close enough to linear, however, to feel reasonably good about a SCAT analysis, provided the feasibility relation does indeed permit us to apply the recommendation SCAT gives.

Another interesting mathematical fact is that if we assume a more liberal substitution policy, that is if we for the purpose of analysis assume that more substitutions can be made than are in practice permissible, and if we then apply a procedure which leads to a minimum cost solution under the more liberal assumptions, and if the solution thus obtained is consistent with the original feasiblity constraints, then the solution we obtain is the minimum cost solution. Thus the recommendations made by SCAT may be used with a fair amount of confidence whenever they make sense.

The real problem with the SCAT recommendations is that they will not always make sense. There is no structure within SCAT to handle a feasibility relation which makes real constraints on substitutions. To create a framework in which such constraints may be factored into the solution requires some foundational mathematics.

The first tool which we wish to apply to aid in obtaining a solution to the commonality problem is the concept of a "separator". A separator is a relation used to separate a single set of a partition into two disjoint subsets. For example, if δ is a separator and $a_1 \; \delta \; a_2$ (read "a_1 is δ-related to a_2"), then the partition $\{\{a_1, a_2, a_3, a_4\}\}$, with a_1 being the chosen substitute, is less cost-effective than $\{\{a_1, a_3\}, \{a_2, a_4\}\}$, or $\{\{a_1\}, \{a_2, a_3, a_4\}\}$, or $\{\{a_1, a_3, a_4\}, \{a_2\}\}$, or any partition where a_1 is in one set and a_2 is in the other, if a_2 is chosen as the substitute in the latter.

For the above linear cost function, the relation δ defined as follows is a separator:

$a_i \; \delta \; a_j$ is true whenever a_i is a feasible substitute for a_j and

$$k_i >= k_j + d_j / q_j$$

Note that the above is equivalent to saying that $k_i q_j >= c_j$. In short, what we say when we say that this relation is a separator is that if we can always produce item a_j from scratch for no more than the cost of producing "q_j more" of item a_i, then we will never be better off to recommend a strategy which includes substituting item a_i for item a_j.

For examples of other cost functions and separator relations associated with those cost functions, see the paper (Yeager 1987).

What we are aiming for with the introduction of the concept of a separator relation is a way of reducing the number of potential solutions which must be examined. The overall strategy is to introduce an initial "solution" which is close to the actual solution in the sense that we can obtain the latter by a series of refinements of the former.

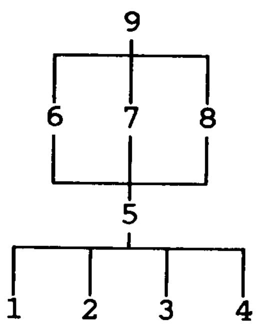
Suppose we partition the set of items into subsets which have the following property: each set K of the partition which contains item a_i also contains all items a_j which are <u>not</u> δ related to a_i in either direction. That is, if it is false that a_i is related to a_j and it is false that a_j is related to a_i, then a_i and a_j are in the same subset of the partition. This defines a unique partition of the set of items, which we will call the partition induced by the separator δ. Under certain conditions it can be shown that <u>every</u> true solution of the commonality problem is obtained by "refining" this particular partition, i.e. splitting its subsets into smaller subsets.

It turns out that one of the situations under which the above partition represents a valid initial estimate of a solution is that in which the elements of the set are linearly ordered by the feasibility relation.

Example 1. The following comes from a set of design data on nine types of storage tank intended for use on the NASA Space Station Project. Cost figures are in thousands of dollars.

Tank #	DDTE d_i	Unit cost k_i	Quantity q_i	Cost c_i	$k_i + d_i / q_i$
1	46.166	36.116	1	82.29	82.29
2	49.374	40.204	1	89.57	89.57
3	67.833	64.598	4	326.23	81.56
4	71.860	70.598	4	354.25	88.57
5	92.819	102.514	2	297.85	148.92
6	355.772	775.184	2	1906.14	953.07
7	366.685	810.760	6	5231.25	871.87
8	378.240	844.656	3	2912.21	970.74
9	464.314	1152.108	4	5072.75	1268.19
"Unique Cost"				16272.54	

The feasibility relation is based solely on size. Since the tanks are numbered in increasing order according to size, the relation allows each tank to replace all tanks numbered lower than it, thus establishing a linear order. The separator δ defined above is graphically depicted in the following diagram. Here the nodes are identified by tank number and the tanks which are δ-related to tank t are reachable from node t via a downward-trending path. For example, tank 9 is δ-related to all other tanks, and tank 7 is δ-related to tanks 1, 2, 3, 4, and 5.



The subsets forming chains of non-related items are {9}, {6,7,8}, {5}, and {1,2,3,4}. The mathematical properties we have established for separator relations assure us that any solution of this commonality problem is obtained by refining this partition.

Now let us contrast the performance of a "heuristic" solution with that of a solution using the above knowledge of the structure of the problem. A Prolog program to selectively search for the optimal feasible partition of this set took six minutes on an IBM XT to produce the following solution:

Set	Representative	Cost
{1,2}	{2}	129.78
{3,4}	{4}	636.64
{5}	{5}	297.85
{6,7,8}	{8}	9669.46
{9}	{9}	5072.75

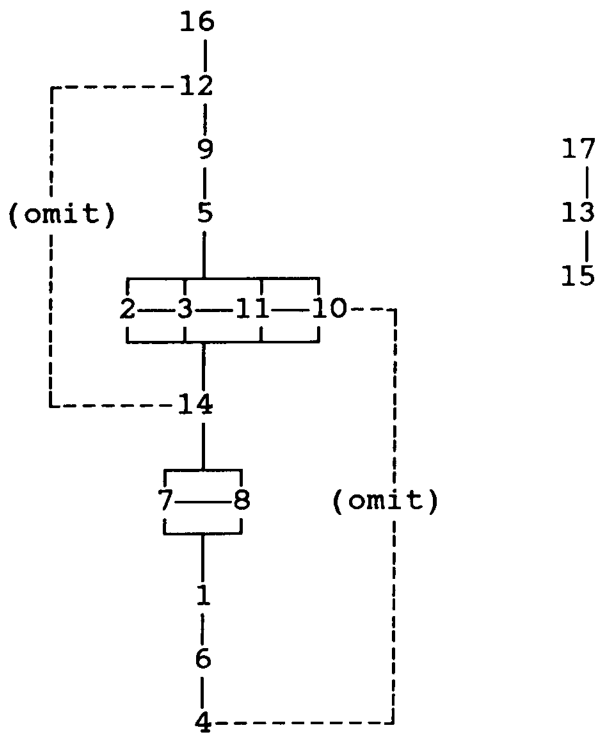
Minimum cost 15806.48

The same program, utilizing partition {{1,2,3,4}, {5}, {6,7,8}, {9}} as a starting point, arrived at the same solution in three seconds. Note that the optimal solution is only one immediate refinement away from that partition.

Example 2. A second set of data from an independent source, also pertaining to tanks proposed for use on the Space Station Project, is given below:

Tank #	DDTE d_i	Unit cost k_i	Quantity q_i	Cost c_i	$k_i + d_i / q_i$
1	153.110	192.750	2	538.61	269.31
2	566.140	1395.895	2	3357.92	1678.97
3	573.640	1388.401	4	6127.24	1531.81
4	91.985	88.815	3	358.43	119.48
5	606.570	1551.440	2	3709.45	1854.73
6	106.660	109.060	4	542.90	135.73
7	178.570	259.000	1	437.57	437.57
8	178.570	259.000	1	437.57	437.57
9	663.290	1883.300	1	2546.59	2546.59
10	566.140	1395.895	2	3357.92	1678.97
11	549.650	1412.390	1	1962.04	1962.04
12	604.520	3877.855	4	16115.92	4028.99
13	200.900	701.580	4	3007.22	751.81
14	306.376	1295.465	12	15851.90	1321.00
15	101.800	236.377	12	2938.32	244.86
16	1382.490	13966.940	5	71216.99	14243.44
17	459.315	2517.827	5	13048.42	2609.69
"Unique Cost"				145555.01	

The feasibility relation α for this set is more complex, and is illustrated by the following diagram:



This feasibility relation has two components: tanks 13, 15, and 17, which are linearly ordered with 17 being "most substitutable", and the others, which are related in a more complex way. This in fact represents two separate problems. No tank in one group may be substituted for any tank in the other group. So we split the problem up and attack it in two pieces. Looking at the larger, more complex group, we see that it is "almost" linearly ordered. Tank 16 may be substituted for any other tank in the group, for example. But tanks 7 and 8 are mutually interchangeable, and the tanks in the group 2, 3, 10, and 11 are mutually interchangeable. The most glaring exceptions are the two "omitted" relationships. If the relation were "transitive", then tank 10 would be an acceptable substitution for tank 4, and tank 12 would be an acceptable substitute for tank 14. But the source supplying the data specifically forbids those two substitutions. So we have a non-transitive feasibility relation.

Extending the feasibility relation to include the two omitted pairs gives us a transitive feasibility relation. If we obtain a valid solution based on that expanded relation, we have solved the problem. If not, we will have to approach the solution in another manner.

Let us assume for now that our relation is transitive. Even so, we cannot treat this problem the same as Example 1 because the feasibility relation is not linear. To handle this case, we introduce the notion of a <u>selector</u>. A selector is a relation β on the set A which has the property that if a β b then not only can a be substituted for b but whenever a and b are in the same set K and each of a and b can be substituted for any element of K, then a is always a more economical choice. The selector relations on a set, like the separator relations, are dependent on the cost function being used. A very simple selector σ for the linear cost function is defined as follows:

$a_i \ \sigma \ a_j$ if and only if
(1) a_i is substitutable for a_j ,
(2) $k_i <= k_j$, and
(3) $d_i <= d_j$.

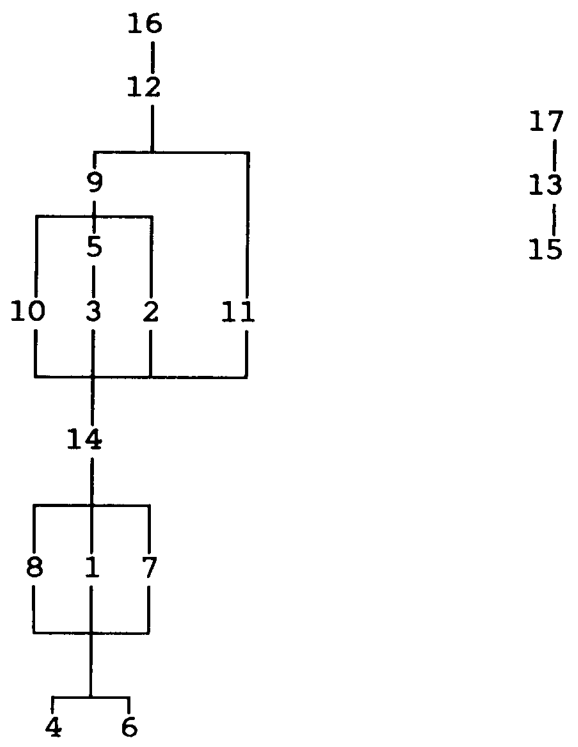
This is a very intuitive selector. It should be obvious that if both the initial costs and the per-unit costs associated with item a_i are less than those respective costs for item a_j , then a_i is always a better choice than a_j . We should point out here that there is another more finely discriminating selector for the linear cost function (Yeager, 1987), but this one will do for the current example.

Another selector, which works for any cost function, is the selector α' defined as a α' b if and only if a can be substituted for b but b cannot be substituted for a. Since there does not exist a set K containing both a and b in which both of a and b can be substituted for all elements of K, α' vacuously satisfies the requirements for a selector.

Finally, we observe that "the union of two selectors is a selector". In particular, if we combine the selectors α' and σ above into one relation, we still have a selector σ'.

It can be shown that if a separator δ is contained in a selector σ', and if the partition induced by δ has the property that for each set K of the partition there is an element t which is σ'-related to every element of K, then there is a minimum-cost solution which is a refinement of the partition induced by δ whose set of representatives includes all such elements t.

Now let us look at the relation δ:



The reader is invited to inspect this graph and confirm that whenever two elements are δ-related they are also σ'-related - i.e. δ is contained in σ'.

Here we see the importance of treating the example as two separate problems. If we consider tanks 13, 15, and 17 as part of the same set, then δ will give us essentially no information. If we throw them out, then the chains of non-δ-related elements are {16}, {14}, {12}, {2,3,5,9,10,11}, {4,6}, and {1,7,8}. The substitution choices are forced by the relation σ', except that we are free to choose either tank 7 or tank 8 in the final set, since the costs of the two tanks are identical and each is σ'-related to the other.

Using the above as an "initial approximation" to a solution, the following results were obtained by an exhaustive search which occupied less than a minute on an IBM AT:

Set	Representative	Cost
{1,7,8}	{7}	1082.07
{2,3,10,11}	{3}	13069.25
{4,6}	{6}	809.35
{5}	{5}	3709.45
{9}	{9}	2546.59
{12}	{12}	16115.92
{13}	{13}	3007.22
{14}	{14}	15851.90
{15}	{15}	2938.32
{16}	{16}	71216.99
{17}	{17}	13048.42

Minimum cost 143395.48

Notice that the above solution is a valid solution to the original problem, since it separates the pairs (4,10) and (12,14).

To further document the results of the above search, we present the following analysis of the partitioning of the largest set, i.e. the set {2,3,5,9,10,11}. The representative of each set in a given partition is underlined.

Number k of sets in the partition	Best partition into k subsets	Cost of partition
1	{{2,3,5,$\underline{9}$,10,11}}	23262.89
2	{{2,$\underline{3}$,10,11},{5,$\underline{9}$}}	19382.44
3	{{2,$\underline{3}$,10,11},{$\underline{5}$},{$\underline{9}$}}	19325.29
4	{{2,$\underline{3}$,10},{$\underline{5}$},{$\underline{9}$},{$\underline{11}$}}	19898.93
5	{{$\underline{2}$},{$\underline{3}$,10},{$\underline{5}$},{$\underline{9}$},{$\underline{11}$}}	20480.06
6	{{$\underline{2}$},{$\underline{3}$},{$\underline{5}$},{$\underline{9}$},{$\underline{10}$},{$\underline{11}$}}	21061.18

In each of the above examples we had mathematical proof that the solutions we obtained were correct. In the first example it was practical to find the solution by an exhaustive search strategy. In the second example we have reason to doubt that such a search strategy is practical. In that example, even after the initial partition (in which tanks 13, 15, and 17 were identified as forming a separate component), the solution obtained by an exhaustive search would have taken almost four hours even on our hypothetical million-potential-solutions-a-second computer. In an environment in which several sets of items need to be presented to a computer at a setting, this is unacceptable. What's more, it is likely that even larger sets of different items would need to be subjected to analysis. So the existence of mathematical properties that guarantee a minimal-cost solution to lie in a highly restricted area of the solution space is significant.

But even in situations where such mathematical laws are not operating we need some help. We need to guide our search for solutions more precisely, even if we may be steering toward a near-optimal rather than an optimal solution. Consider the following observation: the operation of finding and examining all immediate refinements of a given partition (i.e. all partitions obtained by splitting a single set of the original partition into two smaller subsets) is equivalent to the operation of finding all subsets of the set and thus has a complexity no worse than 2^n, where n is the number of elements of the set. Suppose we begin with a one-set partition, choose the immediate refinement of minimum cost, and restrict our search for solutions to refinements of this partition. Repeat the procedure until all immediate refinements result in an increase in cost. This is a natural, intuitive approach which may have some mathematical basis.

Another approach which may have even more merit because of its reduced complexity is to start with a partition consisting of n singleton subsets and successively join pairs of subsets until we can no longer reduce the cost with such a joining. The complexity of the search for the ´best´ joining is no worse than $C_{n,2}$. For this type of strategy, a relation called a <u>joiner</u> may be of some help. A joiner is a relation ß which has the property that when a ß b and a and b are the chosen representatives of respective sets K_1 and K_2 of a partition, then it is always more economical to join the two sets into one and use the item a as the chosen representative. For some examples of joiners for different cost functions, see (Yeager, 1987).

Notice in the last example that in breaking down the set {2,3,5,9,10,11} into finer and finer partitions and exhaustively searching for the most economical partition the six "best" partitions of orders 1, 2, 3, 4, 5, and 6 satisfied a very interesting property. The best partition of order k was always a refinement of the best partition of order k-1 and a ´joining´ of the partion of order k+1. To test whether this might always be the case, when no mathematical basis was discovered for the property, a large number of trials were performed using random data. In a large majority of cases this behavior was indeed present. However, several exceptions were noted.

CONCLUSIONS AND RECOMMENDATIONS

Figures 1-6 detail the recommended architecture for a collection of software tools to aid the commonality analysis process. Integration of these tools using a single, consistent user interface is also recommended. A menu structure is the simplest approach, but a natural language interface may be the best long-term solution.

We are assuming here a "loosely coupled" configuration, in which the actual creation and maintenance of the database is performed by the DBMS itself. Commands and data are passed to the DBMS from "front end" software modules and data is passed from the DBMS back to those modules for the performance of operations outside the capabilities of the DBMS.

The database creation module is illustrated in Figure 1. During database creation the database administrator makes a number of decisions which will seriously affect the usefulness of the database for the purpose of commonality analysis. Attributes, names of attributes, representation (character string or integer, for example), default values, and many other database configuration factors must be carefully chosen. Knowledge is an important component of the skills needed to create such a database. Some of that knowledge will be of such an ad hoc nature that it must reside with the database administrator himself. The knowledge needed to enforce consistency and uniformity across all commonality databases is of a less changeable nature, however, and could conceivably be encoded as rules which the database creation module would draw upon in its interaction with the human creator.

An essential ingredient of the database creation module is its synonym bank. The purpose of the synonym bank is to insure that different names are not being used in different commonality databases to refer to logically equivalent attributes. Each group of synonymous attribute names has a default representative to be used as the actual attribute name. The database administrator is informed of the substitution and is given the chance to over-ride for good cause.

Figure 2 details the software component used for entering of new data and new knowledge. As discussed above, knowledge concerning commonality relationships among the data is inherently associated with the data and should not be separated from it. When new data is entered, the database administrator should be prompted to review and perhaps modify the knowledge about commonality relationships in the data.

In Figure 3 we see the first stage of the commonality alternative selection module. Here the user requests a given

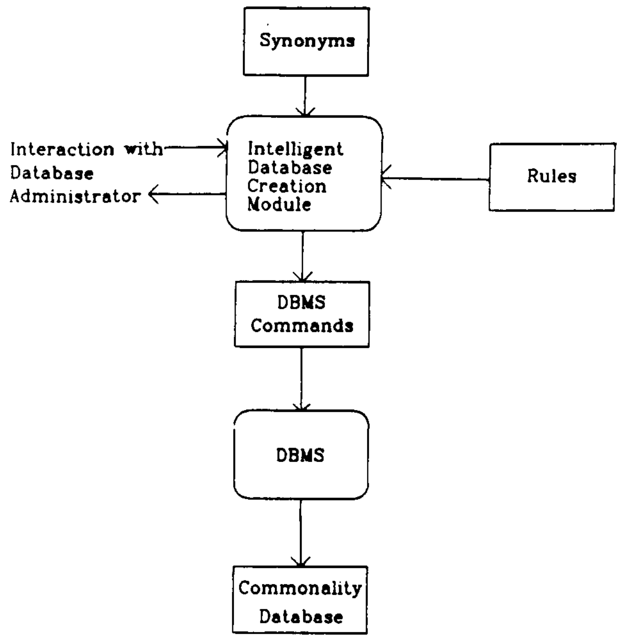

Synonyms
Interaction with Database Administrator
Intelligent Database Creation Module
Rules
DBMS Commands
DBMS
Commonality Database

Figure 1. Intelligent database creation module.

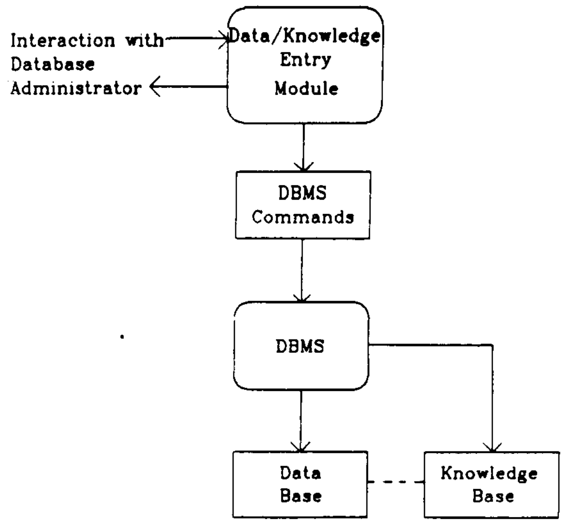

Interaction with Database Administrator
Data/Knowledge Entry Module
DBMS Commands
DBMS
Data Base
Knowledge Base

Figure 2. Data and Knowledge Entry Module.

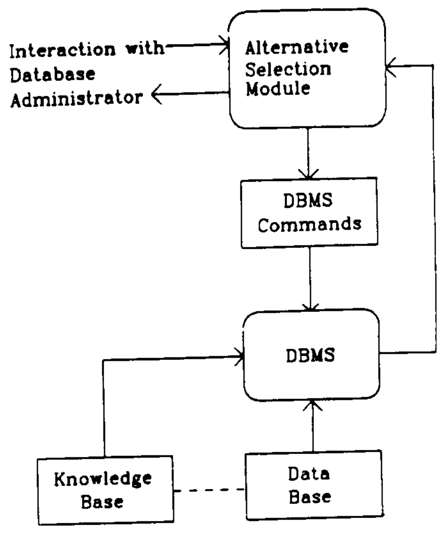


Figure 3. Commonality Alternative Selection.
Stage 1: Input.

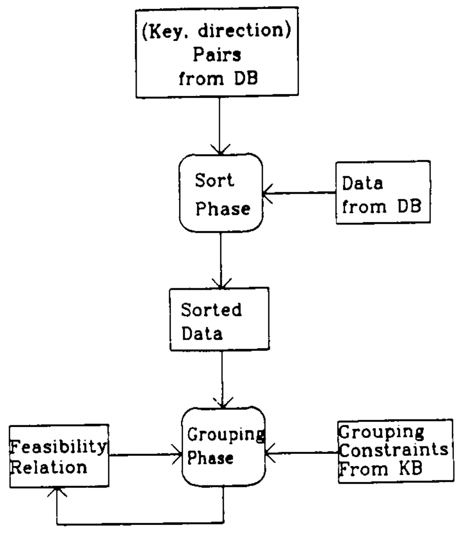


Figure 4. Commonality Alternative Selection
Stage 2. Construction of the
Feasibility Relation.

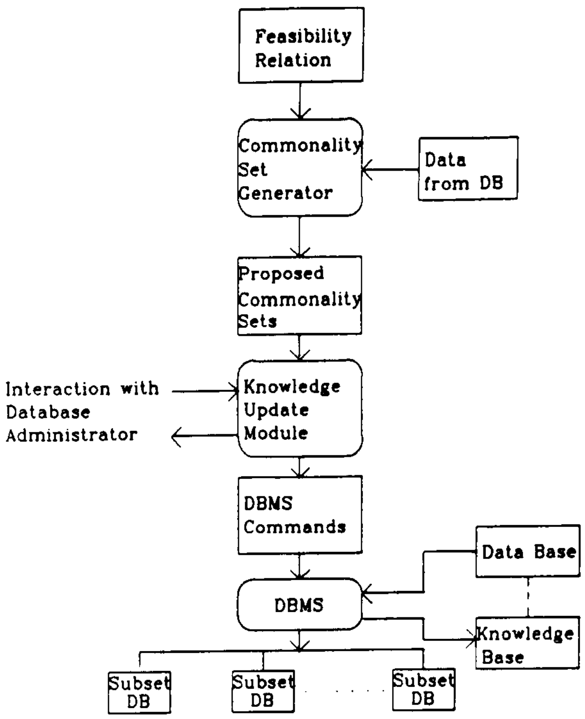


Figure 5. Commonality Alternative Selection. Stage 3. Generation, Approval, and Output of Subset data bases.

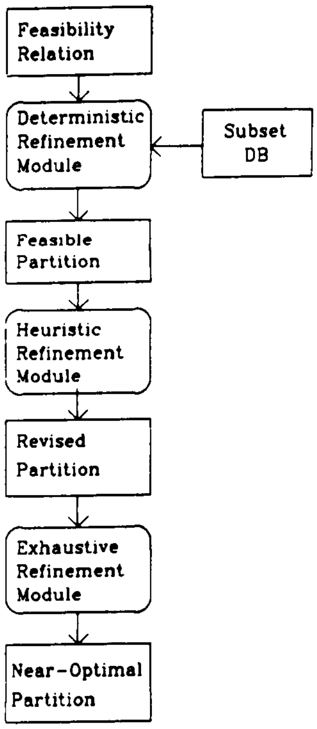

Feasibility Relation
Deterministic Refinement Module
Subset DB
Feasible Partition
Heuristic Refinement Module
Revised Partition
Exhaustive Refinement Module
Near-Optimal Partition

Figure 6. Commonality Analysis.

set of data from a commonality database, and that data is transferred to the front-end module along with the knowledge needed for constructing a feasibility relation on the data. Figure 4 depicts the data being alternately sorted and grouped using the information from the knowledge base. This sorting and grouping phase is applied iteratively, and the feasibility relation is extended in size with each such application. Finally, we see in Figure 5 the process in which commonality alternative sets are created using the feasibility relation and in which a subset database is generated for each set of alternatives. If α denotes the feasibility relation, then each set of alternatives is an α-connected component. What this means is that records a and b are in the same set if there exists a series of records $x_1, x_2, \ldots, x_k$ for which $a = x_1$, $b = x_k$, and for each $i = 1, 2, \ldots, k-1$, either $x_i \; \alpha \; x_{i+1}$ or $x_{i+1} \; \alpha \; x_i$. Each subset is presented to the database administrator or other user as it is generated. The user has the opportunity then to (a) accept the set of alternatives as a valid set, (b) discard the set of alternatives, or (c) accept a subset of the set presented. In the latter two cases, the system has reason to doubt the encoded knowledge which led to the generation of that particular set of alternatives. It is important at that point for the system to engage in a dialog with the user and attempt to update its knowledge base so that the same mistake does not recur.

The final module to discuss is the enhanced commonality analysis module, shown in Figure 6. Here a given subset of commonality alternatives is analyzed to find an optimal or near-optimal substitution strategy. An essential ingredient here is the feasibility relation generated in stage 2 of the commonality alternative selection process. The commonality analysis process breaks down into three sub-modules. The first utilizes deterministic mathematical rules which narrow the solution space as much as possible. The output from this sub-module is a feasible partition which may be the true minimum-cost solution or may be many refinements removed from the minimum-cost solution, depending on the properties of this particular feasibility relation and this particular set of data. If the feasible partition is not known to be optimal, then it is passed on to a submodule which attempts to produce a lower-cost partition using heuristic strategies. Finally, if it can be ascertained during the heuristic refinement process that the number of possibilities remaining to be checked is small enough that a complete pass through the entire set of remaining partitions can be reasonably undertaken, the sub-module for exhaustive search continues the refinement process to produce the final solution.

REFERENCES

COLLIER, D. A. 1978. A Product Structure Measure: the Degree of Commonality. Proc. Tenth National AIDS Meeting, St. Louis, Missouri.

COLLIER, D. A. 1980. Justifying Component Part Standardization. Proc. Twelfth National AIDS Conference, Las Vegas, Nevada.

COLLIER, D. A. 1981. The Measurement and Operating Benefits of Component Part Commonality. Decision Science 12, No. 1.

COLLIER, D. A. 1982. Aggregate Safety Stock Levels and Component Part Commonality. Management Science Vol. 28, No. 11.

MSFC. 1986. Commonality Analysis Program Review, October 9, 1986. Contract NAS8-36413, NASA George C. Marshall Space Flight Center, Alabama.

MSFC. 1987. Commonality Analysis Study, User Manual for the System Commonality Analysis Tool (SCAT), D483-10064, March 1987. Contract NAS8-36413, NASA George C. Marshall Space Flight Center, Alabama.

YEAGER, D. P. 1987. A Formulation of the Commonality Analysis Problem and Some Partial Solutions, submitted to Operations Research.