International Journal of Intelligent Systems
Volume 21, Issue 12, 2006

INTELLIGENT EVOLUTIONARY DESIGN:
A New Approach to Optimizing Complex Engineering Systems and its Application to Designing Heat Exchangers

Ryszard S. Michalski* and Kenneth A. Kaufman

Machine Learning and Inference Laboratory

George Mason University

Fairfax, VA, 22030

*Also with the Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract

A new method for optimizing complex engineering designs is presented that is based on the *Learnable Evolution Model* (LEM), a recently developed form of non-Darwinian evolutionary computation. Unlike conventional Darwinian-type methods that execute an unguided evolutionary process, the proposed method, called LEMd, guides the evolutionary design process using a combination of two methods, one involving computational intelligence and the other involving encoded expert knowledge. Specifically, LEMd integrates two modes of operation, *Learning Mode* and *Probing* Mode. Learning Mode applies a machine learning program to create new designs through *hypothesis generation* and *instantiation*, while Probing Mode creates them by applying expert-suggested *design modification operators* tailored to the specific design problem.

The LEMd method has been used to implement two initial systems, ISHED1 and ISCOD1, specialized for the optimization of evaporators and condensers in cooling systems, respectively. The designs produced by these systems matched or exceeded in performance the best designs developed by human experts. These promising results and the generality of the presented method suggest that LEMd offers a powerful new tool for optimizing complex engineering systems.

Keywords: engineering design optimization, evolutionary computation, function optimization, learnable evolution model, genetic algorithms, machine learning, intelligent evolution, computational intelligence.

1 Introduction

Conventional methods of evolutionary computation employed in complex engineering design have been inspired by a Darwinian model of evolution in which new individuals (designs in this case) are created by random mutations and/or recombinations, and are selected for the new parent population through natural selection, according to the principle of the survival of the fittest. In contrast, the recently introduced Learnable Evolution Model (LEM), while adopting the principle of the survival of the fittest, introduces the novel idea of creating new individuals under the guidance of computational intelligence, specifically, through an inferential process of hypothesis generation and instantiation[1,2]. In this process, a machine learning program is used to generate a hypothesis describing differences between the highest and lowest performing individuals (in the case of engineering design, between highest and lowest quality designs). Such a hypothesis demarcates areas in the search space that likely contain promising designs. By sampling these areas through instantiating the hypothesis in different ways, new, promising designs can be obtained[1,2,3].

LEM is thus a form of evolutionary computation radically different from the Darwinian model of evolution. It is guided by machine learning, rather than being a purely trial-and-error method. Instead of modeling biological evolution, it attempts to model *intellectual evolution,* the evolution of human artifacts, such as automobiles, computers, buildings, drugs, toys, refrigerators, skis, etc. In such an evolution, new designs are created as a result of an analysis of advantages and disadvantages of past designs by human designers, that is, through an intellectual process.

In all experiments with LEM that have been conducted, it has always outperformed tested Darwinian-type evolutionary algorithms in terms of the *evolution length*, defined as the number of fitness evaluations needed to achieve a target solution. It was able to find problem solutions that conventional Darwinian-type evolutionary methods either found after a much longer evolutionary process, or were unable to find at all. The latter situations have been encountered in optimizing functions of large

numbers of variables (e.g., about 1000 continuous variables), but were successfully solved by a program implementing learnable evolution[4,5].

Executing hypothesis generation and instantiation operators in LEM is, however, computationally more complex and thus more time-consuming than executing standard mutations and/or recombinations. Therefore, the LEM *evolution time*, defined as the time needed to reach the target solution, reflects the tradeoff between LEM's shorter evolution length and the longer execution time of its more complex operators. In our experiments, LEM employed a general purpose AQ-type rule learning program[6] that was not optimized for the application to learnable evolution. Despite this limitation, LEM always outperformed the tested conventional evolutionary methods in terms of the evolution length, and whenever fitness evaluation required more than a small fraction of a second, also in terms of the evolution time.

Because evaluating a design usually requires a non-trivial amount of time, LEM appears to be particularly suitable for design optimization problems. Our experiments show that the LEM advantage in terms of the evolution length grows with the complexity of the problem (as measured by the number of controllable parameters or variables). Thus, the more complex the engineering design problem, the more advantageous it may be to use learnable evolution.

The latter point is illustrated below by experiments on the minimization of the Rastrigin function for different number of variables. The Rastrigin function is defined by:

$$f(x_1,...,x_n) = 10*n + \sum_{i=1}^{n} (x_i^{\,2} - 10*\cos(2*\pi*x_i)) \qquad (1)$$

and has been frequently used as a benchmark for testing evolutionary algorithms. The function has multiple local minima, and one global minimum (Figure 1).

Figure 2 compares the performance of the LEM3 program implementing learnable evolution[5] and the EA program implementing a conventional, Darwinian-type evolutionary algorithm in optimizing the Rastrigin function for numbers of variables ranging from 10 to 1000. In this figure, the *y*-axis represents the number of fitness evaluations required to determine a δ =0.1 solution of the function of

the number of variables indicated on the *x*-axis. A δ =0.1 solution is a solution that is 10 times closer to the global optimum than the best solution in the initial population[5]. As one can see, the number of fitness evaluations required by EA significantly increases with the number of variables, while the number of fitness evaluations required by LEM3 grows relatively slowly.

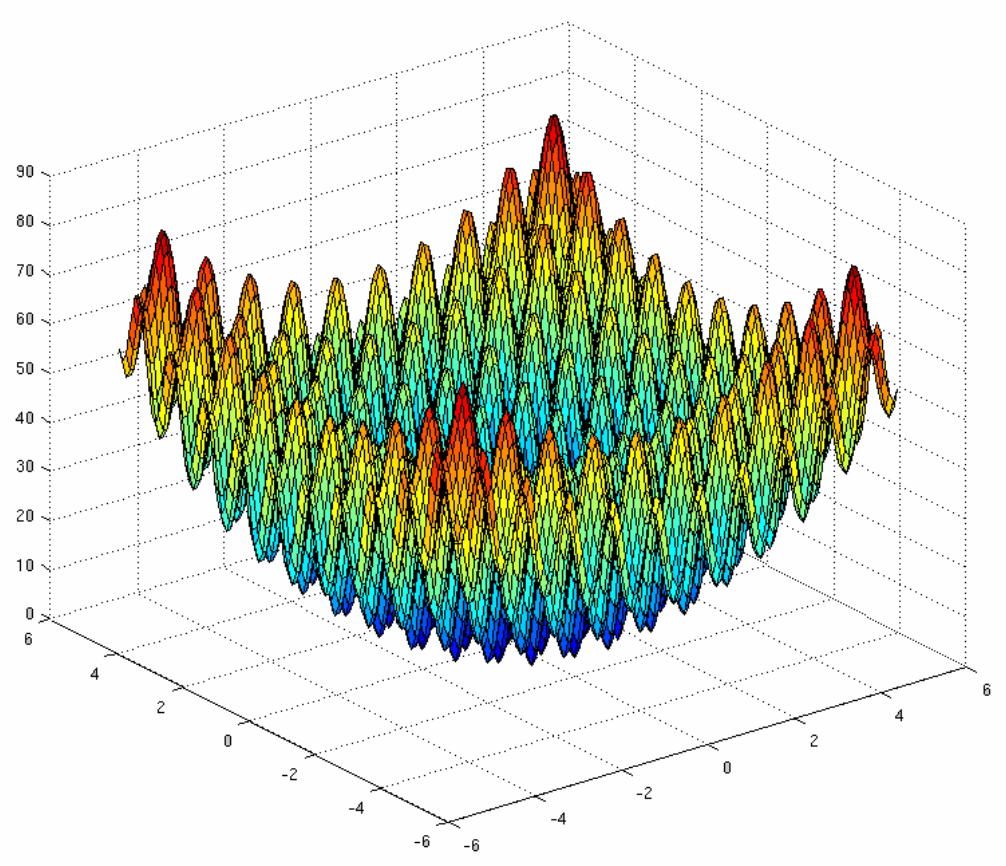

Figure 1. A graph showing the Rastrigin function of two variables.

The above behavior, which was consistently reproduced in experiments with different functions, indicates a high attractiveness of LEM for solving very complex optimization problems in which fitness evaluation is costly or time-consuming. Such problems frequently occur in the area of engineering design, where intricate simulators may be required to compute an individual's fitness[7]. This idea has motivated us to develop LEMd, a method that tailors the LEM methodology to complex design problems. Among the most important distinguishing features of LEMd is that it allows the user

to define problem-oriented change operators and problem constraints, and that it able to use them effectively in the process of design optimization.

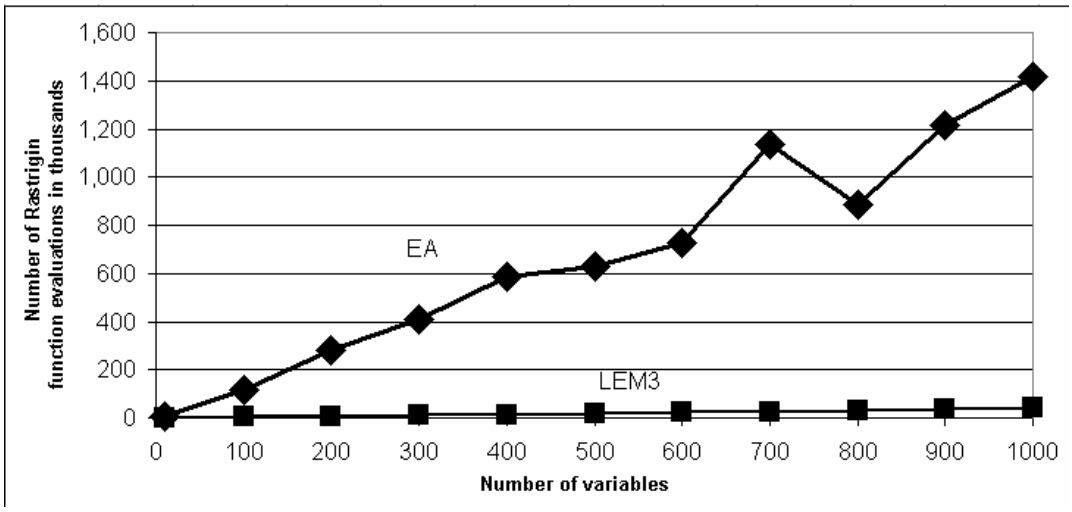

Figure 2. Performance of LEM3 and EA in minimizing the Rastrigin function of 10 to 1000 continuous variables for δ =0.1 (reprinted from [5] with the authors' permission).

In developing LEMd, we encountered problems characteristic of this application area. Design problems differ from typical function optimization problems to which LEM was originally applied not only in that the design evaluation function ("fitness function") is much more complex, but also in that such problems usually have intricate constraints that make a large fraction of the solution space infeasible. To reduce the evaluation of infeasible designs, the change operators should be designed in such a way that they navigate mainly through the feasible portions of the search space. Therefore, when applying LEMd to any particular domain, the domain's inherent constraints need to be identified and taken into consideration when implementing the design change operators.

The reduction of the evolution length is achieved in LEM by applying machine learning-guided design change operators for generating new individuals. Such operators make the search for the solution more directed rather than semi-random search implemented in conventional, Darwinian-type evolutionary computational methods. Because LEM's "intelligent' operators are computationally more complex than the random mutations and recombinations, a problem arises as to how to exploit

the tradeoff between the complexity of the innovation (change) operators and the evolution length in order to achieve maximum efficiency of evolutionary design process.

To this end, a general form of LEM integrates *Probing Mode*, which conducts a conventional type of evolution utilizing domain-specific knowledge, and *Learning Mode*, which uses machine learning to guide the evolutionary process. In such a general form, called *duoLEM*, LEM switches back and forth between Learning Mode and Probing Mode, according to the *Mode Switching Condition* that monitors the progress of the evolutionary computation in a given mode. A LEM version that limits its operation to only Learning Mode is called *uniLEM*.

In LEMd, Probing Mode does not simply apply conventional random mutations or recombinations, as in conventional Darwinian-type evolutionary computation, because such changes would hardly bring improvements in designing complex engineering systems. Instead, it makes design changes (innovations) that may plausibly lead to improved designs based on the expert knowledge. In LEMd, expert knowledge is incorporated in plausible change operators, called *design modification (DM) operators*.

The following sections present details on the LEMd method and its application to implementing two initial experimental systems, ISHED1 and ISCOD1 (briefly, ISHED and ISCOD), specialized for optimizing designs of, respectively, evaporators and condensers in cooling systems. To provide the reader with sufficient background, we start with a very brief review of the Learnable Evolution Model, based on its description in[2].

2 A Review of the Learnable Evolution Model

As mentioned earlier, the Learnable Evolution Model (LEM) represents a fundamentally different approach to evolutionary computation, compared to conventional Darwinian-type evolutionary methods. In Darwinian-type methods, new individuals are generated through mutation and/or recombination operators. Such operators are easy to execute, do not require much domain knowledge,

and thus can be directly applied to a wide range of problems. In search for innovation, these operators make random or semi-random changes in the individuals (in our case, designs) that take into consideration neither the traits acquired during the life experience of the individuals (as postulated in Lamarckian-type evolution), nor the properties of the entire population or a collection of populations (as in LEM's Learning Mode; see below). As a consequence, such unguided Darwinian-type evolutionary algorithms tend to be inefficient, which may significantly diminish their effectiveness in optimizing very complex real-world designs.

In LEM, the process of creating new individuals is guided by a machine learning program that creates hypotheses defining areas in the search space that most likely contain the sought optimum or optima. Such hypotheses are learned by the program from examples of the highest and lowest performing individuals in the current, and possibly also the past, populations of designs. Specifically, in Learning Mode, at each step of evolution, the current population is divided into three groups of individuals: those that score high on the fitness function, defined as the *H-group*, those that score low on the fitness function, defined as the *L-group*, and the rest. The partition of the population into these groups can be done using different methods, and is controlled by program parameters set by the user[2].

The selected H-group and L-group are then supplied to a learning program that induces a general hypothesis distinguishing between these groups. This hypothesis is then instantiated in various ways to produce new individuals for inclusion in the new population. Their fitness is determined by some evaluation method, which in LEMd will typically involve running a design simulator program. In principle, any inductive learning program could potentially be employed in the LEM methodology. In the current implementation, we use the AQ learning method, which produces hypotheses in the form of sets of attributional rules[13] that have proven particularly suitable for this application.

The most important feature of the LEM methodology is that in Learning Mode the generation of new individuals may not only be a function of the properties of individuals, but of the properties of the current population, or even of the entire history of the evolution process. Initial experiments have

shown that due to guiding evolutionary processes by learning, the LEM methodology can achieve a dramatic evolution length speedup[2,5].

The next section describes the LEMd system, which tailors the LEM methodology toward design optimization problems. Sections 4 and 5 describe details of its application to the development of two systems, ISHED and ISCOD, for heat exchanger design optimization. Section 6 describes related research, and the Conclusion makes suggestions as to the future work.

3 The LEMd Method

3.1 General Description

Design optimization problems often involve very large, poorly structured search spaces that are accompanied by multiple and complex constraints. Because of these features, such problems constitute an application domain for which the search capabilities of evolutionary computation algorithms can be particularly useful[8,9]. In design problems, variables represent controllable aspects of the system being designed, such as parameters affecting the system's performance, positions or configurations of system components, the type of the connections among them, the cost of the components, controllable operating conditions, etc.

Such variables are typically subject to many constraints; therefore, random changes to them may easily result in solutions that are either physically infeasible or practically unacceptable. Because the design search space may be extremely large, the subareas corresponding to infeasible designs may also be very large. This means that the fitness evaluator that recognizes infeasible solutions by returning appropriately low fitness scores for them (e.g., [10]) will be rejecting a large number of designs. In this situation, the effectiveness of learnable evolution will be severely compromised, as its main advantage is the reduction in the number of fitness evaluations in comparison to a conventional evolutionary computation method.

Thus, in applications concerning engineering design, problem-oriented design modification operators are needed that produce feasible solutions satisfying problem constraints, as opposed to random mutations and/or crossovers (e.g., [11]). This understanding guided us in developing LEMd's Probing Mode and appropriately integrating it with Learning Mode.

Basic features of LEMd are summarized below:

- It provides a mechanism for creating an initial design population from previously developed designs, the best known designs, expert-proposed most promising designs, random or partially random designs, or any combination of all these methods.
- It supports problem-oriented representation of variables to be optimized. These variables control the performance of the system being designed, and are subject to domain-specific constraints.
- It employs problem-specific *Design Modification (DM)* operators in Probing Mode that represent expert knowledge as to what type of changes to the current designs may improve them.
- It employs domain constraints in Learning Mode, and instantiates learned hypotheses in a way that generates only feasible designs.

A precondition for application of LEMd is that the quality of initial and subsequently generated candidate designs can be evaluated in some way, for example, by a design simulator. A general flow diagram of the LEMd method is presented in Figure 3.

The initial design population may include user-specified designs, or it can be generated randomly using one the methods mentioned in Section 4.2. Details on representing individuals in the population are given in Section 3.2.

The next task is to choose mode of operation, which can be Learning or Probing. The initial choice may be arbitrary, but once a mode is chosen, it is used until the *Mode Switching Condition* is satisfied, which occurs when after a given number of iterations, no satisfactory progress in the design improvement has been achieved. Then, either control switches to the other mode, or LEMd is terminated. Learning and Probing Modes are described in detail in Sections 3.3 and 3.4.

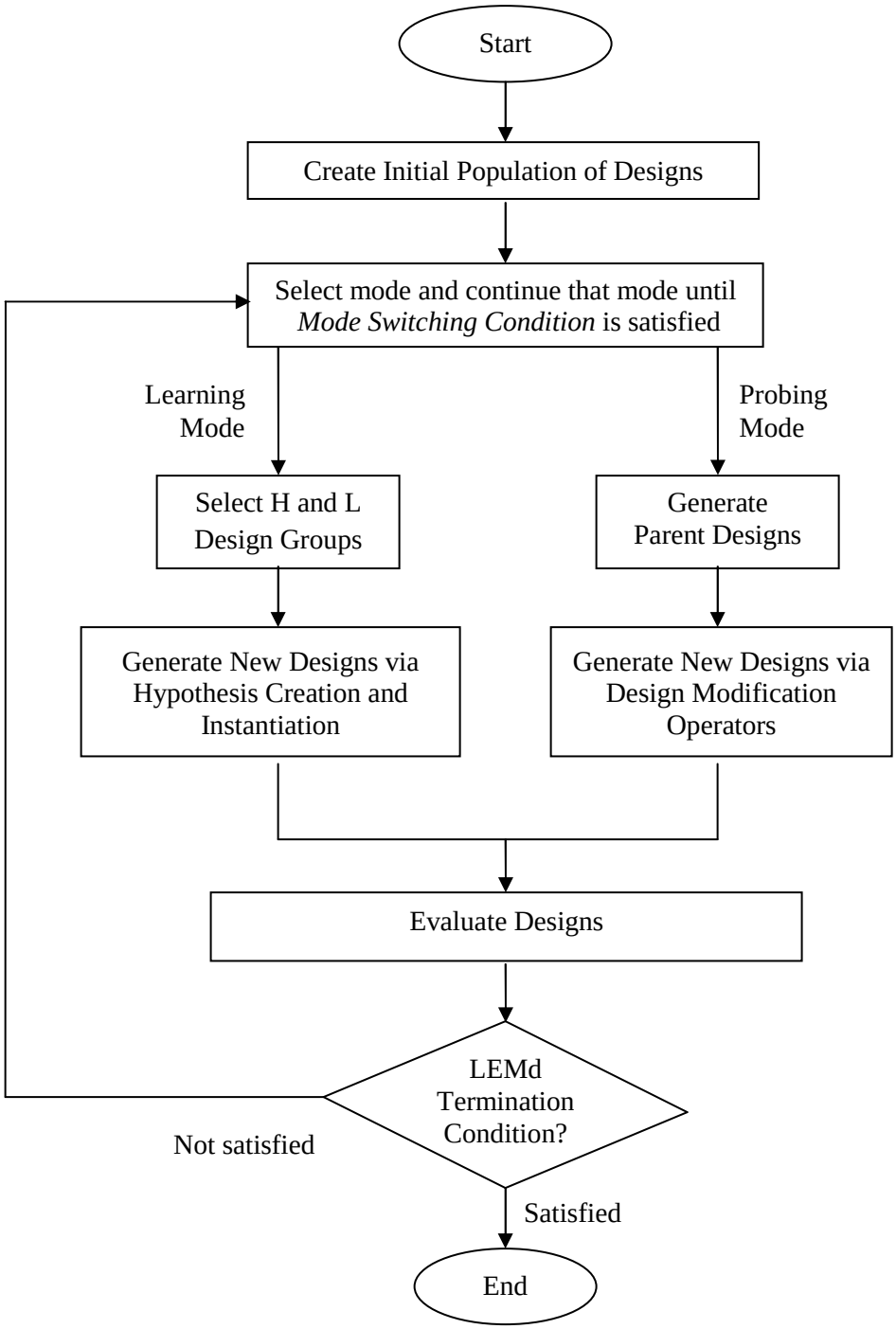

Figure 3. A flowchart of the LEMd system.

After utilizing either mode to generate a new population, the new designs are evaluated to determine their quality (to compute a fitness score). The evaluation can be done by running a design simulator,

by a subjective judgment of an expert or group of experts, or by some other method. After that, the LEMd termination condition is tested, and if satisfied, the program ends; otherwise, the population of designs with their evaluations is passed to the Control Module to start the next generation (iteration). The LEMd termination condition is met either when the obtained best design is deemed satisfactory, or the allocated computational resources are exhausted.

If the Mode Switching Condition is set up so that the program never switches to another mode, LEMd works either as uniLEM (executing only Learning Mode) or as a conventional evolutionary algorithm (executing only domain-oriented Probing Mode). This feature facilitates comparative studies of the two modes.

To apply LEMd to a specific design domain, one needs to define a suitable representation for candidate designs, specify and implement design modification operators suggested by an expert in the given application domain, determine the design constraints and algorithms for applying them, and specify parameters for executing Learning and Probing Modes.

3.2 Attributional Representation of Designs

In the application of LEMd to heat exchanger design (Section 4), Learning Mode employed an AQ type learning method (AQ19) that requires that designs be represented as vectors of attribute values[6]. The method allows one to use many different attribute types in describing a design, such as nominal, structured, rank, cyclic, interval, ratio, and absolute. The attribute types are taken into consideration when generating new candidate designs. Nominal attributes are used to represent unordered sets, such as, for example, the type of refrigerant selected for use in a heat exchanger. Structured attributes are used to represent hierarchically ordered domains; an example would be the type of material from which a certain component was made (assuming that material types are arranged into a hierarchy). Rank attributes are used to represent ordered lists of discrete values, for example, a qualitative characterization of a component's heat conductivity (e.g., low, medium, high). Cyclic attributes are used to characterize cyclically ordered qualities, for example, the compass direction that a piece of

equipment faces (e.g., North, Northeast, East, etc.). Interval attributes represent properties measured on a scale in which 0 is arbitrary, for example, temperature in degrees Fahrenheit. Ratio attributes are used to represent measurements of physical quantities, such as the length of an object in predefined units. Absolute attributes represent counts of distinct elements, such as the number of outlet tubes in a heat exchanger.

3.3 Learning Mode

In Learning Mode, the first step is to determine a group of the highest and a group of the lowest quality designs in the population, called the H- and L-groups, respectively. We assume that the quality ("fitness") is determined by a design simulator[2,5], however, any method can be used for this purpose in LEMd. The H- and L-groups can be formed solely on the basis of the current population, or involve also past populations. Section 4.3 describes how the H- and L-groups are selected in ISHED and ISCOD.

The H- and L-groups, serving as positive and negative examples, respectively, are then supplied to a learning program that creates a general hypothesis distinguishing between them. The program outputs a hypothesis, in the studied case in the form of a set of rules that generally characterize distinctions between the H- and L-groups of designs. Each rule defines a subspace in the search space that likely contains a high quality or even the best design. In ISHED and ISCOD, the AQ19 learning program[6] is used for this purpose, due to its particularly useful features for this type of application (see Section 4.3).

Here is the procedure for creating new designs in LEMd:

1. Arrange rules in the ruleset representing a hypothesis learned from H- and L-groups in the descending order of their significance, which is measured by the sum of the fitness values of the H-group designs satisfying that rule.

2. Instantiate each rule in different ways to produce different designs. The number of designs generated from a rule should be roughly proportional to the rule's significance, but not smaller

than parameter *i-min*. The total number of new designs generated is *Population-size – elite-size*, where *elite-size* is a parameter defining the number of the best performing designs ("elite") found so far that are to be propagated to the next population. By including elite designs in every new population of designs, the system keeps track of the best designs that were determined during the evolution process.

The reason for parameter *i-min* in step 2 is to enforce parallel exploration in different subareas of the search space. Because each rule carves out a subarea in the search space, this parameter guarantees that all of these subareas are represented in the population of designs, regardless of their current significance. This feature is particularly important in the case of multi-modal landscapes.

The above procedure has been used for creating new designs in the ISHED and ISCOD systems described in Section 4.

3.4 Probing Mode

After designs in the current population are evaluated, Probing Mode generates a set of new designs that are then selected as parent designs for the next population. Each design selected to be a parent design may be used to derive multiple new designs for the next generation. The selection may be done by applying one of the well-known selection methods developed in the field of evolutionary computation, for example, a probabilistic selection or tournament selection. The designs so selected join the elite in the parent set.

A new population is then generated through the application of design modification (DM) operators to the parent designs. These operators represent experts' assessments of meaningful changes in designs that are may improve the quality of the design while maintaining design feasibility. Section 4.4 presents eight DM operators used in Probing Mode of ISHED and ISCOD for the optimization of heat exchangers.

4 Case Study: Optimizing Heat Exchanger Designs

4.1 Problem Description

The LEMd method was used to develop two systems for optimizing heat exchangers, one, ISHED, for optimizing evaporators, and the second, ISCOD, for optimizing condensers. The problem of optimizing heat exchangers is complex and economically very important due to the widespread use of heat exchangers in air conditioners and refrigerators. This application was done in consultation with an expert in the area of cooling systems, Dr. Piotr Domanski of the National Institute of Standards and Technology.

To explain the optimization problem faced in this application, let us start by briefly describing how an air conditioning unit works. An air conditioning unit contains an evaporator that transfers heat from an interior space (a room, a refrigerator, an automobile, etc.) into the refrigerant, and a condenser that transfers heat from the refrigerant into the outside air. A refrigerant is a fluid with a low boiling temperature that flows in a loop through both of the above components. It enters the evaporator in liquid form but then turns into gas by absorbing the thermal energy from the warm interior air passing over the evaporator. Subsequently, in the condenser, it is placed in contact with cooler outside air, transfers its thermal energy into the air, and liquefies. Returning to the evaporator, it comes once again into contact with the warm interior air, and the cycle repeats.

Both evaporators and condensers consist of arrays of parallel tubes, encased in a structure, through which the refrigerant flows back and forth. For illustration, Figure 4 shows an example of an evaporator. The tubes are connected by *tube bends* on both sides of the array, as specified in the design. The figure shows only one side of the array. Tube connections on this side are indicated by solid lines, and those on the other side are indicated by dashed lines. The tubes are numbered from left to right in each row, starting with the first row (as viewed from the direction from which the air flowing over the tubes arrives).

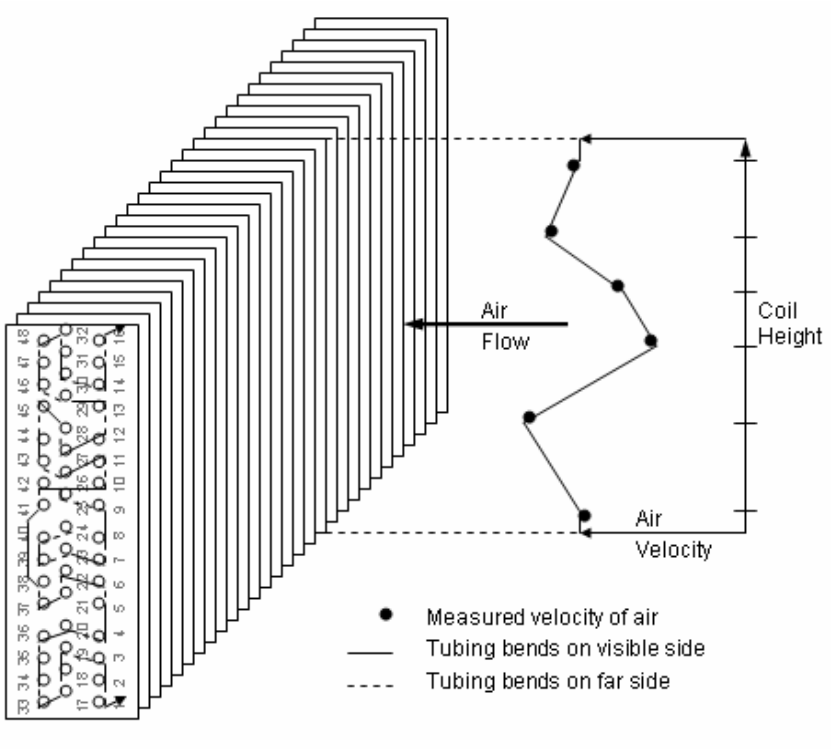

Figure 4. An illustration of 16 x 3 evaporator circuitry.

In general, in a heat exchanger, tubes are connected so that they provide a single path or several independent paths for carrying refrigerant from the input (inlet) to the output (outlet). In an evaporator, some paths may split into two paths on their way to the output, and in a condenser, some independent paths may merge into one. Such a split or merge is achieved by appropriately connecting the tubes. For example, in Figure 4, refrigerant enters the evaporator at tube 24 and flows through tubes 40, 39, 23, 7, 8, 9 and 25, at which point the path splits.

The heat exchanger configuration specified by flow paths created by tube connections affects the heat exchanger's *capacity*, defined as the amount of heat that it can transfer per unit time. The problem of optimizing a heat exchanger (e.g., an evaporator) is to determine a tube configuration that for the given technical and environmental conditions maximizes its capacity. These conditions include the number of tubes, their length and diameter, the type of refrigerant, the distribution of velocity of the air passing over the heat exchanger (see graph in Figure 4), and the average outside and inside temperatures. The technical and environmental conditions are given as assumptions for the problem because they affect

the capacity of the heat exchanger, and thus for different conditions, different configurations of tubes will be optimal.

To get a feel for the complexity of the problem, consider that in a moderate-sized evaporator, such as the one shown in Figure 4, each of the 48 tubes can in theory receive its refrigerant from any of the other 47 tubes, or from the flow of refrigerant into the evaporator. Thus, a naïve search of the design space would have to consider 48^{48} potential designs. Of course, the vast majority of these are infeasible because of the problem constraints, which are discussed in more detail below, but even if a method could be designed that searches only the feasible part of the space, this part would still be orders of magnitude too large to permit any sort of exhaustive search.

To solve this problem, we reduce the search space by defining appropriate constraints, and apply learnable evolution to search for an optimal design in the space of feasible solutions. Based on discussions with a domain expert, we identified six basic constraints and built them into the ISHED and ISCOD programs. The constraints are numbered and ordered in decreasing order of their importance. A controllable threshold number splits them into *strict* constraints that must be satisfied and *flexible* ones that are recommended to be satisfied, if possible. A user can set the threshold according to the needs of a given problem, so that all, some, or none of the constraints are viewed as strict. Here are the six constraints:

1. There can be no looping in the refrigerant flow. Refrigerant may not leave a tube and return to the same tube without exiting the exchanger unit and passing through the rest of the refrigerant circuitry.

2. All inlets and outlets (i.e., locations where the refrigerant enters and leaves the exchanger, respectively) must be on the same side of the heat exchanger manifold. This means that all possible flow paths from inlets to outlets must pass through an even number of tubes. As a corollary, if the heat exchanger contains an odd number of tubes (for example, three rows

of 15 tubes each), at least one path must fork. In Figure 4, the inlet at tube 24 and the outlets at tubes 1 and 16 are all on the same (foreground) side of the manifold.

3. There can be only one split in an independent path through an evaporator, and only one merge in an independent path through a condenser. This constraint comes from an empirical observation that more than one split or merge leads to a lower capacity

4. Inlet and outlet tubes should not be adjacent to one another. By the time the refrigerant has reached the outlet, it has undergone a full phase change, and will be at a temperature substantially different from that in most of the rest of the heat exchanger. If it is adjacent to inlet refrigerant, that increases the amount of heat transfer between the inlet and neighboring tubes, causing a decrease in capacity.

5. The exit tubes should be in the first depth row (the first row the air flows over). In condensers, the inlet tubes should be in the last depth row. The reason for this is that in a condenser (evaporator), the air passing over the first depth row is cooler (warmer) than the air passing over the last one, and the refrigerant at the inlet is warmer (cooler) than the refrigerant at the outlet. (It has been determined that more heat transfer is achieved overall when the cooler air passes over the cooler refrigerant and the warmer air over the warmer refrigerant.)

6. The tube from which an exit tube receives its refrigerant should be adjacent to the exit tube. The reason for this is similar to that in (4): The refrigerant in the exit tube's predecessor will have a temperature significantly closer to that in the exit tube than the temperature of most of the rest of the heat exchanger will. Thus, this will reduce the amount of inter-tube heat transfer.

These constraints are applied in developing the initial population of designs, and in instantiating rules learned in Learning Mode. In Probing Mode, they are used to test each new design generated by a DM operator. Only those that satisfy the strict constraints are added to the population for evaluation.

4.2 Overview of ISHED and ISCOD Systems

This section describes how the LEMd method is implemented in two specific systems, ISHED for optimizing evaporator designs, and ISCOD for optimizing condenser designs. Their basic design closely follows the flowchart in Figure 3.

The first task for these programs is to determine an initial population of designs. Both ISHED and ISCOD allow the user to specify designs with which to seed the initial population. If there is a need for the system to generate any initial designs, they are generated by a random process that is constrained by a required distribution of different types of designs. These constraints are specified in a table, developed on the basis of expert advice, that defines the percentages of different types of designs to be generated. For example, the table may require 20% of the generated designs to have one inlet and one outlet, 35% to have one inlet and two outlets, etc. These percentages are dependent on the total number of tubes in the heat exchanger being optimized. For instance, smaller exchangers usually function better with fewer inlets and outlets. Heuristic procedures are used to build designs according to these specifications.

In ISHED and ISCOD, designs are internally represented by arrays of integers, one integer per tube in the heat exchanger being optimized. In an array representing an evaporator design, each integer represents each tube's source of refrigerant, and in an array representing a condenser design, each integer represents the refrigerant destination after leaving each tube.

Both Learning and Probing Modes use elitist strategies, that is, they keep track of the best-performing designs at each step of computation. These designs constitute an *elite*, which is included in every new population of designs. In Probing Mode, the elite consists of *elite-size* best performing designs determined thus far, where *elite-size* is a parameter of the program (in our experiments, *elite-size* was 1). In Learning Mode, the full H-group comprises the elite.

Which mode to use is determined by a control module that executes one mode until a *Mode Switching Condition* is satisfied, which in ISHED and ISCOD is defined by two parameters specified in the input

file to these programs. One parameter tells how many generations with no progress in both the best and the average value of the fitness function triggers a switch from Probing Mode to Learning Mode, and the other tells how many generations without progress triggers a switch in the opposite direction. Sections 4.3 and 4.4 provide details about Learning Mode and Probing Mode, respectively, in the ISHED and ISCOD systems.

Designs are evaluated by a numerical simulator developed at NIST[12] that determines their capacity. Running the simulator is the most time-consuming part of the evolution process. Given that an evaluation of a single design may take on the order of half a minute on a standard PC, and a typical LEMd run in our experiments required over a thousand fitness evaluations, a single run may take about six hours. Without the LEM-provided reduction in evolution length (which can be an order of magnitude or more[5]), a conventional evolutionary method could then take about sixty hours to complete a run that achieves a similar result. This observation indicates a significant practical advantage of learnable evolution.

4.3 Learning Mode in ISHED and ISCOD

To execute LEMd Learning Mode, ISHED and ISCOD select the H- and L-groups from the current population of designs using a *fitness-based* method [2]. The fitness-based method starts by determining the *fitness range*, defined as the difference between the highest and the lowest fitness values in the given population. The method has two control parameters, *High Fitness Threshold* (HFT), and *Low Fitness Threshold* (LFT). Suppose, for example, that HFT and LFT are both set to 30%. The H- and L-groups will then consist of designs whose fitness values fall within the top 30% and bottom 30% of the fitness range, respectively. These parameters are illustrated in Figure 5 using a *fitness profile function*, which is a mapping of the population of designs ordered by fitness values (the *x*-axis) into their fitnesses (the *y*-axis).

The designs in the H- and L-groups are then abstracted as follows. Each tube in the design is characterized as an inlet tube, an outlet tube, a tube at a fork point, or a generic interior tube. Thus,

each tube is described by a four-valued attribute. In addition, from these attributes, two other attributes are derived, the numbers of inlet and outlet tubes in the design, For example, in the evaporator represented in Figure 4, tube 24 would be labeled as an inlet tube, tubes 1 and 16 as outlet tubes, tube 25 as a fork point, and the other 44 tubes as interior tubes. Attributes would specify that the design had one inlet and two outlets.

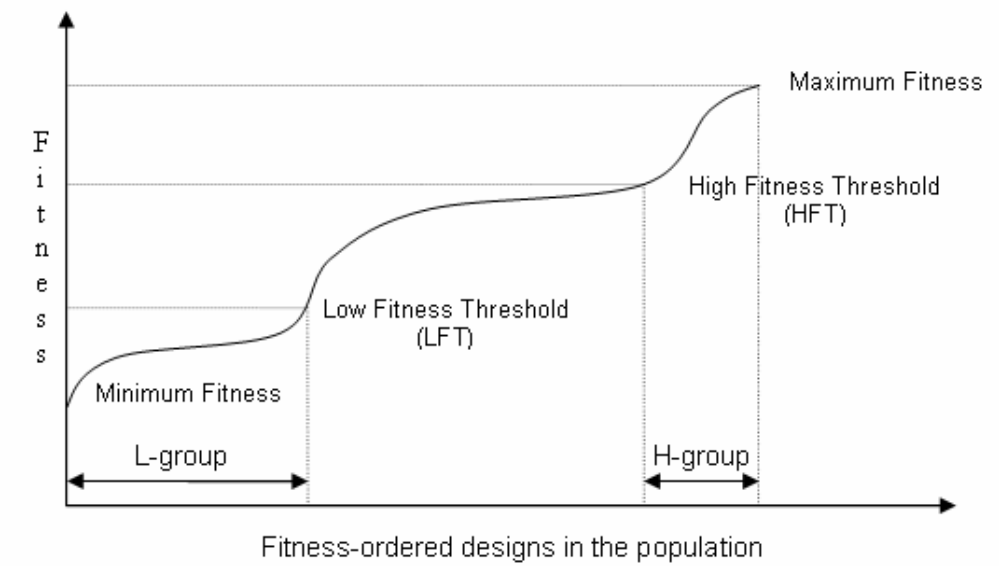

Figure 5. A fitness profile function and parameters used for selecting H- and L-groups.

Each design in the H- and the L-group is thus represented by a vector of attribute values, whose length is the number of tubes plus two. Thus, in the case of the evaporator in Figure 4, there would be 48 four-valued nominal attributes denoting the role of each tube, and 2 absolute attributes representing the number of inlets and outlets in the evaporator. The designs in the H- and L-groups, so encoded, are then supplied as input to a learning program. They serve as positive and negative examples of the concept to be learned, respectively. In this application, the concept here can be characterized as "the set of most promising designs."

ISHED and ISCOD employ the AQ19 learning program[6], which given the H- and L-groups, hypothesizes a set of *attributional rules* that distinguish all designs in the H-group from all those in the L-group. The conditions in the rules may include internal disjunctions of attribute values, ranges of

values, and other constructs, unlike conventional decision rules, whose conditions are all limited to the <attribute-relation-value> form[13]. Using such expressive conditions makes the representation language not only more powerful, but facilitates the instantiation of a single rule into diverse individuals. This feature makes a learning program particularly suitable for implementing the LEM evolutionary process.

In ISHED and ISCOD, AQ19 generates hypotheses in the form of *characteristic concept descriptions*[14] that represent a relatively *low degree of generalization* of the H-group, that is, they are relatively specific. The reason for learning low generalization degree descriptions is that they tend to specify values for a large number of attributes in each rule, which is helpful to the instantiation process that creates new designs.

To create new designs, the learned rules are ordered from the most significant to least significant, and their instantiation proceeds in that order. The number of instantiations of a given rule is roughly proportional to its significance (the sum of fitnesses of designs in the population that satisfy that rule). Such a method is called *proportional instantiation*.

To instantiate a rule, the program assigns values to variables in the rule that satisfy the rule. This can be done either randomly, or according to some algorithm. Variables that are not present in the rule are assigned values from randomly selected members of the H-group. The number of designs to be generated by instantiation is the size of the target population minus the size of the elite (which in Learning Mode is the size of the H-group).

An instantiated vector of attribute values is an abstract representation consistent with a set of specific heat exchanger designs. To arrive at a specific design from that set, the program seeks a design that satisfies the six constraints defined in Section 4.1. To this end, given a set of inlet and outlet points, ISHED and ISCOD determine the physical relationships among these points (e.g., which are to the left of, to the right of, above, below, in the same column, or in the same row as others), and creates a specific design from the abstract design.

To create such a specific design, ISHED and ISCOD utilize a divide-and-conquer method, in which the heat exchanger's tubes are divided into groups to be connected into one path based on physical location. For the sake of ensuring short connections between tubes (on both sides), the groups are made as compact as possible. If there is a split (merge) on the path of an evaporator (condenser), the split (merge) point is treated as additional inlet (outlet). After both paths have been designed, they are united at the split (merge) point.

Due to the constraints defined in Section 4.1, the design search space is divided into many different niches of feasible designs. Therefore, the process of locating these feasible designs is very complex and laborious, and in our experiments usually produced only a very small set of different designs. As a result, the newly generated population may have many repetitious designs, and the diversity of designs in the H- and L-groups will be small, which negatively impacts Learning Mode. When this happens, the control switches to Probing Mode that generates a new, more diverse set of designs.

4.4 Probing Mode in ISHED and ISCOD

In ISHED and ISCOD Probing Mode, the parent designs for the new population are determined by proportional selection (the number of copies of a design included in the parent population is probabilistically proportional to the design fitness). Each parent design is then subjected to one design modification operator. This process is controlled by a table, developed on the basis of expert advice, that defines suggested probabilities of applying different operators (described below) for the given number of inlet and outlet tubes in the design being modified. The location at which to apply the operator is selected randomly from among the applicable sites.

In Figures 6 to 15, individual rectangles represent single tubes. An input arrow to any rectangle and the outgoing arrow from it connected to other rectangle output are on opposite sides of the heat exchanger.

As mentioned earlier, in an evaporator (ISHED) a path may split, but not merge, and in a condenser (ISCOD) a path may merge, but not split. Therefore, in an evaporator, each tube has only one predecessor tube, and in a condenser, each tube has only one successor tube. In order to define design

modification operators in a way applicable both to ISHED and ISCOD, the following convention was adopted: In ISHED, the operator denoted OP1(A1, A2) indicates that the operator OP1 is to be performed at the tube connections immediately preceding tubes A1 and A2 (the nature of OP1 determines what is to happen at those modification points). Similarly, in ISCOD, given OP2(A1,A2), the modification points are in the connections immediately following tubes A1 and A2.

The following eight design modification (DM) operators have been defined for ISCOD and ISHED:

1) ADD-FORK operator (which is called SPLIT in ISHED, and MERGE in ISCOD)

   This operator adds a fork in a path. Two modification points on the same path are selected (by the program, as described above), one of which is an attachment point, and the other a break point. The operator breaks a refrigerant path at the break point (creating a new inlet or outlet) and grafts one of the halves onto a path at the attachment point. Examples of these operators are shown in Figures 6 and 7. In Figure 6, the refrigerant path 1-2-3-4-5-6 is broken between tubes 4 and 5, and the tail end of the path is reattached between tubes 2 and 3. Now the refrigerant flows through tubes 1 and 2 before going down one of two paths: 3-4 or 5-6.

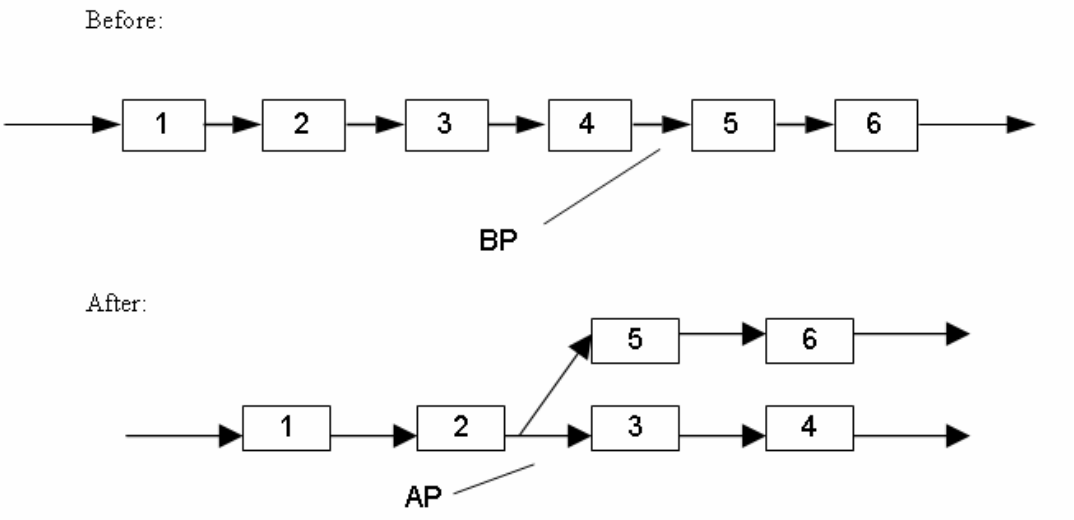

Figure 6. Application of the SPLIT(3,5) operator.

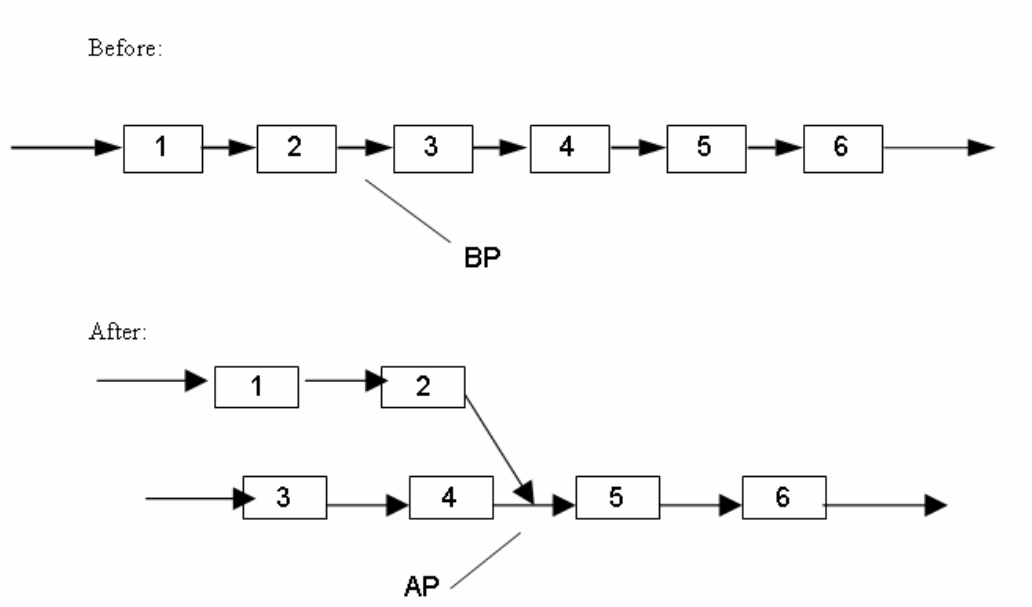

Figure 7. Application of the MERGE(4,2) operator.

2) A BREAK operator, which breaks a path at a given point, creating a new inlet and a new outlet location at the tubes following and preceding the break, respectively. An example of this operator is shown in Figure 8. The path 1-2-3-4-5-6 is broken into two paths: 1-2-3-4 and 5-6.

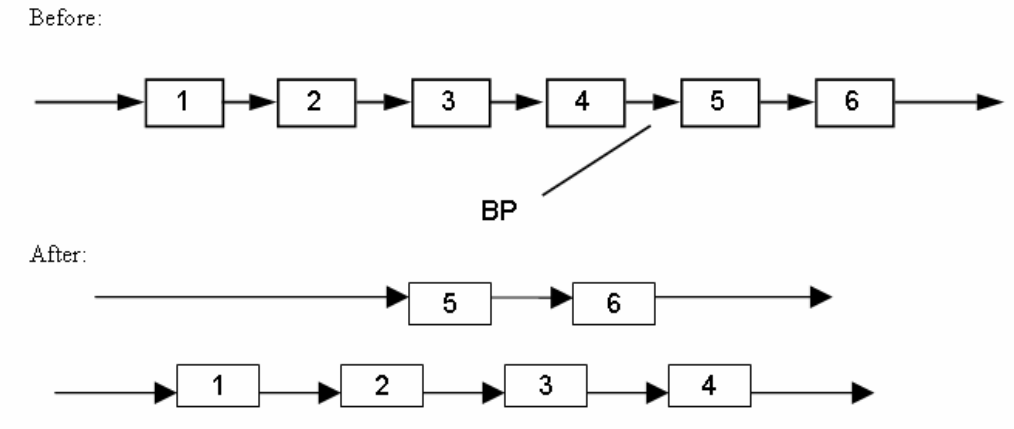

Figure 8. Application of the ISHED BREAK(5) or ISCOD BREAK(4) operator.

3) A COMBINE operator, which combines two separate unforked paths into a single, forked one. An interior attachment point is selected on one of the paths, and the inlet to the second path is

grafted onto it there to create a split in the case of an evaporator, or the outlet point of the second path is grafted onto it there to create a merge in the case of a condenser. An example of the operator for condenser modification is shown in Figure 9, and for evaporator modification in Figure 10. In both cases, the path 7-8-9 is attached to the path 1-2-3-4-5-6 between tubes 3 and 4.

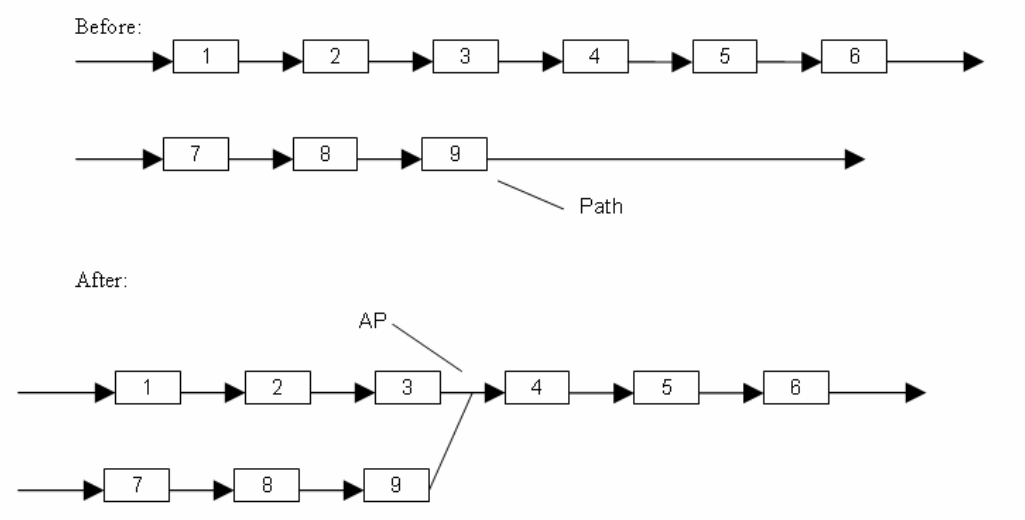

Figure 9. Application of the ISCOD COMBINE(9,3) operator.

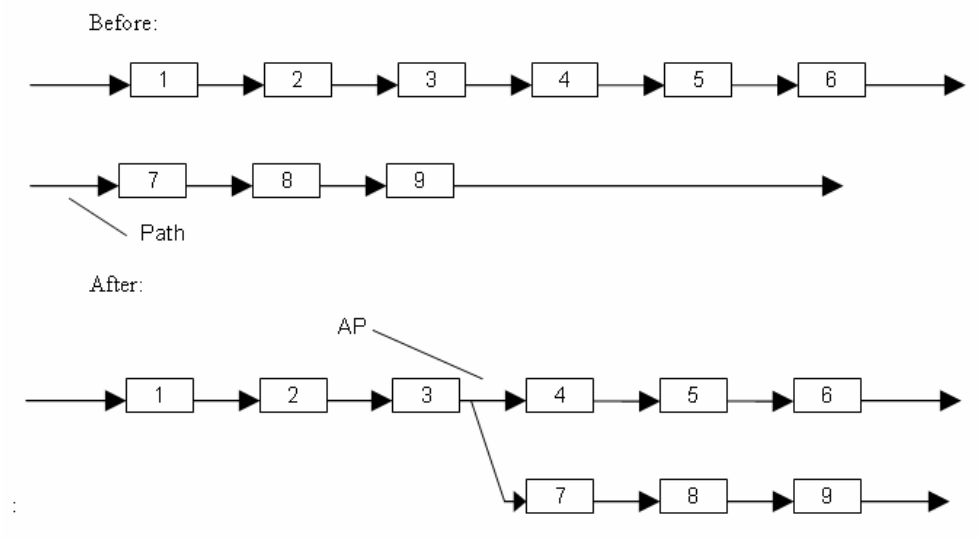

Figure 10. Application of the ISHED COMBINE(7,4) operator.

4) An INSERT operator, which combines two separate unforked paths into a single one. One of the paths is broken at a break point, and the paths are concatenated so that the tube that had preceded the break point now feeds the inlet of the second path, and the outlet of the second path now passes refrigerant to the tube that had followed the break point. An example of the operator is shown in Figure 11.

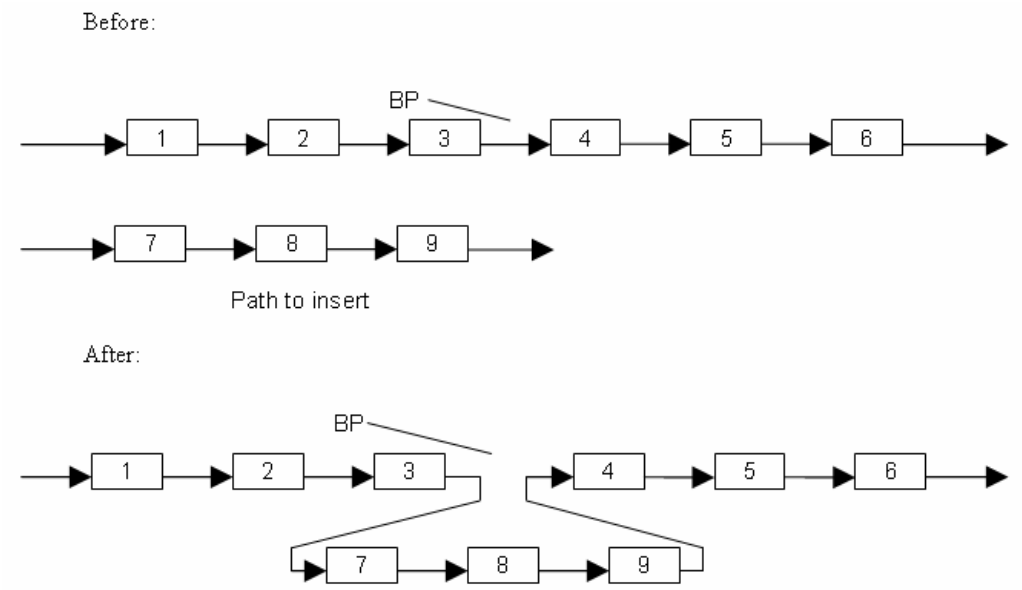

Figure 11. Application of the ISHED INSERT(7,4) and ISCOD INSERT(9,3) operators

5) A MOVE-FORK operator, which moves a fork in a path up or down to an earlier or later point on the path. Figure 12 shows an example of the operator applied to a split in an evaporator path, and Figure 13 to a merge in a condenser path. In both cases, the 7-9 branch beginning is moved upstream two tubes in the path 1-2-3-4-5-6. The operator is denoted MOVE-FORK(FP, Distance). In ISHED, positive distances indicate a downstream move, and in ISCOD, they indicate an upstream move.

6) A SWAP operator, which interchanges the position of two successive tubes in the refrigerant path. An example of the operator` is shown in Figure 14. Tubes 3 and 4 are exchanged, making the resulting path 1-2-4-3-5-6.

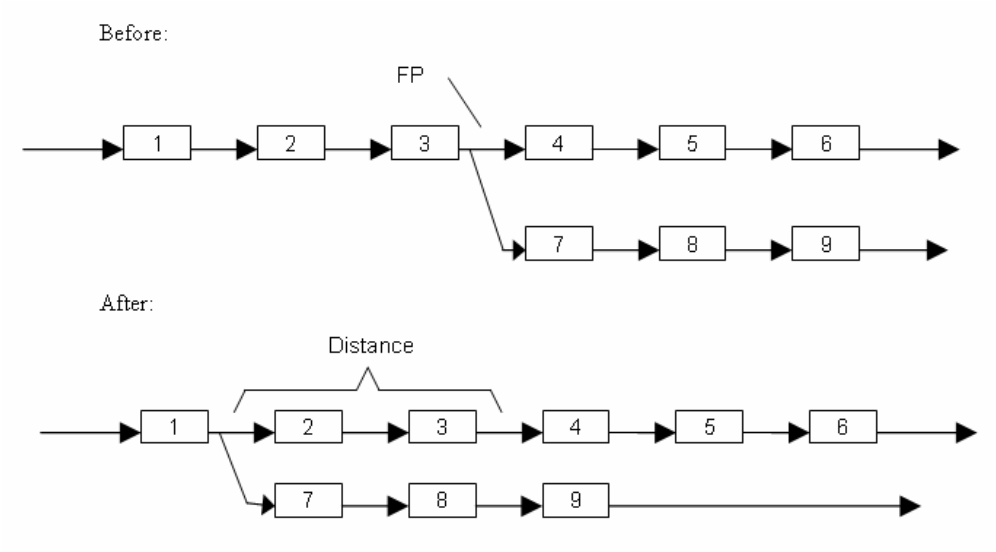

Figure 12. Application of the ISHED MOVE-FORK(7,-2) operator.

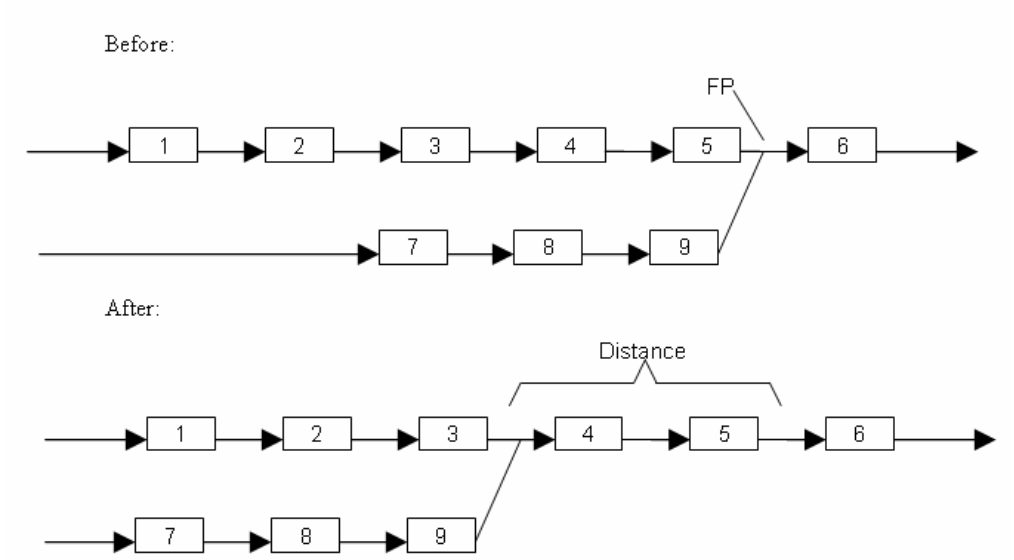

Figure 13. Application of the ISCOD MOVE-FORK(9,2) operator.

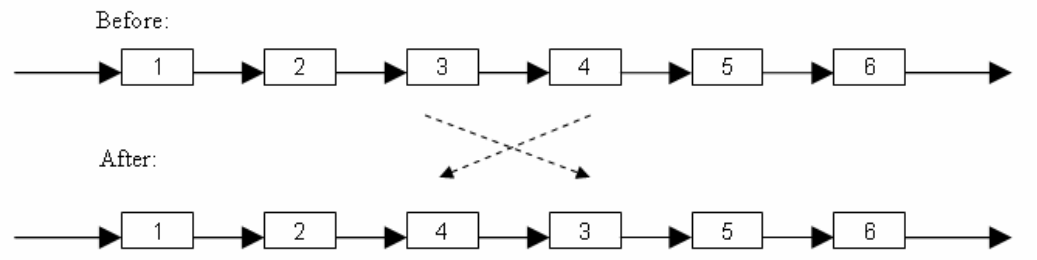

Figure 14. Application of the operator denoted SWAP(2) in ISCOD and SWAP(5) in ISHED

7) An INTERCROSS operator, which given break points on two separate refrigerant paths exchanges the portions of the paths following the breakpoints (analogous to a genetic crossover). An example of the operator is shown in Figure 15, resulting in the two paths 1-2-11-12 and 7-8-9-10-3-4-5-6.

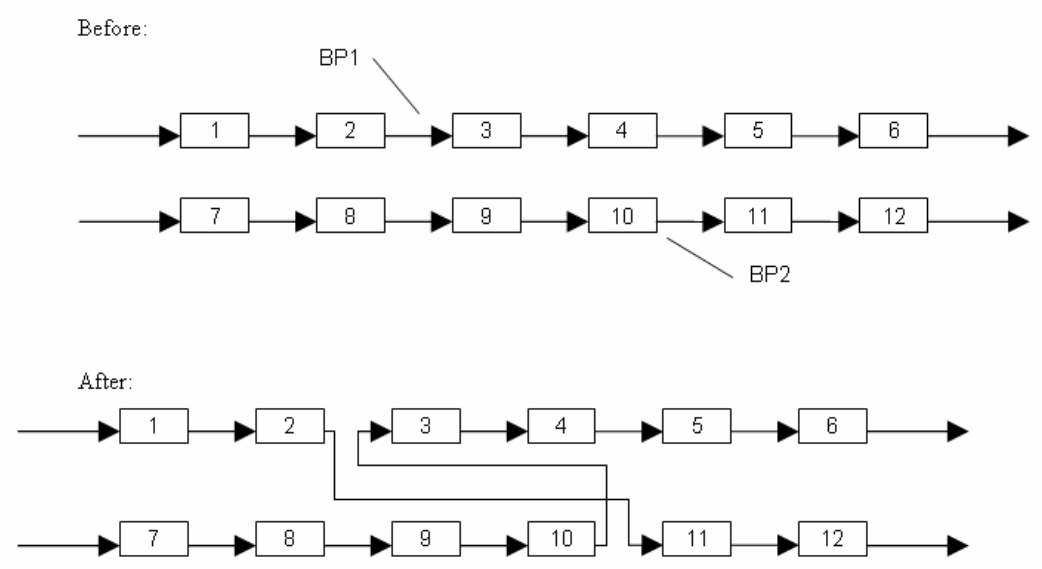

Figure 15. Application of the operator denoted INTERCROSS(3,11) in ISHED and INTERCROSS(2,10) in ISCOD.

8) A NEW-CONNECTION operator, which simply assigns a new connection to the operand tube – a new source in the case of an evaporator, and a new destination in the case of a condenser.

5 Experimental Validation of ISHED and ISCOD

Experiments with ISHED and ISCOD were performed under different values of design parameters, such as type of refrigerant, evaporator and condenser sizes and shapes, and airflow patterns (uniform or non-uniform). Industrially available air conditioning systems typically perform very efficiently for uniform airflow patterns. However, if the airflow is not uniform their efficiency drops off sharply. This is so because the side of the unit over which more air flows has a heavier cooling burden. Thus, for best performance that side needs to carry more and colder refrigerant in evaporators, and more and

warmer refrigerant in condensers. Manufacturers generally have not been building models adapted to non-uniform airflow patterns.

This section focuses on a set of experiments with ISHED aimed at optimizing evaporator designs for different technical and environmental conditions. Because similar results were observed with the application of ISCOD to condenser design, we will focus on results obtained from experiments with ISHED.

The initial experiments with ISHED concentrated on a problem well-known in the industry, 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 Probing Mode evolution, designs would become chaotic in terms of their inter-tube connections (and the simulator wasn't fully reflecting the detrimental effect of this). This problem was reduced by tightening restrictions on the length of permissible tube-to-tube connections, by modifying the simulator to more harshly penalize such designs, and by using visualization tools that allowed the expert to smooth some of the connections without significantly affecting the estimated capacity of the exchanger [15].

In later experiments, the airflow pattern was 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 heat exchangers.

Subsequent experiments varied the refrigerant used, and the size and shape of the heat exchanger. Specifically, they concerned heat exchangers with 40 to 90 tubes, arranged in 2 to 4 rows. Again, in these experiments, ISHED's designs matched human designs for uniform airflow, and exceeded them for non-uniform airflow.

The next experiments concerned the optimization of evaporators when the initial populations had pre-specified members, which were either existing industrial models, or results from prior ISHED runs. Their number varied from 1 to the entire population, whose size varied between 10 and 100.

When a large portion of the initial population (about 60%) was seeded with high-quality human or program designs, ISHED (as well as ISCOD) was often able to further improve them. This means that ISHED provides an effective tool for optimizing existing designs.

A counterintuitive result was that when the initial population had a small number of high quality pre-specified designs and many lower quality random designs, it was difficult for the program to find a design that would exceed the best design in the initial population. The presence of mostly low quality designs in the population apparently hindered the ability of these high quality designs to evolve into better ones. It is also possible that low quality designs that potentially could significantly improve were prevented from doing so by the presence of a few high quality designs. Finding a conclusive explanation of this phenomenon is an interesting topic for further research.

An example of the output from an ISHED run is shown in Figure 16. This example was obtained in the program's verbose mode, which records every design evaluated, every operator applied, and every rule learned. For the purpose of brevity, the figure only shows a very small sample of the full output just to give the reader a flavor of the ISHED optimization process. Line numbers on the right side are cited in the explanations in the text below. In addition to these explanations, some comments (in italics).were added to the log itself.

A candidate design of an evaporator is represented as a vector of integer values. Tubes are numbered left-to-right starting with the first row, and each value in the vector is the number of the tube that provides refrigerant to the given tube, or is an 'I' to indicate that the tube is an inlet. The simulator evaluates the design represented by this vector, and returns the estimated capacity of the design.

Exchanger Size: 16 x 3 1

Population Size: 15 #Generations: 41 2

Operator Persistence: 5 3

Mode Persistence: #ProbingGens=2 #LearningGens=1 4

Initial population (Generation 1): 5

Design #1.3: 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 6
43 44 28 46 30 48 32: **Capacity = 5.5376** 7

Design #1.8: 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 8
41 27 43 29 45 31 47: **Capacity = 5.2099** 9

and 13 others 10

Selected Members: 3, 2, 3, 7, 9, 3, 9, ... 11

Operations: NS(23, 39), SWAP(8), SWAP(28), ..., SWAP(29), SWAP(25), SWAP(1) 12

Below is one of the designs created by the application of a DM operator in Probing Mode (by swapping the two tubes 13
following tube 29 in Design#1.8) 14

Generation 2: 15

Design #2.13: 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 16
41 27 43 46 29 31 47: **Capacity=5.2093** 17

and 14 others. 18

Selected Members: 6, 15, 11, 3, 13, 1, ... 19

. 20

The program soon shifts into Learning Mode: 21

Generation 6: Learning mode 22

Learned rule: 23
[x1.x2.x3.x4.x5.x6.x7.x8.x9.x11.x12.x13.x14.x15.x17.x18.x19.x20.x21.x22.x23.x24.x25.x26.x27.x28.x29.x30.x31.x32. 24
x33.x34.x35.x36.x37.x38.x39.x40.x41.x42.x43.x44.x45.x46.x47.x48=regular] & [x10=outlet]&[x16=inlet] (t:7,u:7,q:1) 25

An example of a generated design: 26

Design #6.1: 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 27
43 44 28 46 14 32 16: **Capacity=5.5377** 28

. 29

Below is a design from the 22st generation: 30

Generation 22: Learning mode 31

Design #22.15: 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 32
26 43 28 27 29 14 48 16: **Capacity=5.5387** 33

and 14 others. 34

Selected Members: 11, 4, 4, 13, 15, 10, 12, 13, 15, 15, 12, 2, 3, 5, 10. 35

. 36

ISHED1 continues to evolve designs, and finally achieves: *37*

Generation 41: 38

Design #41.15: 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 39
40 26 28 35 30 14 48 31: **Capacity=6.3686** 40

Figure 16. An annotated excerpt from the log of an ISHED run.

The first part of the log, lines 1-4, provides the user with a summary of the parameters under which the program was run. Here we see that ISHED was creating designs of evaporators consisting of 3 rows of 16 tubes. The population size was 15, and the program was set to evolve 41 populations. In Probing Mode, the Operator Persistence parameter instructed ISHED to sequentially apply each Design Modification operator to the design being modified up to five times to determine a feasible new

design. If none was found, a different DM operator would be applied to this design. The two Mode Persistence parameters, #Probing and #Learning, instructed ISHED to shift from Probing to Learning Mode when two consecutive Probing Mode generations failed to provide improvement in either the highest fitness design or the average fitness in the population, and to shift back to Probing Mode after one Learning Mode generation did not provide such improvement.

The second part, lines 5-10, shows two of the fifteen designs generated by the program for the initial population, along with their evaluated capacities. The first of those, Design #1.3, was selected according to the elitism principle (it was the best design generated so far). The log shows (line 11) that the first seven of the fifteen members of the new (Generation 2) population would be built from Designs 1.3, 1.2, 1.3, 1.7, 1.9, 1.3, and 1.9, respectively.

Each seed for the new population then had a design modifying operator applied to it as specified in line 12: Design #2.1 was created by applying operator NS(23,39) (change the source of refrigerant for tube 23 from whatever it was to tube 39) to Design #1.3; Design #2.2 was created by applying operator SWAP(8) (swap the positions of the two tubes preceding tube 8) to Design #1.2; etc. It is shown in lines 16 and 17 that Design #2.13, generated by applying operator SWAP(29) to Design #1.8, had a capacity of 5.2093.

Because after two consecutive generations there was no improvement in either the best capacity or the population's average capacity, ISHED switched to Learning Mode, and discovered a rule (lines 23-25) that indicated a pattern in which high-performing designs consisted of an outlet at position 10, an inlet at position 16, and interior tubes at all other positions. The learned rules were instantiated to become members of the new population, such as Design #6.1 (lines 27-28).

The run continued in this manner, and at its halfway point (lines 31-35), true progress in evolving better designs had not emerged. But by the run's end (lines 38-40), there was a significant leap in design quality. While the best design in the initial population had a capacity of 5.5376 kW, the best

design after 40 generations in running ISHED had a capacity of 6.3686 kW, which represents a 15% increase. This result clearly demonstrates the LEMd capability for improving evaporator designs.

Similarly, Figure 17 shows a short excerpt from the log of an ISCOD condenser optimization run. This log reads similarly to the one shown in Figure 16, and for brevity only includes elements from the first and last generations. In this run, performance improved from an initial best of 2.0395 kW to 2.90202 kW, an improvement of 42% in 50 generations.

```
Condenser Size: 15 x 3
Population Size: 25          Number of Generations: 50
Operator Persistence: 5          Structure Probe: 500
Mode Probe Sizes --   #Probing: 2   #Learning: 2

Initial population:

Design #1.1: 2 17 4 19 6 21 O 7 8 9 10 11 12 13 14 1 18 3 5 5 36 23 38 25 40 27 42 29 44 15 16
             31 32 35 20 37 22 39 24 41 26 43 28 45 30:            Capacity=2.0395

Design #1.2: 2 3 18 5 20 7 22 9 24 11 26 13 28 O 14 17 1 33 4 35 6 37 8 39 10 41 12 43 30 15
             16 31 19 19 36 21 38 23 40 25 42 27 44 45 29:         Capacity=0

Design #1.3: 2 17 4 19 6 21 8 23 10 25 O 11 12 13 14 1 18 3 20 5 22 7 24 9 40 27 42 29 44 15
             16 31 32 33 34 35 23 37 38 41 26 43 28 45 30:         Capacity=1.86314

. . .

Generation 50:

Best so far (from Generation 49):
17 21 18 5 20 22 10 10 8 O 9 23 15 45 12 2 3 6 4 24 37 38 11 39 7 27 43 30 28 44 16 31 32 19 1
             34 36 41 25 42 40 26 29 14 13:                        Capacity=2.90202

Design #50.1, obtained from applying SWAP(37) to the best so far:
(this becomes the first element of the new generation):

17 21 18 5 20 22 10 10 8 O 9 23 15 45 12 2 3 6 4 24 37 38 11 39 7 27 43 30 28 44 16 31 32 36 1
             19 34 41 25 42 40 26 29 14 13:                        Capacity=2.90203

Structures obtained by applying DM operators to other members of the population:

Design #50.2: 3 21 18 20 4 34 8 O 40 26 7 11 O 28 12 2 1 6 23 39 37 36 41 25 10 27 43 13 44 14
             32 16 31 19 17 38 22 5 24 42 9 8 29 30 15:            Capacity=2.6719


…
Best performer:

17 21 18 5 20 22 10 10 8 O 9 23 15 45 12 2 3 6 4 24 37 38 11 39 7 27 43 30 28 44 16 31 32 36 1
             19 34 41 25 42 40 26 29 14 13:                        Capacity=2.90203
```

Figure 17. An annotated excerpt from the log of an ISCOD run.

Consecutive LEMd runs building upon one another produced improved designs. For example, one experiment initially performed five independent ISHED runs given a moderately uneven airflow

pattern starting with heuristically generated populations of size 25. All five runs executed 50 generations of evolution, and the best design from each of these runs is shown in Figure 18, which displays them in order of increasing capacity. In this and Figure 19, only the connections between tubes, projected onto one side of the evaporator, are indicated; the connections alternate between sides of the evaporator.

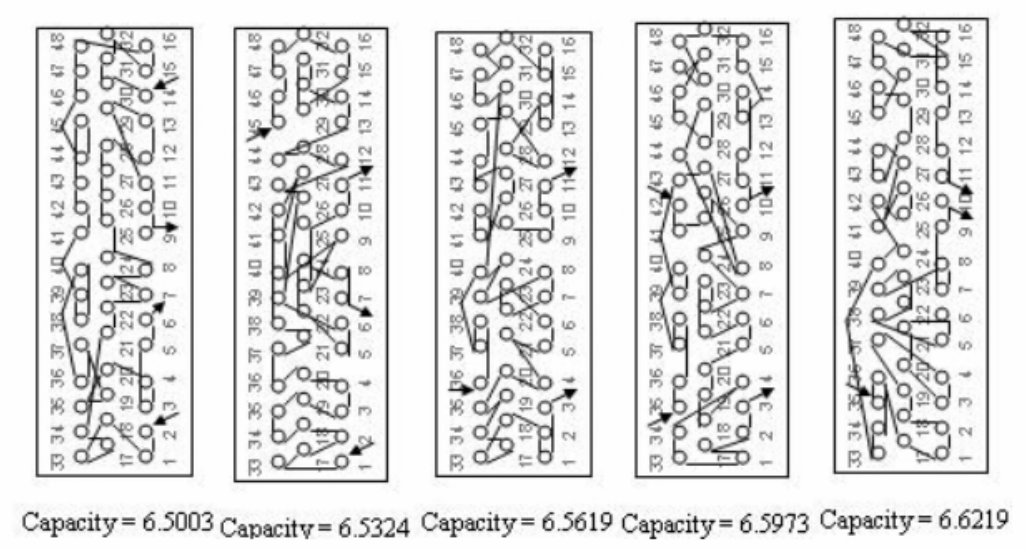

Figure 18. Best designs from five separate ISHED runs using the same parameters.

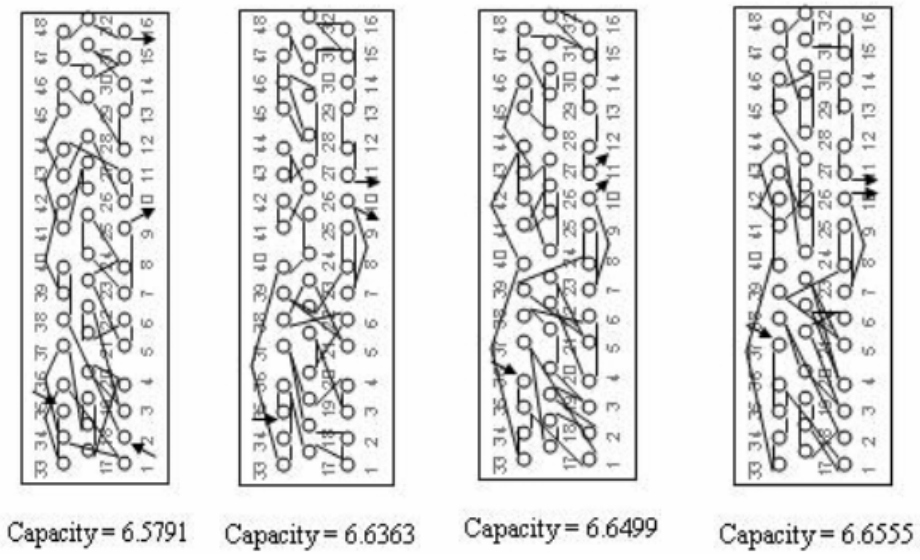

Figure 19. Best designs from four subsequent ISHED runs, each building upon the previous ones.

Although these five designs have relatively similar capacities, the configurations of their tubes differ significantly from each other. For instance, the first, second and fourth design have two inlets (indicated by inward-pointing arrows) and two outlets (indicated by outward-pointing arrows), and the third and fifth have a single inlet whose path splits, resulting in two outlets. The locations of the inlets and outlets also differ.

In the four subsequent runs, the population size was doubled (increased to 50), and the number of generations was raised to 150. The initial population for the first of these runs was seeded with three copies of each of the five designs shown in Figure 18, and the other 35 designs were generated heuristically according to the table mentioned in Section 4.2 Figure 19a shows the best design produced by this run. The initial population of the second run of the four had the same fifteen designs as the first run, plus two copies of the best design from the first run. The best design produced by the second run is shown in Figure 19b. After adding two copies of that best design to the initial population for the third run, ISHED's next run produced the design in Figure 19c. And after two copies of that design were added, the design in Figure 19d, with a capacity exceeding 6.65 kW, was generated during a fourth run. The capacity of this last best design is better than that of the best design (capacity = 6.20 kW) in the five initial populations by over 7%. The obtained design was evaluated as better than best human designs.

The connections in the designs in Figure 19 appear to be more complicated than those in Figure 18. If these were to be considered for manufacturing, they could be additionally fine-tuned by hand [15], to make them fully viable. These experiments show that ISHED was able to produce consecutively better designs in subsequent runs of evolution.

6 Related Work

LEMd represents to the authors' best knowledge a novel and original method for optimizing complex engineering systems through evolutionary computation. It is based on Learnable Evolution Model (LEM), which strives to model evolution guided by an "intelligent mind" rather than evolution that

proceeds without such guidance. The approaches closest to LEM appear to be cultural evolution algorithms that execute a process of dual inheritance (e.g., [16,17,18]). Unlike LEM, cultural evolution works at a "micro-evolutionary level," which involves individuals described by traits and modified by conventional evolutionary operators, and a "macro-evolutionary" level, in which individuals generate "mappa" representing generalized beliefs that are used to modify the performance of individuals in the population. LEM is significantly different from cultural evolution algorithms in both the way the learning process is implemented and in the way its results are used during evolutionary computation.

A system somewhat related to LEMd in its general goals is GADES[19], a "generic" evolutionary design system that aims to serve as a method for application to diverse design problems, and has been applied to such problems as furniture, heat sink and building design. The system employs a conventional Darwinian evolutionary system, but allows the user to specify the exact nature of the genetic operators to apply. The system also provides a means for the user to specify the representation of individuals in the population, and how they translate to real-world objects, as well as to provide a fitness evaluation function. LEMd also has such capabilities, but it includes also an entirely new form of generating individuals provided by Learning Mode.

Some research has explored a tight coupling of symbolic learning with Darwinian operators. Sebag and Schoenauer applied AQ-type learning to adaptively control the crossover operation in genetic algorithms[20]. In their system, the rules are used for the selection of the crossover operator. Sebag, Schoneauer and Ravise used inductive learning for determining mutation step-size in evolutionary parameter optimization[21]. Ravise and Sebag described a method for using rules to prevent new generations from repeating past errors[22], and in a follow-up work, Sebag, Schoenauer and Ravise proposed keeping track of past evolution failures by using templates of unfit individuals, called "virtual losers" and using an evolution operator to create individuals different from them[23]. Grefenstette developed a genetic learning system, SAMUEL, that implements a form of Lamarckian evolution[24]. The system was designed for sequential decision making in a multi-agent environment. A strategy, in the form of *if-then* control rules, is applied to a given world state and certain actions are

performed. This strategy is then modified either directly, based on the interaction with the environment, or indirectly by changing the rules' strength within the strategy. The changes in a strategy are passed to its offspring. This is a Lamarckian-type process that takes into consideration the performance of a single individual when evolving new individuals.

Another approach that extends the traditional Darwinian approach can be found in the GADO algorithm[25]. GADO is an evolutionary algorithm developed for complex engineering problem optimization. It differs from traditional genetic algorithms primarily in the way new individuals are generated. It uses five different crossover operators, three of which, *line crossover*, *double line crossover* and *guided crossover*, were introduced in GADO. However, unlike LEM, the algorithm does not create any generalizations of the current population, and therefore is significantly different.

Yet another unconventional form of evolutionary computation is implemented in *memetic algorithms*, in which genetic crossover operators are combined with local optimization via simulated annealing[26]. Memetic algorithms can be quite effective for some optimization problems; they have been applied to many optimization problems, among them the optimization of communication network design[27]. Memetic algorithms differ significantly from LEMd, as they do not use machine learning to guide the evolutionary design process.

7 Conclusion

Darwinian-type evolutionary computational methods are frequently applied to optimizing very complex designs when standard optimization methods are inadequate. For example, these methods have been used for aircraft wing design[7] and drug design[28]. Many other applications of conventional evolutionary methods have been catalogued by Bentley[8]. Because the innovation is done through random mutations and/or recombinations, rather than by the guidance of an "intelligent mind" these applications usually suffer from low efficiency, and may require a prohibitively large evolution length when optimizing complex designs. Thus, they may be particularly unsatisfactory for optimization problems in which fitness function evaluation is time-consuming and/or costly.

In contrast to conventional, unguided evolutionary computation, LEM attempts to model an intellectual evolution that governs the development of human artifacts. As mentioned earlier, in such evolution, the generation of new populations is based on the results of human designers' analysis of the advantages and disadvantages of past populations. LEM approximates this process by generating new designs through hypothesis formation and instantiation, and has proven to be very effective in reducing the evolution length. Early applications of LEM include the design of very large neural networks[29], digital signal filter design[30], and energy distribution design[31].

The LEMd method, described herein, tailors the LEM methodology to complex design problems. Two specialized LEMd implementations, ISHED and ISCOD, for optimizing tube arrangements in evaporators and condensers in heat exchangers have proven to be very powerful tools for this purpose. They were able to evolve designs satisfying given technical and environmental constraints that were better or comparable to the best human designs used in the cooling industry according to experts[15].

In the case of uniform airflow, roughly symmetric designs were generated, but in the case of uneven airflow, more elaborately configured designs evolved. These results highlight LEMd's potential. ISHED and ISCOD represent, however, only rudimentary implementations of the LEMd methodology, and suffer from various limitations. Neither is capable of an orderly instantiation of many of the possible design specifications consisting of three or more inlets or outlets. The complexity of the instantiation process makes it sometimes difficult to inject a sufficient amount of diversity in the new population. This makes Probing Mode particularly important in the current implementation of the ISHED and ISCOD systems.

Experiments have also revealed a weakness in these systems regarding an occasional lack of evolutionary advancement in seeded initial populations. There exist ways to work around the problem, such as short, sequential runs building upon each other's results. Further research will explore more thoroughly the nature of the experienced problems.

The LEMd method is at an early stage of development, and poses many challenging research problems. They include the theoretical and experimental investigation of change operators in Learning Mode, testing LEMd in different application domains, and extending it to optimization problems under complex constraints and to dynamic landscapes. The experience with heat exchanger optimization has indicated another important topic for future research, namely, the need for developing methods for efficiently instantiating learned rules in a search space that is constrained in many complex ways.

Another topic involves the automatic encoding of design tasks into representations and design modification operators that LEMd can utilize. LEMd could also be improved by developing a language for representing design modification operators and constraints, so that arbitrary design domains can be specified for LEMd application.

Regardless of the current limitations, the experiments described here have demonstrated that LEMd can already serve as a powerful tool for assisting human designers in optimizing complex systems, especially those in which fitness function evaluation is costly or time-consuming. Because experimental studies of learnable evolution have demonstrated that its advantage over Darwinian-type evolutionary computation grows with the complexity of the problem[5], the LEMd approach appears to be particularly promising for optimizing very complex engineering systems with a large number (on the order of hundreds or more) of controllable discrete and/or continuous parameters.

Acknowledgments

The authors thank Dr. Piotr Domanski from the National Institute of Standards and Technology for introducing us to the domain of heat exchanger design and for providing simulators for evaluating evaporator and condenser designs. He and David Yashar have also provided consultation and feedback in the development of ISHED and ISCOD. Special thanks go to Janusz Wojtusiak for valuable comments on this paper, and his work on development of the LEM3 system that represents the most advanced implementation to date of the LEM methodology. We also thank Jaroslaw Pietrzykowski for his careful review and insightful comments on this paper. We are grateful to John Ashley, Scott

Mitchell, Hugo de Garis, and the team of Mark Coletti, Tom Lash, Craig Mandsager and Rida Moustafa, whose early experimental LEM applications to various practical problems have provided an encouragement and stimulus for the development of LEMd.

Theoretical research that provided a basis for the development of the LEMd method and the ISHED and ISCOD implementations has been conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research has been supported in part by the National Science Foundation under Grants No. IIS-0097476 and IIS-9906858, and in part by the UMBC/LUCITE #32 grant.

References

1. Michalski R. Learnable evolution: Combining symbolic and evolutionary learning. Proc.4th Intl Workshop on Multistrategy Learning, 1998. pp.14-20.
2. Michalski R. Learnable evolution model: Evolutionary processes guided by machine learning. Machine Learning. 2000, 38:9-40.
3. Kaufman K, Michalski R. Applying learnable evolution model to heat exchanger design. Proc.12th Intl Conf on Innovative Applications of Artificial Intelligence, 2000. pp. 1014-1019.
4. Wojtusiak J. The LEM3 implementation of learnable evolution model: user's guide. Reports of the Machine Learning and Inference Laboratory, MLI 04-5, George Mason University, Fairfax, VA, 2004.
5. Wojtusiak J, Michalski, R. The LEM3 system for non-Darwinian evolutionary computation and its application to complex function optimization. Reports of the Machine Learning and Inference Laboratory, MLI 05-2, George Mason University, Fairfax, VA, 2005.
6. Michalski, R, Kaufman, K. The AQ19 system for machine learning and pattern discovery: A general description and user's guide. Reports of the Machine Learning and Inference Laboratory, MLI 01-2, George Mason University, Fairfax, VA, 2001.

7. Oyama A. Multidisciplinary optimization of transonic wing design based on evolutionary algorithms coupled with CFD solver. European Congress on Computational Methods in Applied Sciences and Engineering. 2000.

8. Bentley P, editor. Evolutionary design by computers. Menlo Park, CA: Morgan Kaufmann, 1999.

9. Bentley P, Corne, D. Creative Evolutionary Systems. San Francisco, CA: Morgan Kaufmann/Academic Press, 2002.

10. Siddall J. Optimal engineering design – Principles and applications. Marcel Decker, 1982.

11. Michalewicz Z. Genetic algorithms + data structures = evolution programs. London: Springer-Verlag, 3rd ed. 1996.

12. Domanski P. EVSIM-An evaporator simulation model accounting for refrigerant and one dimensional air distribution. NISTIR 89-4133, 1989.

13. Michalski R. Attributional calculus: A logic and representation language for natural induction. Reports of the Machine Learning and Inference Laboratory, MLI 04-2, George Mason University, Fairfax, VA, 2004.

14. Michalski R. A theory and methodology of inductive learning. In Michalski R, Carbonell J, Mitchell T. editors, Machine learning: An artificial intelligence approach, Palo Alto: Tioga, 1983. pp. 83-134.

15. Domanski P, Yashar D, Kaufman K, Michalski R. An optimized design of finned-tube evaporators using the learnable evolution model. Intl. J Heating, Ventilating, Air-Conditioning and Refrigerating Res. 2004, 10:201-211.

16. Reynolds RG. An Introduction to Cultural Algorithms. Proc. 3rd Annual Conf on Evolutionary Programming, 1994. pp. 131-139.

17. Rychtyckyj N, Reynolds R. Using cultural algorithms to improve performance in semantic networks. Proc. Congress on Evolutionary Computation. 1999, pp. 1651-1663.

18. Saleem S, Reynolds R. Cultural algorithms in dynamic environments, Proc. Congress on Evolutionary Computation. 2000. pp.1513-1520.

19. Bentley P. From coffee tables to hospitals: Generic evolutionary design. In Bentley P, editor, Evolutionary design by computers. Menlo Park, CA: Morgan Kaufmann, 1999. pp.405-423.

20. Sebag M, Schoenauer M. Controlling crossover through inductive learning, Proc. 3rd Conf on Parallel Problem Solving from Nature. 1994, pp. 209-218.

21. Sebag M, Schoenauer M, Ravise C. Inductive learning of mutation step-size in evolutionary parameter optimization. Proc. 6th Annual Conf on Evolutionary Programming. 1997, pp. 247-261.

22. Ravise C, Sebag M. An advanced evolution should not repeat its past errors. Proc. 13th Intl Conf on Machine Learning, 1996, pp. 400-408.

23. Sebag M, Shoenauer M, Ravise C. Toward civilized evolution: Developing inhibitions. Proc. 7th International Conference on Genetic Algorithms. 1997, pp.291-298.

24. Grefenstette J. Lamarckian learning in multi-agent environment. Proc.4th Intl Conf on Genetic Algorithms. 1991, pp. 303-310.

25. Rasheed K. GADO: A genetic algorithm for continuous design optimization. Ph.D. diss. Technical Report DCS-TR-352, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1998.

26. Moscato P. Memetic algorithms: a short introduction. In Come D., Dorigo M., Glover F., Dasgupta D. Moscato P., Poli R., Price K.V. editors, New ideas in optimization, Maidenhead, England: McGraw Hill, 1999. pp. 219-234.

27. Runggeratigul S. A memetic algorithm for communication network design taking into consideration an existing network. In Resenda M.G.C., Sousa J.P. editors, Metaheuristics: Computer decision-making, Boston: Kluwer Academic Publishers, 2004. pp. 615-626.

28. Embrechts M, Ozdemir M, Lockwood L, Breneman C. Bennet K, Devogelaere D, Rijkaert M. Feature selection methods based on genetic algorithms for in silico drug design. In Fogel G, Corne D. editors, Evolutionary computation in bioinformatics, San Francisco: Morgan Kaufmann, 2002. pp. 317-340.

29. de Garis H, Korkin, M. The cam-brain machine (CBM): An FPGA based hardware tool which evolves a 1000 neuron net circuit module in seconds and updates a 75 million neuron artificial brain for real time robot control. Neurocomputing 2002: 42:35-68.

30. Coletti M, Lash T, Mandsager, C, Michalski R, Moustafa, R. An experimental application of the learnable evolution model system LEM1 and genetic algorithms GA1 and GA2 to parameter identification in digital signal filter design. Reports of the Machine Learning and Inference Laboratory, MLI 99-5, George Mason University, Fairfax, VA, 1999.

31. Ashley J. Experiments with LEM3: Insights into generator electricity market competition between two heterogeneous agents using a game theoretic approach. Reports of the Machine Learning and Inference Laboratory, George Mason University, 2006 (to appear).