ISHED1: Applying the LEM Methodology to Heat Exchanger Design

Kenneth A. Kaufman
Ryszard S. Michalski

MLI 00-2

ISHED1: APPLYING THE LEM METHODOLOGY TO HEAT EXCHANGER DESIGN

Kenneth A. Kaufman
Ryszard S. Michalski

Machine Learning and Inference Laboratory
School of Information Technology and Engineering
George Mason University
Fairfax, Virginia 22030-4444
{kaufman, michalsk}@mli.gmu.edu

P 00-2
MLI 00-2

January 2000

ISHED1: APPLYING THE LEM METHODOLOGY TO HEAT EXCHANGER DESIGN

Abstract

Evolutionary computation has traditionally employed a Darwinian approach, in which a population of individuals "evolves" based on random perturbations in their "genetic" makeup and the probablistic survival of the fitter. A new paradigm, the *Learnable Evolution Model* (LEM), integrates a symbolic learning component to evolutionary computation; it seeks out rules explaining the differences between the better and worse performers in the population, and generates new individuals based on the templates specified in these rules. The LEM methodology has proven able to improve on the efficiency of the evolutionary process

This paper describes the ISHED1 system, which applies the LEM methodology to a problem of engineering design. Specifically, it searches for the best arrangement of the evaporator tubes in the heat exchanger of an air conditioning system, a problem with a search space too large for exhaustive methods. This problem did not involve a trivial application of either the Darwinian or the symbolic evolutionary components due to the real-world constraints on heat exchanger design; as a result, ISHED1 employs a version of LEM customized for this problem.

The results of initial experiments have been promising; ISHED1 has been able to discover heat exchanger structures comparable in performance to the current state of the art. However, these industry models assume a uniform air flow across the heat exchanger and perform poorly under different airflow conditions. By testing the performance of individual structures under the given environmental conditions, ISHED1 automatically adapts to the problem at hand, and selects architectures better suited to the defined real-world situation.

Keywords: Evolutionary Computation, Multistrategy Learning, Engineering Design, Learnable Evolution Model

ACKNOWLEDGMENTS

The authors thank the National Institue of Standards and Technology and Intelligent Information Systems, Inc. for their support on this project. This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's activities are supported in part by the National Science Foundation under grants IIS-9904078 and IRI-9510644, in part by the Defense Advanced Research Projects Agency under Grant No. F49620-95-1-0642 administered by the Air Force Office of Scientific Research, and in part by the Office of Naval Research under grant N00014-91-J-1351.

1 INTRODUCTION

This paper describes the application of state-of-the-art ideas and methods of machine learning and evolutionary computation to the development of an intelligent system for heat exchanger design, ISHED1. This system employs a methodology based on the Learnable Evolution Model (LEM), which integrates Darwinian and symbolic methods of evolution (Michalski, 2000).

The Darwinian module is based on traditional genetic methods (e.g., Holland, 1975), in which a population of individuals goes through a cycle of generate-and-test iterations, with generation being a two-step process of selection (probabilistically based on the results of the test phase, and perturbation, in which various types of changes are applied to the selected individuals). The strength of such strategies is their ability to perform a pseudo-directed search in large event spaces; a major disadvantage is their slowness to converge.

Symbolic learning attempts to make the evolutionary process more efficient by investigating the differences between those individuals that perform well and those that perform poorly, to encapsulate that knowledge in rules, and to apply those rules in generating individuals to comprise the next population.

ISHED1 is a customized application of LEM, tailored to a a problem of engineering design. Specifically, it searches for the best arrangement of the evaporator tubes in the heat exchanger of an air conditioning system, a problem whose search space is too large for exhaustive methods. Furthermore, the air conditioning systems currently available rely on certain assumptions about the conditions under which it will be working (temperature, humidity, airflow, etc.) Since ISHED1 relies on a simulator that takes these factors into account (Domanski, 1989) to evaluate potential heat exchanger architectures, its populations of trial individuals will evolve toward structures suited for the given conditions.

This problem did not involve a trivial application of either the Darwinian or the symbolic evolutionary components due to the real-world constraints on heat exchanger design; as a result, ISHED1 employs a version of LEM specially customized for this problem. In the next section, we describe the heat exchanger optimization problem in further detail. Sections 3-5 describe the ISHED1 system, followed by a description of experiments and results.

2 PROBLEM STATEMENT

ISHED1 conducts an evolutionary optimization process that seeks a heat exchanger that provides the maximum heat transfer for the given size of the exchanger and the assumed environmental conditions and other factors. These factors include the outside air temperature and humidity, the flow of air through the heat exchanger, the number of rows of tubes and the number of tubes per row in the exchanger, the refrigerant used, etc.

In an air conditioning unit, the refrigerant flows through a loop. It is superheated and placed in contact with cooler outside air (the condenser unit), where it transfers heat out and liquifies. Coming back to the evaporator, it comes into contact with the warmer interior air that is being pushed through the heat exchanger, as a result cooling the ear and heating and evaporating the refrigerant.

The heat exchanger itself consists of an array of parallel tubes through which the coolant flows back and forth. A typical model is shown in Figure 1. In this figure, there are three rows of 16 tubes, with the tubes (numbered 1-48 here) being viewed roughly end-on in the figure. The figure depicts an architecture in which the coolant enters the heat exchanger at tube 24, proceeds through *tubing bends* to tube 40, and then 39, and so on, until it reaches tube 25, at which point the coolant path splits. One branch continues from tube 10 to tube 16, and the other from tube 42 to tube 1. While this figure illustrates a coolant path with only one inlet tube (24), multiple inlets are permissable.

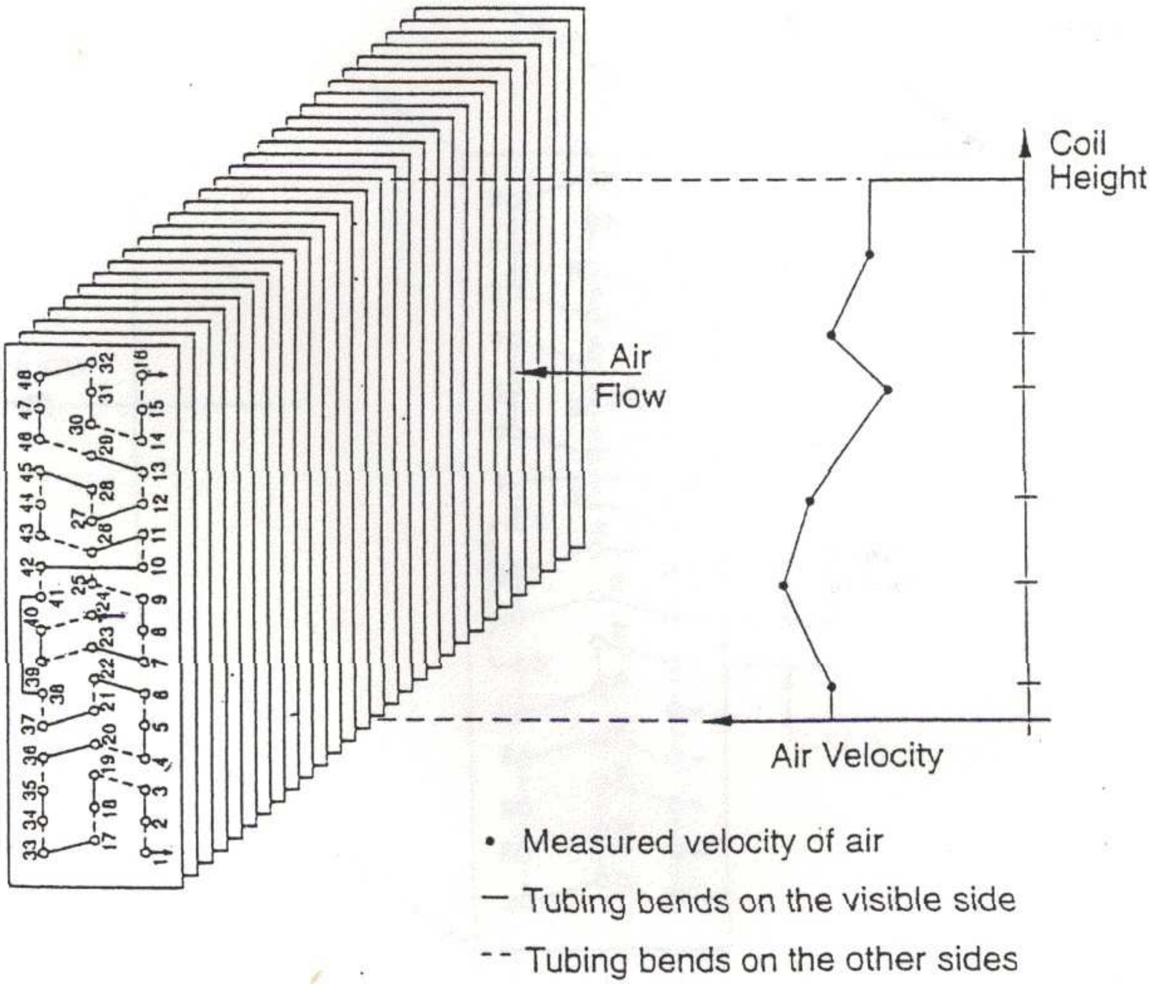

Figure 1: An Architecture of Evaporator Circuitry: A sample16 x 3 configuration.

Meanwhile, air is forced through the unit, with a velocity/volume profile as described in the figure. The air first comes into contact with and is cooled by the refrigerant in the first depth row (tubes 1-16), then the second, then the third row.

The amount of heat transfer cooling the air conditioner will provide will be the aggregate of the heat transfer provided by each of its tubes. These terms will be a function of the temperature and volume per unit time of both the air and the refrigerant coming into contact at that tube. Different orderings of the tubes will change the characteristics of the refrigerant passing through each tube, and the results of prior air/refrigerant interactions will affect their temperatures at later interactions. Other factors also affect later interactions. For instance, the refrigerant will lose pressure (and velocity) while passing through the bends between tubes; it will thus help in maximizing heat transfer if adjoining tubes are physically close to each other.

In short, the goal of the ISHED project is to determine how to order the flow through the tubes such that heat transfer is maximized given a set of environmental conditions. Note that the number of depth rows and the number of tubes per row are mutable, and ISHED1 can handle different heat exchanger sizes so long as there are equal numbers of tubes in each row, the number of depth rows does not exceed 5, and the total number of tubes does not exceed 130.

ISHED1 is able to apply background knowledge based on the nature of the problem in order to constrain the search to plausible architectures. A user-defined parameter (or its default value imposes limitations on the lengths of most tube bends. In addition, the program enforces six constraints on architectures, listed in increasing order of stringency in Table 1. The program rejects structures that violate a required constraint, only generates structures from scratch that adhere to constraints numbered (by importance level) 2 and up, and only under special circumstances (namely when designing a more coherent architecture is very difficult) generates structures from scratch that violate constraint 1.

Constraint Description	**Importance Level**
Predecessor to exit tube should be adjacent to exit tube. *Suggested.*	1
The exit tubes should be in first depth row. *Advisory.*	2
Inlet and exit tubes should not be adjacent to one another. *Strongly advised*.	3
The path from an inlet tube should split at most once. *Required*.	4
All exit tubes should be on the same side of the manifold as the inlet tube (i.e., all refrigerant paths should consist of an even number of tubes). *Strong requirement.*	5
All tubes must receive their refrigerant from another tube or the inlet tube (i.e., no self-looping). *Necessary condition.*	6

Table 1: Constraints implemented in ISHED1.

Constraints 1 and 3 allude to the fact that while through most of its travels through the heat exchanger the refrigerant is a mixture of liquid and gaseous refrigerant, and thus at a temperature close to its evaporation point at its current pressure, the refrigerant in the exit tube is all gas, and as such warmed rapidly to a higher temperature by the exchange of heat (as opposed to it being used in a phase shift). There is some noticible conduction of heat between the exit tubes and their immediate neighbors; this is minimized when the refrigerant in those neighboring tubes is also close to leaving the heat exchanger system. Similarly, the second constraint is based on the fact that the overall cooling will be most effective when this warmest refrigerant encounters some of the warmest air.

The fourth constraint is based on the unacceptable drops in refrigerant pressure that will occur if a single path undergoes multiple splits. The fifth is based on the structural requirements of the air conditioning unit, and the sixth should be obvious.

3 OVERVIEW OF ISHED1

The purpose of ISHED1 is to apply advanced machine learning and evolutionary computation technologies to assist an expert in designing optimized heat exchanger architectures for given operating conditions. ISHED1 works in conjunction with two other major systems, a simulator to estimate the performance of a given heat exchanger architecture (Domanski, 1989) and a general purpose AQ-type inductive learning system (Michalski, 1983), which is employed in the Symbolic Learning Module of ISHED1.

The ISHED approach to this problem applies the LEM evolutionary computation methodology (Michalski, 2000). Specifically, the approach integrates two strategies: *Darwinian evolutionary learning* and *symbolic learning,* explained below. Figure 2 presents a general diagram of the implemented ISHED1 system. Darwinian evolutionary learning is performed by the Evolutionary Learning Module, and symbolic learning is performed by the Symbolic Learning Module.

The Control Module takes the current population and determines which evolutionary strategy to pass it to. The selected strategy operates on the population, generates the subsequent population, and passes it to the simulator for evaluation of the individual structures. These structures and their evaluations are returned to the Control Module for the next generation (iteration).

Two related parameters guide the Control Module in determining which strategy to apply. Essentially, ISHED1 applies Darwinian evolution until the population is no longer improving. It then switches to symbolic learning until similarly the performance (both by the best individual and the population overall) has plateaued, continuing to alternate modes based on performance.

Because of the nature of the problem and the feasible ways of internally representing heat exchanger structures, both evolutionary modules required problem-specific customization. Traditional genetic operators would be, for the most part, unworkable in the Darwinian evolution module in this domain, so eight analagous domain-specific *structure modifying* (SM) operators were implemented, such as swapping the positions of two adjacent tubes in the flow, or moving the source of a tube’s refrigerant further upstream, in the process creating a split path. The SM operators change the characteristics of the candidate exchangers in ways that most likely lead to *admissible* new structures, that is, structures satisfying the given physical and environmental constraints, as described above. A selected operator is tried repeatedly with different operands in order to generate a feasible structure, until it either succeeds or “times out” (based on control parameters), in which case another operator, hopefully more applicable, will be tried.

The second strategy, based on symbolic learning, examines the characteristics of both well- and poorly-performing designs, and automatically creates hypotheses (in the form of attributional rules) that characterize the well-performing architectures. These hypotheses are then applied to generate a new population of designs.

Due to the complexity of the domain, the learner is presented with an abstract, rather than precise, specification of the different structures. Hence, the rules produced are also in these abstract terms, and as such may not in themselves provide information about the problem that is usable by domain experts. Conversely, the abstraction increases the generality of the generated rules, making it possible for more ways of instantiating them.

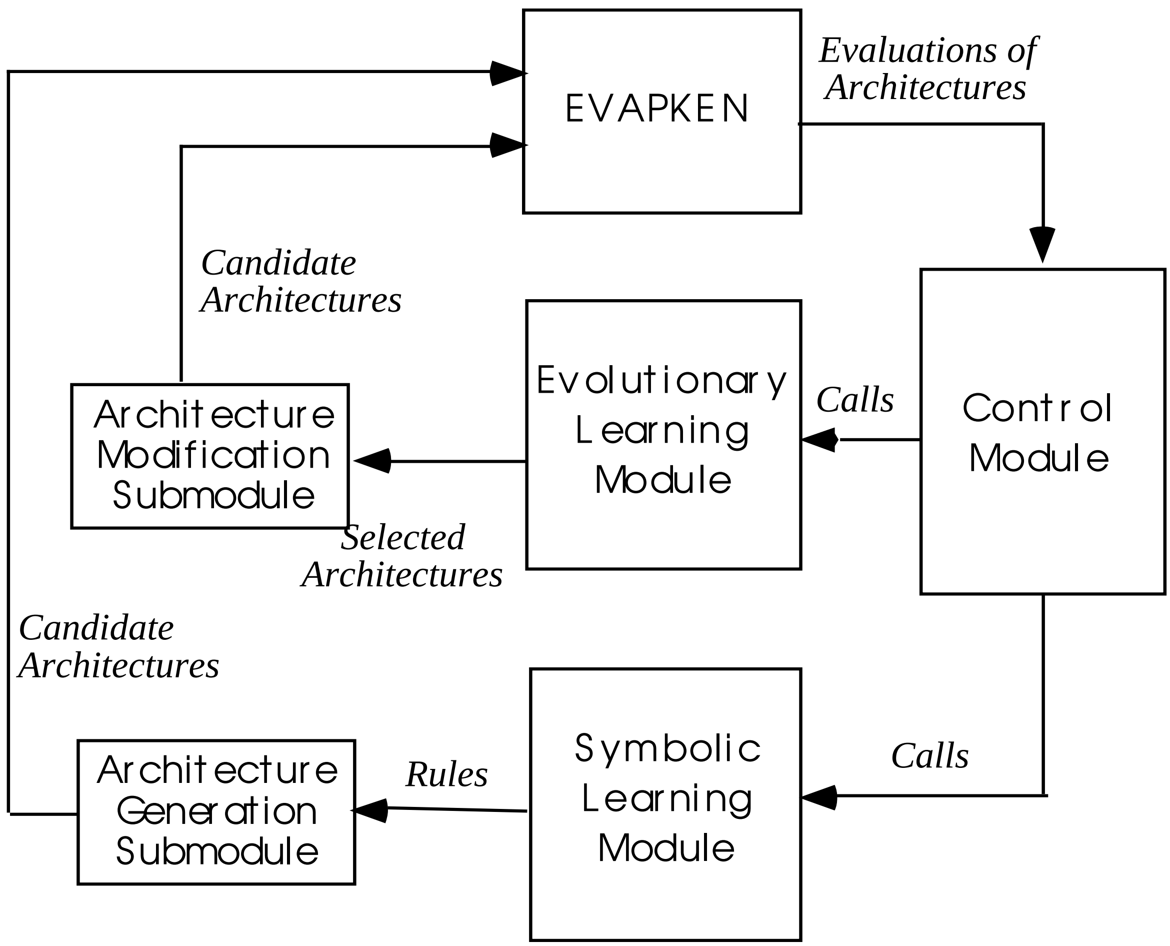

Figure 2: A general functional architecture of ISHED1.

There is a limitation, however to the instantiation process in this initial implementation of the Symbolic Learning module. The process of generating from a specification of inlet, outlet and split tubes a heat exchanger structure that satisfies the constraints described above is not trivial. Hence, ISHED1 in most cases deterministically generates only one architecture given a specification. Thus, the symbolic learning step as currently implemented more consolidates knowledge than encourages population diversity.

To summarize, given instructions characterizing the environment for the sought heat exchanger design, an initial population of designs (either specified by the user, randomly generated, or a combination of the two), and parameters for the evolutionary process, ISHED1 evolves populations of designs using the above two strategies for a specified number of generations, and returns a report that includes the best designs (architectures) found and their estimated capacity, as determined by the simulator. A control module determines when to apply each of the two evolutionary strategies (see the diagram in Figure 2).

4 SYSTEM OPERATION

4.1 Control Parameters

The first step of the ISHED1 function is to read a file that defines the control parameters for the program and characteristics of the desired architecture. These parameters, which override defaults when read, allow users latitude in controlling the system run. These parameters can be grouped into the following clusters (some of the individual parameters are alluded to in multiple clusters):

- The characteristics of a heat exchanger: its size and its shape
- The characteristics of the initial population: its size, any user-specified individuals, and the nature of the randomly specified individuals by the system
- The length of the evolutionary process
- The airflow through the heat exchanger, defined by a list of locations in the cross-section of the exchanger, and the velocities of the air at these. The velocities in other locations are linearly interpolated.
- Run control parameters, including the persistence of the Darwinian and symbolic learning modes, parameters for guiding Darwinian mode operation and symbolic mode architecture generation, and the level of detail to be presented in the output file.

4.2 Defining the Initial Set of Structures

ISHED1 allows a user to define an initial set of architectures of the heat exchanger. If the user does not define such a set, the system generates the initial set randomly. The user may define one or more initial architectures, and specify the number of copies of each architecture to be generated to fill the initial population. The user-specified architectures may be based either on previous ISHED runs, or draw upon the user’s knowledge of the problem, such as to test hypotheses or to try to improve upon industry models.

It is also possible that the user defines only a portion of the initial population, in which case the system randomly generates the remaining individuals. The random generation process creates different types of architectures in proportions defined a priori in the program. These proportions, determined through estimations of the form promising architecctures are likely to take, are listed in Table 2.

This table indirectly represents the requirement that all inlet-outlet paths must pass through an even number of tubes. For this to be the case, all heat exchangers with an odd total number of tubes must have at least one split path (since otherwise, at least one path would have odd length). Hence, their outlets must outnumber their inlets.

Number of Inlets	Number of Outlets	Percentages of Initially Generated Population					
		Odd number of tubes			Even Number of tubes		
		up to 49	51-85	87+	up to 50	52-84	86+
1	1	0%	0%	0%	10%	0%	0%
1	2	30%	10%	0%	20%	10%	0%
2	2	0%	0%	0%	20%	20%	0%
2	3	30%	20%	20%	20%	10%	15%
3	4	30%	20%	20%	20%	10%	15%
3	3	0%	0%	0%	10%	20%	20%
3	4	10%	17%	20%	0%	10%	20%
3	5	0%	16%	20%	0%	10%	15%
3	6	0%	17%	20%	0%	10%	15%

Table 2: Distribution of architecture types in initial population generation for different exchanger sizes.

4.3 System Control

The control module begins running in Darwinian evolution mode using an elitist strategy (i.e., the best performing architecture so far *always* gets passed to the next generation; it will be the first element of the upcoming population). Elitism also is used in Symbolic learning mode; here, the best architecture so far *and* all architectures that were in the "good" class for rule learning propagate directly to the next generation. Elitism has proven to be an important feature of evolutionary computation processes and is used in many algorithms.

In Darwinian mode, the structure modifying operator selected for application to a given structure is selected probabilistically, based on the topology of the heat exchanger structure (the number of inlets, outlets and splits). The probabilities are based on an estimate of how likely an application of this operator is likely to result in a favorable change. Typically, the less catastrophic operators and those that will maintain architectures with three inlets or fewer are favored.

4.4 Darwinian Evolution Module

To guide the process of modification of structures in the evolutionary design process in harmony with the design constraints imposed on heat exchangers, we developed and implemented eight Structure Modifying (SM) Operators for the ISHED1 system:

1) SPLIT (Figure 3): Creates two paths from a single path, starting from a given split point. Select a split point and a break point on the same path randomly, under the constraint that the break point is after the split point in the flow of the coolant. Change the source of the tube following the break point to the split point. The tube preceding the break point now becomes an outlet tube, unless it was already a split point, in which case it will have one fewer split. Check integrity.

Notation: SPLIT(SP, BP)

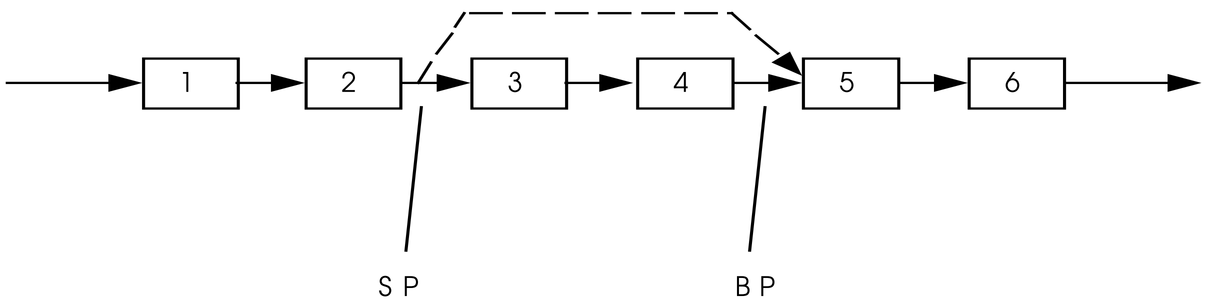

Figure 3: Application of the SPLIT Operator: SPLIT(2, 5)

2) BREAK (Figure 4): Creates two full paths from the input to the output. Select a break point randomly. Change the source of the tube following the break to the heat exchanger's input port. As in (1), the tube preceding the break point now becomes an outlet tube, unless it was already a split point, in which case it will have one fewer split. Check integrity.

Notation: BREAK(BP)

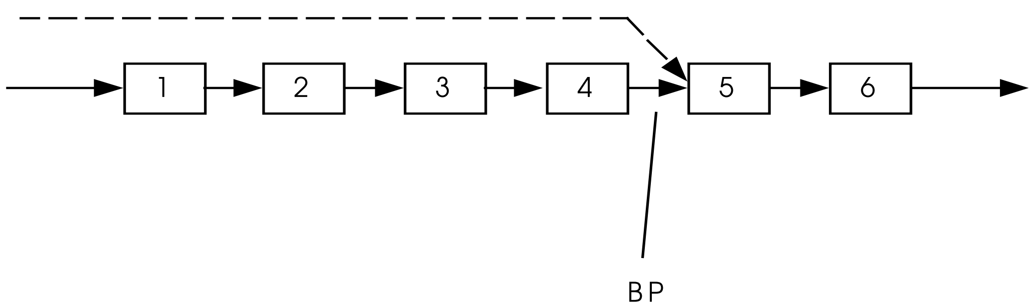

Figure 4: Application of the BREAK Operator: BREAK(5)

3) COMBINE (Figure 5): Takes two paths, and creates a single split path from them, which has a single inlet and two outlets. Select a split point randomly. Give a tube whose source was the input port and from which coolant did not flow through the selected split point another source, namely the selected split point. Check integrity.

Notation: COMBINE(SP, PATH)

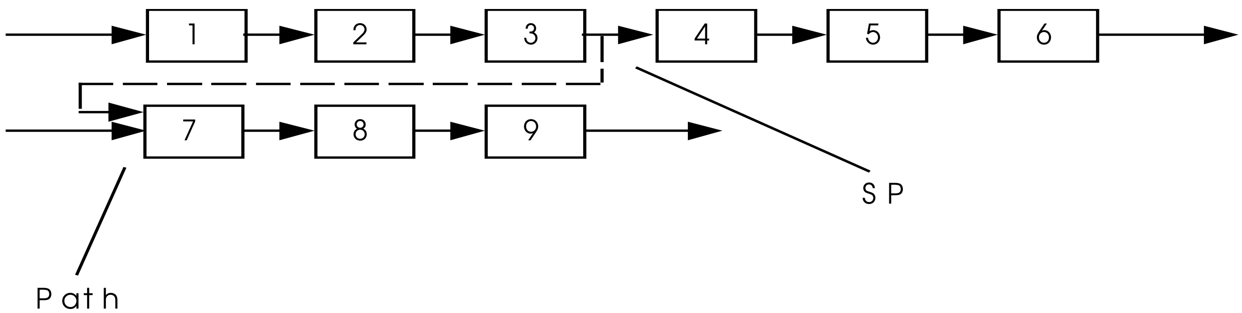

Figure 5: Application of the COMBINE Operator: COMBINE(7, 3)

4) INSERT (Figure 6): Takes two paths, and incorporates one into the other at some break point. Select a break point and a tube whose source is the heat exchanger's input port randomly such that the selected inlet tube's coolant does not flow through the selected break point. Move the source of the selected inlet tube to the tube following the break point, and the source of the tube that had followed the break point to what had been an outlet tube fed by the selected inlet tube. Check integrity.

   Notation: INSERT(BP, PATH)

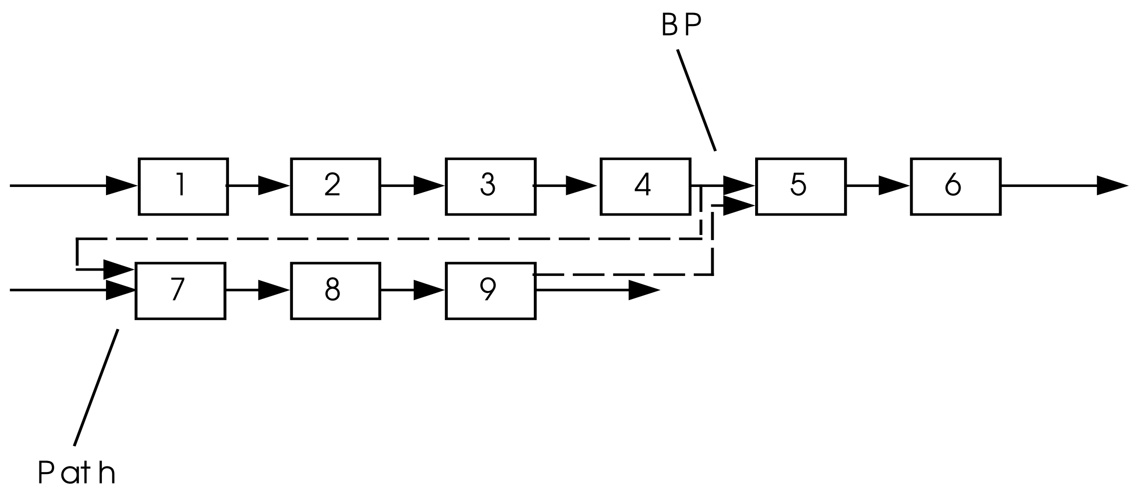

Figure 6: Application of the INSERT Operator: INSERT(7, 5)

5) MOVE-SPLIT (Figure 7). Move a split point an even number of tubes upstream or down one of its paths (the even number is necessary to keep the even path length constraint (importance level 4)). The distance denotes the number of tubes to move forward or backward. If it is positive, the move is forward (downstream), and if negative, the move is backward (upstream). Check integrity.

   Notation: MOVE-SPLIT(SP, Distance)

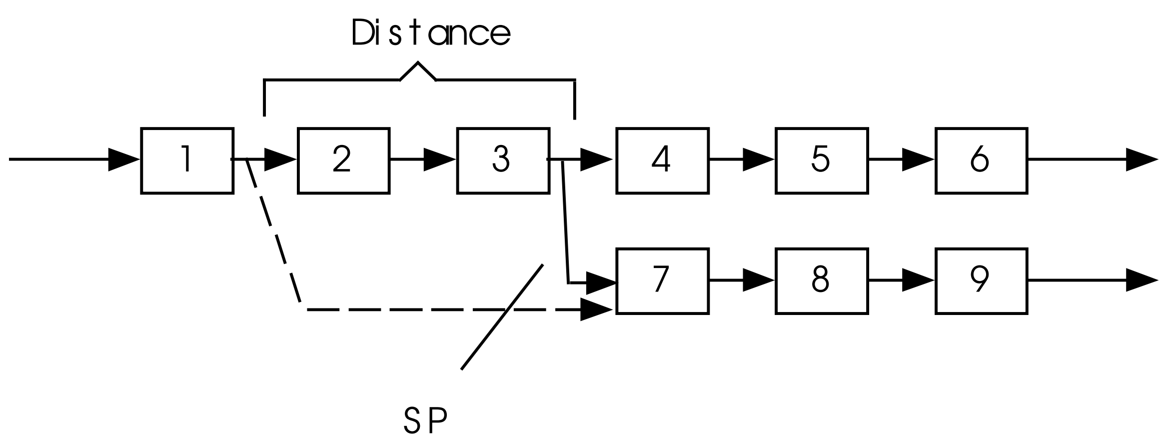

Figure 7: Application of the MOVE_SPLIT Operator: MOVE_SPLIT(7, -2)

6) SWAP (Figure 8): Reverse the order of two adjacent tubes in a flow structure. Select a random break point, at least two tubes away from its inlet. Swap the two tubes preceding the break point. Note that if either swapped tube has "siblings" due to a split, the siblings are transferred to the other swapped tube. Check integrity.

Notation: SWAP(BP)

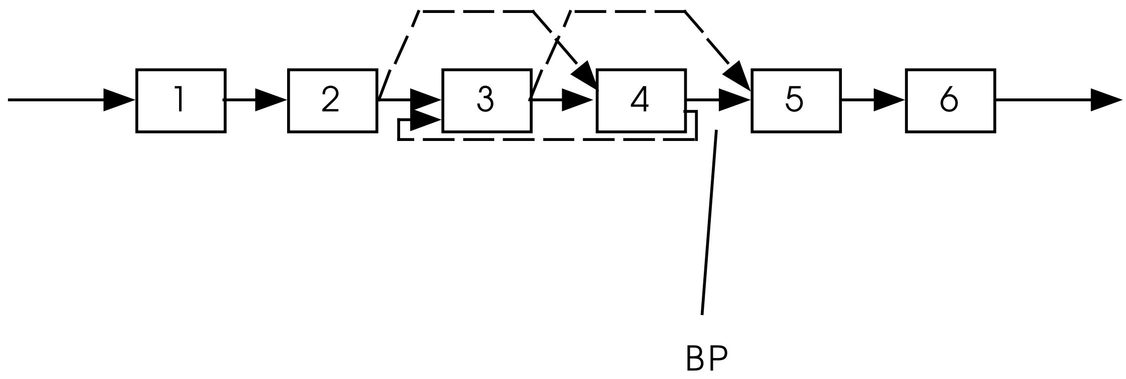

Figure 8: Application of the SWAP Operator: SWAP(5)

7) INTERCROSS (Figure 9): Swap the sources of two tubes in a flow structure that are not upstream of one another. Select randomly two break points, and make the switch. Check integrity.

Notation: INTERCROSS(BP1, BP2)

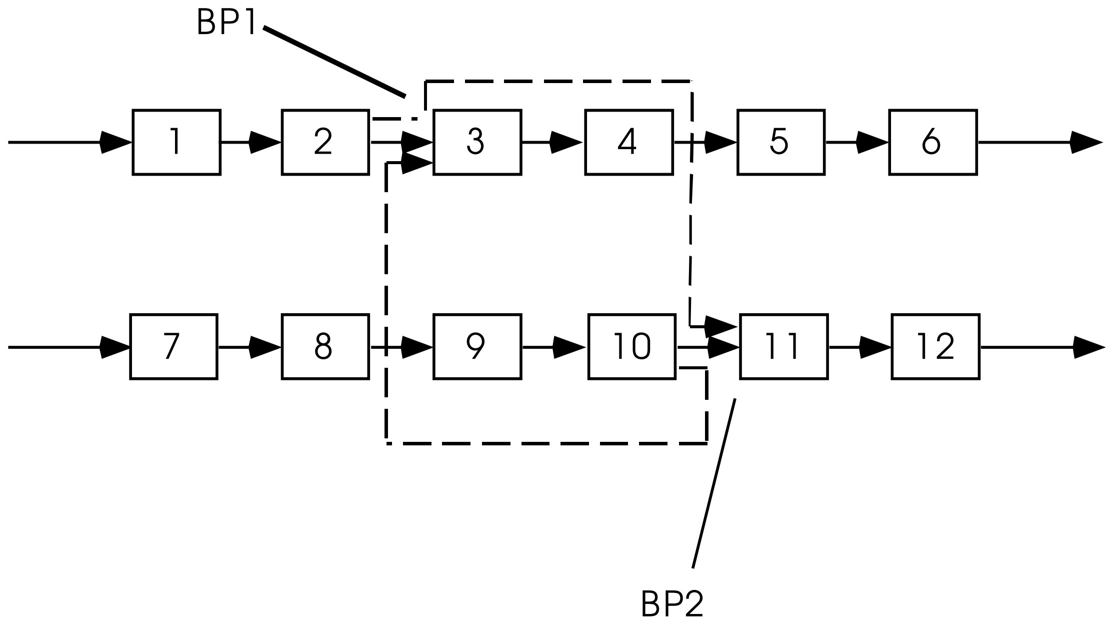

Figure 9: Application of the INTERCROSS Operator: INTERCROSS(3, 11)

8) NEW-SOURCE: Create a new source for a randomly selected tube, by changing the source of the tube at the break point to the tube designated as the attachment point. Note that to avoid loops, the attachment point can not be downstream of the break point. Check this and other integrity constraints.

   Notation: NEW-SOURCE(BP, AP)

4.5 Symbolic Learning Module

When the Symbolic learning module is applied, the old population is divided into three classes based on their cooling capacity. If all members of the population have identical performance, this mode is possible, and Darwinian evolution will take place instead.

The range from best performance level in the population to the worst is examined. Individuals with performance in the top 25% of this range are placed in the "good" class. Similarly, individuals in the bottom 25% are placed in the "bad" class. Any other individuals are placed in the "indifferent" class.

The learning program generates one or more rules to distinguish "good" architectures from "bad". It should be noted that during consecutive generations in this mode, rules will be in the context of their predecessors, so as to further focus the concept of optimality. As stated above, the next generation will consist in part of the best architecture discovered so far, and all the other members of the "good" class. The rest will be generated by applying the learned rules; if there are enough open slots in the population, each rule will be applied at least twice, by rotating among them, ordered by the number of training examples satisfying each rule (called t-weights -- an indicator of the rule’s applicability). Finally, any other individuals will be generated based on rules chosen probabilistically, based on their t-weights.

5 PROGRAM OVERVIEW

This section describes the organization of the ISHED1 program itself, so as to serve as a reference for future developers. The program is written in ANSI-C, and is distributed among four files.

The two smallest files are ISHED.C and ISHED.H. The former is simply a skeleton file that loads and integrates the other source files; the latter defines the data structures and global variables.

File TUBEGEN.C contains the routines that create heat exchanger designs based on a given set of inlet and outlet tubes. Split paths are not generated in these functions; if a split is required, it will be incorporated elsewhere, either by changing the source tube of the head of an independent path from the inlet manifold to an interior tube in another path, or by cutting an inlet-outlet path at an interior point, and attaching the former lower section to a higher point on the upper section.

The fourth file, IMAIN.C contains the majority of the functions and the flow control for the program. The remainder of this section provides an overview of the program flow through this file.

The first major duty of the program is to call out to function get_params, and read in the parameters file. The variables corresponding to the individual User Control parameters are initialized at their default values, and whenever a line in the parameters file indicates a parameter to set, the corresponding value is overwritten. Once all parameters have been read, the size and shape of the heat exchanger will be known to the program; ISHED1 can now allocate size-dependent data structures and write what will become the header section of the input files to the MEL rule learner (specifically, the definition of the attributes from which MEL will generate rules; in ISHED1, one attribute is defined per tube in the heat exchanger).

The next step is to generate the initial population. ISHED1 reads an initial population file if one has been specified, and makes the corresponding instantiations. If more individuals need to be incorporated into the population, the program calculates how many to make with various numbers of inlets and outlets, randomly selects the appropriate number of inlet and outlet points, and then calls upon the appropriate TUBEGEN routine to generate ordered strands of tubes, before adding any necessary splits as described above.

The program then enters its main loop: evaluate the population, determine whether to use Darwinian or symbolic evolution for the next generation, and create the new population. The evaluation step is straightforward. Each individual is passed to the simulator, and its discovered capacity is read and stored in an array. New individuals are tested to see if their capacity is amont the top five discovered so far; if so, the arrays maintaining the top five structures and their capacities are updated accordingly..

The program begins in Darwinian mode and does not switch modes unless the population quality has not improved for a number of generations specified by the appropriate PROBE parameter. The only exception occurs if a symbolic learning step is called for, but all members of the population have identical capacities, so that it is not possible to differentiate between better- and worse-performing designs. In this case, ISHED1 immediately reverts to Darwinian mode.

The Darwinian new population creation procedure is twofold. In the first phase, a set of base individuals are selected. The first member of the new population is based on the best design discovered so far during the run. Every other member of the new population, is based on a randomly selected individual in the most recent population; an individual's probability of selection is proportional to its heat exchanger capacity.

During the second phase of Darwinian population creation, the program attempts to apply one of the eight Structure Modifying operators to each of the base individuals in the new population. The operator to try is selected probabilistically, based on the number of inlets and outlets in the design of the individual being operated upon. If, after a number of attempts specified by a control parameter, the application of the operator has failed to generate a feasible architecture according to the constraints listed in Table 1, the program will once again select an SM operator to try. ISHED1 is equipped with a time-out feature, so that in the unlikely event of continued failures to find a feasible design after hundreds or thousands of operator applications, the program will instead apply a "NO-OP" and move on to the next individual. Once operators have been applied to all individuals, the new population has been determined.

The Symbolic new population creation procedure is twofold as well. ISHED1 first examines the capacities of each of the members of the population, putting those individuals in the HIGH group if their capacities are in the top 25% of the range between the highest and lowest capacity in the population, in the LOW group if their capacities are in the bottom 25% of that range, and in the INDIFFERENT group otherwise. The characteristics of the members of the HIGH and LOW groups (in terms of whether each tube in the associated heat exchanger is an inlet tube, an outlet tube, a tube above a split point, or none of those) are passed to the MEL rule learning program, which determines rules that distinguish the HIGH group from the LOW group.

During the second phase of Symbolic population creation, the population itself is built. The first individual will be a copy of the best design found so far during the ISHED run, and every other member of the HIGH group will also be included in the new population. The rest of the population is determined by instantiating the newly learned rules. First the program decides which rule characterizing the HIGH group to instantiate as follows: If there exists a rule that has not yet been instantiated twice, instantiate the rule that has been instantiated the fewest times; if more than one such rule exist, select the one satisfied by the most members of the HIGH group. If each rule has been instantiated at least twice, select a rule probabilistically, with each rule's probability of selection proportional to the number of HIGH individuals that satisfy it.

A rule is instantiated as follows: For each tube in the exchanger, it is determined which of the {inlet, outlet, split, regular} roles the tube may take on according to the rule. Each tube is then randomly assigned one of its permissible roles in such a way that the number of outlets is equal to the number of inlets plus the number of splits, and the number of inlets is at least half the number of outlets. A TUBEGEN.C function is then called to create a heat exchanger model consisting of paths from each of the inlet and split tubes to each of the outlets. Finally, each split tube is grafted into an inlet-outlet path, to complete the instantiation process.

After the prescribed number of generations, ISHED1 outputs to the log file the five best designs it has found, in two formats for easy future use. Figure 10 shows a very brief excerpt from an ISHED1 run log, with interjected comments in italics. The initial designs evolved to better designs (as measured by the capacity).

6 SUMMARY OF EXPERIMENTS

During the course of ISHED1 development, many experiments with the system were conducted. The initial experiments concentrated on a well-known problem, using a common heat exchanger size and a fairly uniform airflow pattern. ISHED designs provided results comparable to the industry standard. One concern in some of the ISHED-generated designs was that after many generations of Darwinian evolution, designs would become chaotic in terms of their inter-tube connections. Nonetheless, using available tools, an engineer could smooth some of the connections, hopefully at little or no cost to the estimated capacity of the exchanger.

In later experiments, the refrigerant was changed, and the airflow pattern was defined as highly non-uniform. Under such conditions, industry-standard heat exchangers do not perform well. The best ISHED-produced architectures conformed intuitively to expectations of what a successful architecture in a non-uniform airflow should look like, and indeed performed far better than the currently-used expert-designed structures.

Subsequent experiments varied the size and shape of the heat exchanger -- between 2 and 4 depth rows, with between 40 and 90 total tubes. Similar results were observed. Later, we began experimenting with pre-specified members of initial populations. The results were to some degree mixed. When a very large portion of the initial population was pre-specified with known good architectures, further improvement could often be found. To some degree, the pre-specification is analagous to an initial symbolic learning step using the prior background knowledge; as a result, ISHED begins with a solid population.

```
Exchanger Size: 16 x 3
Population Size: 15   Generations: 40
Operator Persistence: 5
Mode Persistence: GA-probe=2 SL-probe=1

Initial population:
Structure #0.3:  17 1 2 3 4 5 6 7 8 9 12 13 29 15 31 I 18 33 20 36 22 38 24 40 26 42
       11 2 7 45 14 47 16 34 35 19 37 21 39 23 41 25 43 44 28 46 30 48 32:  5.5376
Structure #0.8:  17 1 20 3 4 22 6 24 8 26 10 28 27 15 16 32 33 2 18 19 5 38 7 40 9
       42 11 44 13 46 30 48 34 35 36 I 21 37 23 39 25 41 27 43 29 45 31 47:
       Capacity = 5.2099
and 13 others

Selected Members:  3, 2, 3, 7, 9, 3, 9, 3, 6, 9, 9, 6, 8, 1, 7
Operations: NS(23, 39), SWAP(8), SWAP(28), SWAP(19), SWAP(1), SWAP(27),
            SWAP(40), SWAP(43), SWAP(15), SWAP(25), SWAP(7), SWAP(36),
            SWAP(29), SWAP(25), SWAP(1)
```

Below is one of the structures created by the application of a SM operator in Machine Learning algorithm mode (by swapping the two tubes following tube 29 in Structure #0.8)

```
Generation 1:

Structure #1.13: 17 1 20 3 4 22 6 24 8 26 10 28 27 15 16 32 33 2 18 19 5 38 7 40 9
      42 11 4 13 45 30 48 34 35 36 I 21 37 23 39 25 41 27 43 46 29 31 47:
      Capacity=5.2093
(15 structures in a population)

Selected Members:  6, 15, 11, 3, 13, 1, 10, 6, 12, 10, 5, 4, 13, 1, 3
. . . . . .
Generation 5: Learning mode
Learned rule:
      [x1.x2.x3.x4.x5.x6.x7.x8.x9.x11.x12.x13.x14.x15.x17.x18.x19.x20.x21.x22.x23.x2
      4.x25.
      x26.x27.x28.x29.x30.x31.x32.x33.x34.x35.x36.x37.x38.x39.x40.x41.x42.x43.x44.x4
      5.x46.
      x47.x48=regular] & [x10=outlet] & [x16=inlet]        (t:7, u:7, q:1)

An example of a generated structure:
Structure #5.1:  17 1 2 3 4 5 6 7 8 9 12 29 45 30 31 I 18 33 20 36 22 38 24 40 26 42
      11 27 13 15 47 48 34 35 19 37 21 39 23 41 25 43 44 28 46 14 32 16:
      Capacity=5.5377
.........
```

Below are examples of structures from the 21st generation:

```
Generation 21: Machine Learning mode

Structure #21.7: 18 1 4 2 6 3 5 7 8 9 12 13 45 15 31 I 33 17 35 36 22 39 24 40 42 25
      11 44 30 46 32 47 34 19 20 37 21 23 44 41 26 43 28 27 29 14 48 16:  4.1702
Structure #21.15 2 18 4 1 6 3 5 7 8 9 12 13 45 15 31 I 33 17 35 36 22 39 24 40 42 25
      11 44 30 46 32 47 34 19 20 37 21 23 38 41 26 43 28 27 29 14 48 16:  5.5387
and 13 others

Selected Members:  11, 4, 4, 13, 15, 10, 12, 13, 15, 15, 12, 2, 3, 5, 10.
.........
```

Finally, ISHED1 achieves:

```
Generation 40:

Structure #40.15: 33 17 2 41 4 5 6 9 7 8 12 29 46 45 47 I 1 34 20 36 22 38 24 3 42
      43 44 27 13 15 32 16 18 11 19 37 21 32 23 25 40 26 28 35 30 14 48 31:
      Capacity=6.3686
```

Figure 10: Excerpts from the log of an ISHED1 run

But when fewer individuals were used to seed the initial population, improvement was hard to come by. While further experimentation is needed to determine if this is a regular occurrence, and if so its cause, it is possible that a level of imbalance is reached in the population that hinders both the establishment of large numbers of seeded examples and their kin for improvement, and the blossoming of promising, but relatively weak, randomly generated individuals. It is also possible that system parameters need to be adjusted for such a scenario.

The experiments during all stages of this project served to confirm the ability of ISHED1 to generate improved designs. Thus, it has proven to be a powerful tool for automating the heat exchanger design processes.

7 CONCLUSION

This report presented a description of the ISHED1 system, its design, its implementation, and an experimental run. An accompanying User’s Guide describes in detail how to use the system.

The experiments with ISHED1 demonstrated that it is capable of significantly improving the initially defined designs. As commented by Dr. Domanski, some of the preliminary designs generated by ISHED1 match the best human expert designs, or may be superior to them, in particular, in cases of non-uniform air flow. ISHED1 thus provides a powerful and unique tool for designing superior heat exchanger structures.

REFERENCES

Domanski, P.A., "EVSIM - An Evaporator Simulation Model Accounting for Refrigerant and One Dimensional Air Distribution," NISTIR 89-4133, 1989.

Holland, J., *Adaptation in Artificial and Natural Systems*, Ann Arbor: The University of Michigan Press, 1975.

Michalski, R.S., "A Theory and Methodology of Inductive Learning," in Michalski, R.S. Carbonell, J. and Mitchell, T. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, TIOGA Publishing Co., Palo Alto, pp. 83-134, 1983.

Michalski, R.S., "LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning," *Machine Learning*(38), pp. 9-40, 2000.