use as reading for Dominic 1. ***

[illegible]

DOMINIC I: PROGRESS TOWARDS DOMAIN INDEPENDENCE IN DESIGN BY ITERATIVE REDESIGN

J. R. Dixon
Department of Mechanical Engineering

A. Howe and P. R. Cohen
Department of Computer and Information Science

University of Massachusetts
Amherst, Massachusetts

M. K. Simmons
Knowledge-Based Systems Branch
General Electric Corporate Research and Development
Schenectady, New York

ABSTRACT

This paper describes the first working version of a program called Dominic that performs design by iterative redesign in a domain independent manner. The paper describes in detail the program's strategy, which stresses the concept of redesign dependencies to guide its redesign process.

Dominic has been succesfully tested in four different domains. Its performance on two of these (v-belt drive design and design of extruded heat sinks) is presented here.

The redesign class of design problems on which Dominic works is that large class of problems which are intellectually manageable and solvable without sub-division into smaller parts. This includes the various sub-problems ultimately created when large complex problems are decomposed for solution.

Dominic is a hill climbing algorithm, similar in this respect to standard optimization methods. However, its problem formulation or input language is more flexible for some design applications than optimization techniques.

Work is continuing on a Dominic II in an effort to overcome some of the limitations of Dominic.

1.0 INTRODUCTION

In our research we are modelling design as a process of repeated decomposition into subproblems until the subproblems are solvable as entities without further decomposition. These subproblems are then solved by a process of iterative redesign [1, 2, 3, 4, 5] as shown in Figure 1. This paper reports on progress made towards generalizing the redesign process. A program called Dominic is described which is able to do design by iterative redesign in several domains. Thus Dominic is, to a degree, domain independent. Though it works on only a subclass of redesign problems, and is not perfected, it represents progress in our efforts to learn how to construct more general programs for design.

Modelling the design process has been a perennial topic for discussion and research. Textbooks on design nearly always present a model of the process [6, 7, 8, 9]. Practicing designers have reflected on and written about the process [10, 11]. Artificial intelligence researchers have also examined design models [12, 13].

Modelling design is also an active area of current research. Brown and Chandrasakeran [14] describe design in terms of types from routine to innovative. They also present an initial attempt at a language for design knowledge based on a hierarchical organization of specialists, plans, tasks and steps [15]. Latombe [16] has developed a system called TROPIC which operates on a top-down problem solving model that increases the level of detail as the design proceeds. Other studies related to design process models include references [17, 18, 19, 20, 21, 22].

2.0 THE ITERATIVE REDESIGN MODEL

As noted above, iterative redesign is a model for design problem solving when the problem requires no further decomposition to be manageable intellectually. After problem specifications are obtained, an approximate or rough initial design is first synthesized, by copying a previous case, by rule of thumb, or by some default procedure. Though it is to be hoped (and is usually true) that the initial trial design so obtained will not be absurd, seldom will it be an acceptable final solution. (In those problem domains where research and/or experience have developed a procedure that can produce an acceptable design directly (i.e., without iteration) from the problem specifications, "design" is intellectually a dead issue and there is no need for additional application of intelligence, artificial or otherwise.)

Once an initial design has been created, its performance is simulated or predicted by analysis. (Correct analysis is a critical support needed for good design.) With the trial design's performance known, the acceptability of the design can be judged. If it is acceptable, the task is complete. If not, the design must be redesigned, the analysis for expected performance repeated, and so on iteratively as shown in Figure 1.

ASME CIE 1986

A great deal of design is actually done by this iterative redesign process. Moreover, the decomposition process that preceeds the redesign cycles is also iterative, consisting largely of iterative respecification and possibly also (though rarely) some iterative re-decomposition. The essence of the entire design process is iteration. Therefore, if we are to understand how to construct programs that design, we must understand how to control, guide, and generalize iterative design processes so that they converge efficiently to acceptable designs.

3.0 THE ARCHITECTURE OF DOMINIC

The architecture of Dominic is shown in Figure 2. Knowledge of a domain is obtained from a domain expert by a knowledge acquisition module. The information defining the specific problem to be solved is obtained from a user/designer. Together, the domain knowledge base and problem data base form the input to the inference engine.

Each of these modules is described in detail below. First, however, the structure by which design problems are formulated in Dominic must be described.

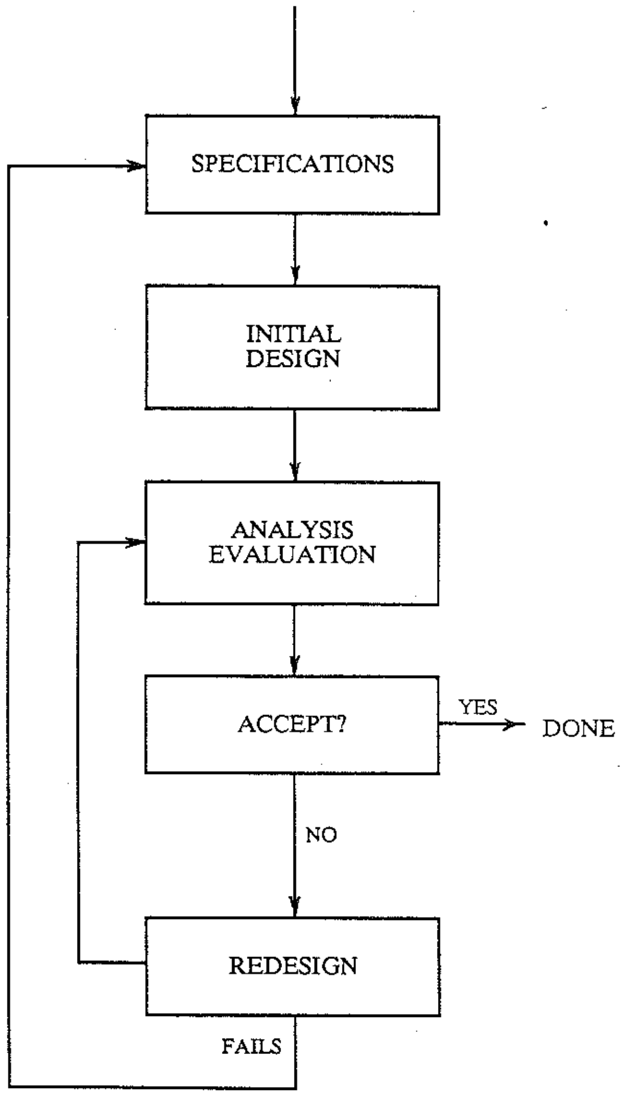


FIGURE 1 - The Redesign Model of Design

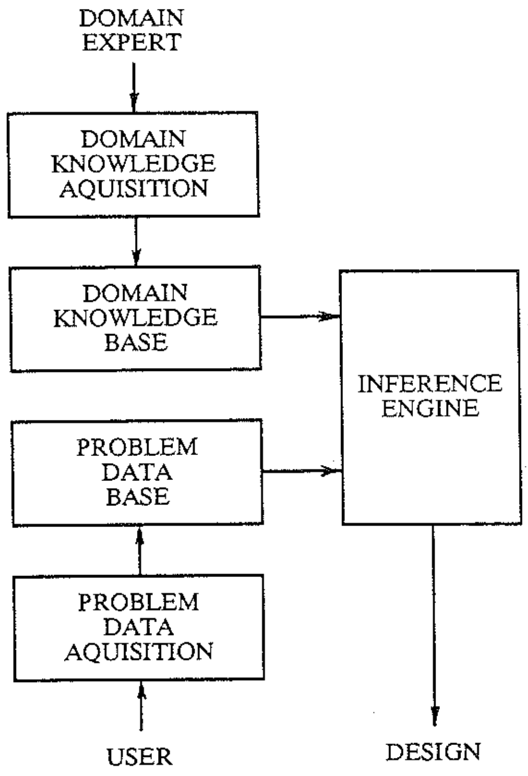


FIGURE 2 - Architecture of Dominic.

4.0 PROBLEM FORMULATION

Design problems for Dominic are formulated in terms of:

+ problem specification parameters,
+ design variables,
+ performance parameters,
+ an initial design procedure,
+ analyses,
+ dependencies,
+ dependency order list,
+ performance satisfaction scales,
+ constraint equations, and
+ constants.

Each of these terms are now described with respect to the architechture and operation of Dominic.

Problem specification parameters are those parameters whose values define a specific problem in a domain. For example, in the domain of standard v-belt drive design [3], the problem specifiers are horsepower to be transmitted, center-distance limits, drive speed, desired load speed, and desired life of the system. In the domain of rectangular extruded aluminum heat sinks [4], problem specification parameters include the overall size, the surface and ambient temperatures, the maximum fin height allowed, and the amount of heat to be transferred.

Design variables are those factors whose values the designer can select and control by design decisions. In the v-belt domain these are the pulley sizes, the belt type, the belt length, and the number of belts. In the heat sink domain these include the fin thickness, fin spacing, and fin height. The number of design variables determines the size of the design space that must be explored. Minimum and maximum limits for each design variable also must be specified in Dominic.

goals

Performance parameters are those factors by which the quality of a design is judged. Thus this is how the designer's goals are expressed in Dominic. In the v-belt domain, the performance parameters are the predicted values of load speed, life, belt velocity, lateral force on the shaft, and cost. In the heat sink domain, the performance parameters are the predicted values for heat transfer and cost. In Dominic, each performance parameter is also assigned a priority category of high, moderate, or low.

The initial design procedure is an optional part of a problem formulation in Dominic. This procedure must take as input the set of values for the problem specification parameters, and compute a full set of values for the design variables. It is not required that the initial design so generated be a feasible solution to the design problem; it is only a starting point. If an initial design procedure is not available, the program defaults to a trial initial design in which all design variables are at the midpoint of their maximum and minimum values.

An analysis is a procedure which, when given the set of problem specifications and a set of design variables, can compute values for one or more of the performance parameters. For example, an analysis procedure in the v-belt domain predicts the life of a v-belt system from data on horsepower, belt type, belt length, pulley sizes, etc. In the heat sink domain an analysis computes the heat transfer from the fin given the physical dimensions and temperatures.

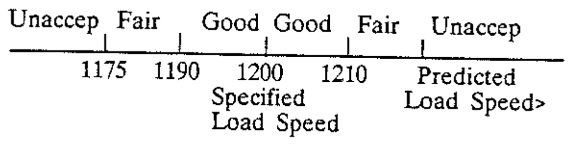
It is important to remember that analysis procedures produce only quantitative information about the goals; they do not make the judgements about what the quantitative information means. That is, the analyses do not by themselves determine whether the design is a good one, or what to do if it is not. Traditionally such judgements and decisions have been made by the human designer. Now we are trying to learn how to have them done by the computer program.

Dependencies are probably the most important problem formulation concept in Dominic. A dependency expresses a relationship between a performance parameter and a design variable. That is, dependencies describe how individual performance parameters depend upon the design variables. For example, in a v-belt system, the predicted system life (a performance parameter) depends very strongly on the number of belts (a design variable). In the heat sink domain, a dependency captures the nearly linear relationship between heat transfer and fin height.

The dependency order list is an ordered list of design variables associated with each performance parameter that are most likely to to effect a change in the performance parameter. In v-belt, for example, the dependency order list for load speed will be just load pulley diameter and drive pulley diameter since these are the best ways (in this simple case, the only ways) to effect the load speed performance. A dependency order list is not required by Dominic, but problem solving is enhanced when it is available. When available but wrong, it can cause serious problems [23].

Performance satisfaction scales indicate how well particular performance parameters are satisfied depending on their current predicted values. For example, in the v-belt domain, the satisfaction scale for load speed might be as follows (where the value of load speed is plotted along the horizontal line):

evaluation



Thus, if a trial v-belt design has a predicted load speed of 1180 rpm, that will indicate a load speed satisfaction of "fair". A speed of 1170 rpm would be categorized as "unsatisfactory". In a complete problem formulation, there are satisfaction scales for each of the performance parameters. The scales may include more discriminating categories than those shown above for evaluation, such as "excellent" or "poor". Finer scales than this, however, do not appear to be helpful.

A constraint equation in Dominic is an inviolable relationship among design variables or among design variables and problem specification parameters. In the v-belt problem, for example, there is a constraint equation which relates center distance to belt length and the pulley diameters. In the heat sink problem, fin width, fin spacing, and the number of fins are also constrained geometrically.

Finally, there may be constants of the problem domain such as physical constants (e.g., the acceleration of gravity) or material properties.

5.0 KNOWLEDGE ACQUISITION

The domain knowledge acquisition module in Dominic must obtain the necessary information from a domain expert to formulate the problem in the terms described in the preceeding section. Thus, the domain expert is asked to name problem specification parameters, the design variables, the performance parameters and their priorities, and to provide the data for satisfaction scales. The domain expert also must supply the required analysis procedures, constraint equations, and constant values. Finally, the domain expert provides opinions on dependency orders, if these are known.

Once the above information is available and stored in Dominic's internal format, the program is ready to solve specific problems in the domain. Using the problem data acquisition module, a user is prompted for values for the problem specifications parameters for the particular problem to be solved. The first user, of course, is the domain expert who must test the performance of the program to insure that it is designing as intended. Input parameters may have been overlooked, for example, or satisfaction scales may need adjusted. Since Dominic is still a research program, its interfaces with the domain expert and user are not as friendly as expected of finished software. Even so, only a few hours are required for a domain expert to input the required information.

There are no theoretical limits to the number of design variables, performance parameters, or problem specification parameters that can be handled by Dominic. We have not explored the practical limits, which will depend on computer memory and speed. It is expected that problems with as many as a dozen or so design variables and as many specification and performance parameters will present no difficulties.

6.0 INFERENCE ENGINE

contd.

The key to Dominic's operation is the redesign strategy. Currently, the program operates with a single redesign strategy. (On-going work is in progress to make a Dominic II in which several strategies are expressed explicitely, and to give the program the ability to dynamically select different strategies based on an evaluation of its own performance.) Faced with the need to perform a redesign, the following set of decisions constitues a strategy:

+ Which performance parameter(s) should be worked on?
+ By how much should the selected performance parameter(s) be modified?
+ Which design variable(s) should be changed to effect the desired change in the performance parameter(s)?
+ By how much should the selected variable(s) be changed?
+ Considering the possible impact on the other performance parameters, should the proposed change(s) be implemented?

There are many possible strategies. The one that Dominic uses is as described below.

First, it should be noted that Dominic works with only one performance parameter and one design variable at a time. This is a distinct limitation, but surely the way to begin; future versions will be more flexible.

To select which performance parameter to work on, the program uses a combination of performance parameter priorities and current degree of satisfaction of the performance parameters. The matrix shown in Figure 3 shows the order of selection. For example, if there is a high priority performance parameter that is currently in the unacceptable category, then that performance parameter will be worked on first as indicated by the number "1" in the matrix. Other combinations are selected in the order shown.

To determine by how much the selected performance parameter should be changed, Dominic computes the amount of change that will move the performance parameter into the next highest category of satisfaction. That is, if a performance parameter is selected that currently has a satisfaction of poor, then the program computes the minimum change from the satisfaction scale required to give that performance parameter a satisfaction of fair. This, it may be noted, is a rather conservative policy. A more aggressive strategy could, for example, propose a change that drives the performance parameter all the way to the highest satisfaction category. ✓

To select which design variable to change in order to effect the desired modification in the selected performance parameter, Dominic first tries to use the dependency order list. If the list contains untried design variables,the next variable on the list is selected. If none are there for use, then the program selects the design variable with the largest dependency value with the correct sign found in what we call the "dependency table."

The dependency table is a matrix relating each design variable to each performance parameter. (Figure 4 shows the status of the dependency table at the conclusion of a v-belt run. The values in the table are the dependencies, d.) Initial entries for the table are optionally obtained from the domain expert who may enter initial dependency information as a numerical value, as "high", "moderate", or "low", or as an equation. Dominic converts "high" to a value of 2, "moderate" to 1, and "low" to 0.5. The domain expert also designates whether the dependency is positive or negative.

		PRIORITY		
		High	Moderate	Low
	Excellent	-	-	-
CURRENT DEGREE OF SATISFACTION	Good	10	11	12
	Fair	6	7	9
	Poor	4 th	5	8
	Unacc	1 st	2 nd	3 rd

FIGURE 3 -- Table for Selecting Next Performance Parameter for Attention. (Example: The First Performance Selected Will Be Any High Priority One With an Unacceptable Satisfaction Level. Next Will Be Any Moderate Priority Performance Parameter With an Unacceptable Rating.)

DV

high = [illegible].0
moderate = [illegible].0
low = 0.5

DESIGN VARIABLES

PP			Drive Diam	Load Diam	Number Belts	Belt Length	
	PERFORMANCE PARAMETERS	Load Speed	1.0	-1.0	0	0	
		Lateral Force	0.59	0.080	0.375	0	
		Belt Velocity	1.0	0	0	0	
		System Life	.44	-0.74	3.89	1.04	values of d
		Cost	.166	0.313	0.869	0.52	

FIGURE 4 -- Example of a Dependency Table. The Values Shown Are From the Conclusion of a V-Belt Run.

Dependency values are used to relate performance parameters and design variables as shown here:

$$\overset{DV_s}{\left(\frac{\text{x-new}}{\text{x-old}}\right)} = \overset{PP_s}{\left(\frac{\text{y-new}}{\text{y-old}}\right)}^{d}$$

where	x-new	= new value for the selected design variable
	x-old	= old value of this design variable
	y-new	= desired new value for the selected performance parameter
	y-old	= old value of this performance parameter
	d	= dependency value.

Thus a dependency of d = +1 indicates a positive linear relationship, whereas a value of, say, -1.45 indicates a more strongly negative relationship. The form of this equation makes possible a relatively simple transformation from qualitative and intuitive judgements about dependencies in a domain to an intuitively satisfying numerical scale. The particular quantitative definition of dependency is otherwise arbitrary. We experimented with another, but found the one stated above to work better -- and reasonably well.

After each iteration in the course of a design, the dependency table is updated based on the actual values obtained from analysis of each trial case. In this way, values not obtained from the domain expert are filled in, and an increasingly more accurate set of values is maintained. Note, however, that these values indicate local information about dependencies in the neighborhood of the last design tried. The values are not globally valid except for special (simple) cases.

Once a design variable is selected (whether from the dependency order list or from the dependency table), since the performance parameter and the amount of its change have been previously determined, the amount to change the design variable is easily estimated using the above equation from the dependency value found currently in the table.

Next the program must decide whether or not to implement the proposed change, or to seek a different redesign. In case any proposed change of design variables violates a constraint equation, that change is not allowed. If constraints are satisfied, Dominic uses the current values in the dependency table to estimate new values for each of the performance parameters. The new level of satisfaction is then computed for each performance parameter. Next the program considers the overall impact on the design using a matrix as shown in Figure 5. For example, the overall evaluation of the design is III if (1) all high priority performance parameters are good or better, and (2) if all moderate priority performance parameters are good or better, and (3) if all low priority performance parameters are fair or better. Dominic has two ways to deal with the question of whether to implement or not. One is to allow only changes that do not decrease the overall design evaluation level. In the other, Dominic will implement changes that allow the overall design to drop by one level so long as this does not reduce the evaluation to unacceptable. That is, a proposed change is allowed if it is predicted to change the overall design evaluation from level III to level II. However, changes from III to I or from I to 0 (unacceptable) are not allowed. The decision of which policy to follow is now only under manual control of the user, but will be automated in Dominic II. The second method permits the program more flexibility to explore larger changes in the design variables. We are experimenting to learn the circumstances when each of these methods should be applied.

If a proposed change is accepted, the change is implemented and the new design analyzed again. If the change is rejected, then Dominic tries the next most influential design variable for the selected performance parameter. If all design variables are tried and no change can be implemented, then the program selects the next performance parameter, and starts again to look for a design variable to improve that performance parameter. If all performance parameters have been tried and no changes are possible, Dominic stops unless there are vacancies in the dependency table at this point.

There may be, and usually are, vacant entries in the dependency table at the outset since not all dependency information is known to domain experts. Usually these "holes" are filled by Dominic during the course of a design as explained above. However, if a hole persists to the end, Dominic will try to fill it before actually quitting for good. It does this by performing what may be viewed as an "experiment"; it implements an arbitrary change in the design variable where the vacancy exists. The subsequent analysis then fills the vacant spot, and the program can begin again.

These decision procedures constitute a redesign strategy for Dominic. There are many variations possible in this strategy, and we have experimented with some of them [23]. Current research is directed at enabling the program to change strategies as it runs depending on its own progress (or lack of progress).

PRIORITIES

		High	Moderate	Low
OVERALL RATING OF DESIGN	IV	=Exc	>Good	>Good
	III	>Good	>Fair	>Fair
	II	>Fair	>Fair	>Poor
	I	>Poor	>Poor	>Poor
	0	=Unacc	=Unacc	=Unacc

FIGURE 5 -- Table for Determining Overall Rating of a Design. (Example: To Achieve a Level III Rating, a Design Must Have All High Priority Performance Parameters Satisfied to Good or Better, and All Moderate Priority Performance Parameters Satisfied to Fair or Better, and All Low Priority Performance Parameters Satisfied to Fair or Better.)

The program must also decide when to stop. Currently it stops when an "excellent" design is achieved (i.e., when all performance parameters are satisfied to their highest possible level), or when it has explored all possible changes without being able to implement any, or when it has executed a preset limit on the number of iterations.

7.0 EXAMPLES

Dominic has been tried in four domains: v-belt drive design, extruded heat sink design, rectangular beam design, and a two-member hollow tube truss design. In all of these the program has produced acceptable designs, though not without some difficulties as described in Section 8.0 below. Since the beam and truss problems were very simple problems used by us only to test the "domain independence" of the program, only the v-belt and heat sink results are presented here.

Six different v-belt problems have been run on Dominic. Three typical case results are shown in Figure 6. The numbers in parentheses are values from the rule-based expert system for v-belt design called Vexpert [3]. The values in brackets are from a human expert.

Four heat sink problems have been run on Dominic. Two typical case results are shown in Figure 7. Values in parentheses are from the special purpose expert sytem for heat sink design called Xenif [4]. Figures in brackets are from a human expert.

The results indicate that Dominic is a reasonably capable designer in these domains (and in the two others tried) though the two domain-specific programs (Vexpert and Xenif) produced slightly superior performance in most cases. Compare the design costs, for example, in cases V-1, V-2, HS-1, and HS-2. Apparently the price paid for Dominic's generality is a very slight loss in quality of performance. Perhaps future improvements in Dominic will change this result.

In addition to these performance tests, a large number of experiments were run in which various changes were made in Dominic's redesign strategy [23]. In these experiments, the program seldom failed to produce some
satisfactory design. In general, the strategy described above was as good as or better than the variations.

Dominic is currently implemented in CommonLISP and runs on a VAX 11/780.

8.0 DIFFICULTIES ENCOUNTERED

Despite Dominic's general success, there are some difficulties. Perhaps the most significant from a research standpoint is Dominic's inability to deal with limitations on closely coupled design variables. An example of this is the limitation on the ratio of fin height to fin spacing in the heat sink problem. Manufacturing considerations constrain this ratio to be less than or equal to 4.0. If the design happens to evolve to a place where this ratio is very near the limit, this constraint becomes active. Then proposed increases to fin height or decreases to fin spacing get rejected because they violate the constraint. Since the program can change only one design variable at a time, it cannot make both change, keeping the ratio satisfied. Instead, it simply gets stuck, and does nothing. At present, when this happens, the program has no way to detect the situation or to break out of the corner it is in. ✓

A somewhat similar difficulty occurs occasionally in the v-belt domain when the program attempts to change one of the pulley diameters. Such a change, without a nearly similar change in the other pulley diameter will cause the load speed performance parameter to decrease dramatically. If the proposed change causes the satisfaction of load speed to go to unacceptable, then the change is rejected. To keep the load speed satisfaction above unacceptable, only a very small change in one pulley size is permitted, and the program begins to creep along making only the tiny progress allowed in this situation. Again the program cannot detect or cure this difficulty at the present time. It may be noted that the fact that the program can change only one design variable at a time is also largely ✓ responsible for this difficulty.

Another difficulty with Dominic is its inability to deal with discrete design variables. This occurs on two levels. The first, rather easy to correct, is illustrated by the fractional number of belts that Dominic recommends in the cases shown in Figure 8. More important, however, is the current inability of the program to consider large single moves in the design space. Dominic has no way to obtain or to use domain knowledge that would, say, direct the design to use gears instead of v-belts.

Current work on Dominic II is directed towards resolving these difficulties.

9.0 SUMMARY AND CONCLUSION

Though the program has some difficulties as described, Dominic generally works to produce acceptable designs for problems that fit into the redesign class.

Dominic is essentially a hill climbing algorithm. It is neither a very "strong" method (i.e., one that is powerful because it has a knowledge rich strategy in a very narrow range of applicability), nor is it a very "weak" method (i.e., one that is very general and thus widely applicable). It falls somewhere in-between: it works generally, but only on that subclass of design problems we are calling the redesign class.

As a hill climbing algorithm, Dominic is similar to standard optimization methods. The major difference between Dominic and optimization methods lies in the language of the problem formulations. Optimization techniques require the user to formulate problems in terms of a quantitative, mathematical objective functions, whereas Dominic's formulation is more akin to a very low level design language; that is, Dominic accepts inputs in terms somewhat closer to, or more natural to, the physical design problem or domain.

The redesign class of problems on which Dominic works includes a great many mechanical design problems, including many sub-problems formed by decomposition of more complex problems. As a advance in applying knowledge-based systems to design, Dominic's relative success appears to justify optimism that many common design problems can ultimately be understood sufficiently well as iterative processes so that useful general problem solving algorithms for design can be written.

ACKNOWLEDGEMENT

This work has been supported in part by a grant from General Electric Company to the University of Massachusetts.

REFERENCES

[1] Dixon, J. R., and Simmons, M. K., "Expert Systems for Design: A Program of Research, ASME Paper No. 85-DET-78, presented at the Design Engineering Conference, Cincinnati, OH, September 10-13, 1985.

[2] Dixon, J. R., Simmons, M. K., and Cohen, P. R., "An Architecture for Applying Artificial Intelligence to Design", Proceedings 21st ACM/IEEE Design Automation Conference, Albuquerque, NM, June 25-27, 1984.

[3] Dixon, J. R., and Simmons, M. K., "Expert Systems for Design: Standard V-Belt Drive Design as an Example fo the Design-Evaluate-Redesign Architecture", Las Vegas, Nevada, August 12-16, 1984.

[4] Kulkarni, V. M., Dixon, J. R., Sunderland, J. E., and Simmons, M. K., "Expert Systems for Design: The Design of Heat Fins as an Example of Conflicting Sub-goals and the Use of Dependencies", Proceedings ASME Computers in Engineering Conference, Boston, MA, August 4-8, 1985.

[5] Dixon, J. R., Libardi, E. C., Luby, S. C., Nielsen, E., and Jones, C. D., "Knowledge Representation in Design: Issues and Examples", SAE Conference, Detroit MI, February 24-27, 1986.

		EXAMPLE NUMBER		
		V-1	V-2	V-3
PROBLEM PARAMETERS	Drive Speed (rpm)	3450	1800	1500
	Load Speed (rpm)	2000	1200	900
	Horsepower	10	40	400
	Ctr Dist Min (in)	12	12	12
	Ctr Dist Max (in)	24	28	32
	Life Spec (hours)	4000	4000	8000
RECOMMENDED DESIGN	Belt Type	3v (3v) [3v]	5v (5v) [5v]	8v (8v) [8v]
	Load Pulley (in)	9.9 (4.5) [10.6]	11.8 (9.0) [14.0]	17.6 (22.4) [30]
	Drive Pulley (in)	5.8 (2.7) [6.0]	7.7 (5.9) [9.3]	10.5 (13.2) [18]
	Belt Length (in)	53 (38) [53]	85 (50) [80]	100 (100) [140]
	Number of Belts	.99 (3) [1]	2.7 (4) [2]	15.9 (15) [12]
PERFORMANCE PARAMETERS	Cost ($)	51 (37) [53]	153 (133) [148]	1587 (1629) [1834]
	Life (hours)	4000 (7343) [6722]	5232 (5360) [4535]	8000 (8436) [4586]
	Load Spd (rpm)	1998 (2031) [1952]	1176 (1180) [1189]	913 (883) [900]
	Belt Vel (fpm)	5203 (2393) [5419]	3632 (2780) [4559]	4108 (5183) [7068]
	Shaft Force (lb)	149 (242) [147]	847 (897) [657]	8691 (9336) [10761]

Dominic
Domain specific E.S.
Human Expert

FIGURE 6 - Examples from Dominic For V-Belt Drive Domain. Values in Parentheses are from the Domain Specific Expert System Vexpert [3]. Values in Brackets are From a Human Expert.

		EXAMPLE NUMBER	
		HS-1	HS-2
PROBLEM PARAMETERS	Heat Min (w)	30	70
	Heat Spec (w)	50	90
	Max Height (m)	0.05	0.05
	Length (m)	0.1	0.1
	Width (m)	0.1	0.1
RECOMMENDED DESIGN	Fin Height (m)	.0380 (.0491) [.0352]	.035 (.050) [.050]
	Fin Thick (m)	.001 (.002) [.001]	.0010 (.0017) [.00178]
	Spacing (m)	.009 (.007) [.0123]	.0087 (.0125) [.0125]
PERFORMANCE PARAMETERS	Heat Tx (w)	51 (50) [50]	83.2 (85.2) [85.2]
	Cost ($)	1.10 (.95) [1.10]	2.22 (2.09) [2.05]

FIGURE 7 -- Examples from Dominic for the Heat Sink Domain. Values in Parentheses are From the Domain Specific Program Xenif [4]. Values in Brackets are From a Human Expert.

[6] Gibson, J. E., Introduction to Engineering Design, Holt, Rinehart, and Winston, NY, 1968.

[7] Krick, E. V., An Introduction to Engineering and Engineering Design, Wiley, NY, 1967.

[8] Dixon, J. R., Design Engineering: Inventiveness, Analysis, and Decision Making, McGraw-Hill, NY, 1966.

[9] Asimow, M., Introduction to Design, Prentice-Hall, Englewood Cliffs, NJ, 1962.

[10] Clegg, G. L., The Design of Design, Cambridge Engineering Series, Cambridge University Press, Cambridge, England, 1971.

[11] Marples, D. L., "The Decisions of Engineering Design", IRE Transactions on Engineering Management, June, 1961, pp 55-70.

[12] Simon, H. A., The Sciences of the Artificial, MIT Press, Cambridge, MA., 1967.

[13] Mostow, J., "Towards Better Models of the Design Process", AI Magazine, 6:1, pp 44-46.

[14] Brown, D. C. and Chandrasakeran, B., "An Approach to Expert Systems For Mechanical Design", Proceedings Trends and Applications, IEEE Computer Society, May 1983, NBS, Gaithersburg, MD., pp 173-180.

[15] Brown, D. C., and Chandrasakeran, B., "Expert Systems for a Class of Mechanical Design Activity", Proceedings of IFIP WG5.2 Working Conference on Knowledge Engineering in Computer Aided Design, Budapest, Hungary, Sept., 1984.

[16] Latombe, J. C., "Artificial Intelligence in Computer-Aided-Design: the TROPIC System", Tech Report 125, Stanford Research Institute, February, 1976.

[17] Brown, D. C., "Capturing Mechanical Design Knowledge", Proceedings of the 1985 ASME Computers in Engineering Conference, Boston, MA., August, 1985.

[18] Brown, D. C., "Failure Handling in a Design Expert System", CAD Journal, November, 1985.

[19] Brown, D. C., and Chandrasakeran, B., "Plan Selection in Design Problem Solving", Proceedings of the AISB-85 Conference, Warwick, England, April, 1985.

[20] Mittal, S., Dym, C. L., and Morjoria, M., "PRIDE - An Expert System for the Design of Paper Handling Systems", Applications of Knowledge-Based Systems to Engineering Analysis and Design, ASME WAM, Miami, Florida, November, 1985.

[21] Muster, D., and Mistree, F., "A Curriculum and Paradigms for the Science of Design", Proceedings 1985 ASEE Annual Conference.

[22] Burrow, L. D., "The 'Design to Product' Alvey Demonstrator", Int. Conf. on the Development of Flexible Automation Systems, Publ. No. 237, July, 1984.

[23] Howe, A., Dixon, J. R., Cohen, P. R., and Simmons, M. K., "Dominic: A Domain Independent Program for Mechanical Engineering Design", Proceedings Applications of AI to Engineering Problems, Southampton, England, April, 1986.