Third Oregon Symposium on Logic, Design, and Learning
Portland State University
Portland, Oregon
May 22, 2000

Complete Text for Presentation

Title: The Creation of a Cybernetic (Multi-Strategic Learning) Problem-Solver: Automatically Designed Algorithms for Logic Synthesis and Minimization

By: Karen M. Dill and Marek A. Perkowski
e-mail: dills@worldnet.att.net / mperkows@ee.pdx.edu

Slide 1:
Introduction

Slide 2:

This presentation is about the development of automatically designed algorithms for both logic synthesis and minimizations. But, before beginning to describe this algorithm, I would like to briefly discuss the history and origins of this research and then, some of our preliminary results.

A few years ago, both my own group and that of Drs. Debnath and Sasao investigated the minimization of the GRM (Generalized Reed-Muller) logic form. (This is a canonical type of two-level, AND-EXOR logic.) But these two groups, performing research independently of one another had very different approaches to the problem.

The Debnath/Sasao group implemented an extensive rule-based system, constructed from human design methods. In contrast, our (Dill/Perkowski) method was entirely based on artificial evolution, in which designs are produced automatically, with no human logic design rules applied.

The features of these different implementations can be compared. The rule-based system required extensive human development time and research to construct, whereas, the evolutionary based system required only minimal time to define the application of the Genetic Algorithm (in this case) for the general problem.

The results of the different approaches are that the rule-based design produces smart, often exact solutions, may be applied to either large or small benchmarks, and has a fast computation for results. In contrast, the evolutionary-based design produces good, competitive solutions, for smaller benchmarks, and as characteristic for GA's has a slow computation time.

So this illustration has given us some issues to consider… How should we measure the merit of the rule-based as compared to the evolutionary-based algorithm design? Which is cheaper? More practical? Is this defined by the run-time, man-hours of development time, quality of results, skills of the developer, number of algorithm applications, end-user (pure research at a university versus quick application in industry), etc.? If a more general problem-solving technique for automatically designed algorithms

for logic synthesis and minimization is truly sought, what lessons can be learned from these examples?

Slide 3:

This chart illustrates a comparison of the results from the rule-based (GRMIN2) and evolutionary-based (GA-GRM) GRM minimizers for several small benchmarks. The names of the benchmarks are given along the X-axis, with first the number of input and then the number of output variables in parenthesis. The number of terms, resulting from the minimization is given along the Y-axis. (The benchmarks are arranged in order of a rough complexity, calculated as the number of inputs multiplied by the number of outputs.)

It is shown that for a number of these benchmarks the results are comparable. However, as the size of the benchmark and number of terms increases, the GRMIN2 (rule-based algorithm) achieves minimization results with fewer terms.

Slide 4:

Our group has worked in the field of artificial evolution, utilizing both Genetic Algorithms and Genetic Programming, as applied to logic synthesis and minimization, for several years. Hence, through this research, we have gained a "feel" or intuitive viewpoint of the application of these mechanisms toward logic problems. Our general experience is here illustrated… Note that these relations are not based on numerical data and that the rule-based approach is a general grouping of all such approaches, not specifically indicating the GRMIN2 of Drs. Debnath and Sasao.

This chart graphs the development time versus quality of solution for both general rule-based approaches and that of the Genetic Algorithm.

First, with the Genetic Algorithm, once the problem has been defined and specified, the development time is quick. Regardless of the application, the same code and mechanisms may be applied, only changing the problem size. The sloping line indicates a long run-time, which is proportional to the size of the problem. Finally, a quasi-optimal solution is determined and it becomes difficult to improve upon it, regardless of run-time, using these same standard operators.

In contrast, the general rule-based approaches require extensive development time, as humans must first determine which logic rules are necessary and then implement these in software. The run-time is quick for such customized approaches and the results may be improved with more problem analysis.

Slide 5:

This chart illustrates the difference between the minimal and algorithmic solutions as problem complexity is increased. For problems of low to medium complexity, the results obtained by both approaches may be comparable. However, with increased complexity, the customized, application specific rule-based design excels, whereas the very general and blind mechanism of nature illustrated with the GA cannot achieve this level of specialization.

Slide 6:

From this experience we can make some generalizations about the different types of algorithm designs. Here, the X-axis plots the type of learning from the viewpoint of least to most intensive of human thought. The Y-axis plots the ease of implementation from a human viewpoint.

Recalling the illustration of the GRM Minimization problem approaches, it is shown that these algorithms have diametrically opposed design styles. The GA can be here classified as an automatic type of machine learning, implemented with artificial evolution, with some elements of randomness. In contrast, the rule-based design approach, as a customized, application-specific algorithm utilizes intense human learning for determining the set of logic rules. As we have seen, each of these design styles illustrates different elements of merit.

Hence, in seeking a new algorithm for automatically designing algorithms for classes of logic synthesis and minimization problems, we seek to incorporate elements of both types of design methodology. Then, perhaps the strengths and benefits of each approach can be obtained.

Slide 7:

Here, the need for a new design approach may be summarized…

In this research, we propose a new Cybernetic Problem Solver (CPS). The goal of this software is to automatically design application specific algorithms for classes of problems. The application of the software is for binary and MVL synthesis and minimization problems, as well as robotics.

The reason why the CPS is necessary, is that it provides capabilities beyond that of other current methods.

- The current human-designed expert systems work well, but require long development times. Further, they are usually limited to a single type of application.
- Functional Decomposition may find good solutions, but it is not easily tunable to all technologies.

Slide 8:

The evolutionary algorithms (GA/GP) also present some difficulties:

- Time and Size Limitations: Bigger problems take a lot longer and may not be practical.
- Quasi-Optimal Solutions: The results may be a local optimum in the search space.
- No Guarantees: There is no assurance that the optimal solution will be found.
- No Explanation: There is no explanation of solution choices, except from the fitness function and some random number selections. Further, there is no description of a design methodology or transferable rules of generalization.
- No Optimization Approach for Problem Class: The GA learns specific problems, rather than a problem classification. The learning is hence non-transferable, and the results may not be generalized. As such, a set of rules for a problem class is not developed; each problem must be run individually, with no relation to others.
- Examples of our experience with GA are given for the GRM logic minimization problem. (We also have experience in using GP for logic synthesis.)

The traditional pure search strategies are comprehensive (sure to find a solution) and have short development times, but are highly resource (memory, etc.) and time intensive. Hence, they are not usually practical for real-world applications.

Slide 9:

In summary, the difficulties with the solution approach mechanisms for each method produce unsatisfactory search strategies for a general problem solving technique. This applies to both the Complete (entire state-space) Search and the Incomplete (evolutionary and rule-based) Search classifications. (Leaving no problem-free methods…)

Hence the CPS Algorithm incorporates all the previous techniques:
> Pure and Heuristic Search Strategies,
> Problem Solving/Learning Paradigms

This selection of all the learning techniques is done in expectation that the CPS will build on the strengths, and thus reduce the weaknesses of the individual methods.

Slide 10:

The answer to how such a Cybernetic Problem Solver general algorithm can be created lies in a summation of the two opposing design techniques presented earlier. Herein, human expertise is combined with (automatic) search mechanisms to produce efficient problem-solving methods.

By applying the human expertise or know-how, we don't waste time using search mechanisms (be they evolutionary or other types) to re-invent known methods and solutions.

Hence, the Cybernetic Problem Solver is:
- A multi-strategic design approach,
- It is Intelligent in that problem analysis is performed, followed by decisions,
- Because the solution may be formed with a number of techniques or mechanisms, the algorithm may be viewed as a superset "toolbox" of methods,
- Further, game concepts are employed as these search methods are combined with both competition and cooperation,
- Herein, various learning methodologies are combined, to create a general application algorithm.

Slide 11:

As previously discussed, the CPS is different from conventional rule-based design approaches. It is also quite different from that of the traditional Genetic Algorithm. This point is illustrated with the GRM form (AND-XOR) logic minimization problem.

The GA is a very flexible and generic process that can be applied to any formulated problem. Here, for each GRM minimization, the GA is given a polarity vector input describing the particular problem for which a minimized solution must be found. This process is then repeated in its entirety for each polarity vector. Hence, the GA learns specific problems. But, no information, process, or rules are ever formulated that can be applied more widely, even to instances of the same problem.

In contrast, the CPS works differently, by combining the best features of the GA and true machine "learning" to actually "generalize" information from examples. Again,

for the GRM minimization problem, polarity vectors are given as input. The CPS then acquires knowledge from these polarity vectors as it learns the problem class. Following sufficient learning examples or training sets, a program is developed that "optimally" solves all problems of this class. The chromosome contains a good strategy as to how to approach other problems of the same class. (The CPS also selects a balance between search methods and human logic "tricks" or methods.) Hence, for any new polarity vector of this type, a solution can be given. The CPS generalizes information and learns classes of problems.

Slide 12:

Two examples of the application of the CPS Algorithm will here be given for typical CAD problem types.

The CPS was first applied to the Set-Covering Problem. The Set-Covering Problem is a column minimization problem, often seen in logic design (for example, PLA minimization, test minimization, multi-level design, robotic CIM minimization, security, image processing, layout, etc.)

In this problem, a covering table with a number of rows and columns, each associated with a cost, is given.

-The elements of Set R are the rows of the table.

-The elements of Set C are the columns of the table.

-Row r^i covers column c_j, with the relation $COV(r_i,c_j)$

The goal of this problem is to find a set of rows that cover all the columns at a minimal cost.

Slide 13:

Here is a small example of a set-covering problem, illustrated with a table of rows and columns. The problem is to find the smallest subset of rows that will cover all the columns. Let us work this small problem by hand.

First, examining the rows, R2 is necessary as it is the only row that covers column C1. Row 4, R4, is also necessary, as it is the only cover for column 6, denoted as C6. The other columns have several options for covering rows, as given by the logic expressions. Column C2 is covered by R1 or R3 or R5. Column C3 is covered by R1 or R2 or R3. Column C4 is covered by R4 or R5. C5 is covered by R1 or R4.

Hence, in constructing a decision function, R2 and R4 (shown in bold) are necessary terms (for coverage of C1 and C6). And, the logic expressions for Columns C2-C5 are ANDed into the decision function, which is set equal to "1", since it includes all solutions.

The exact minimal solution can be hand-calculated by observation. The smallest set to satisfy the solution, must cover all columns. R2 and R4 are necessary for covering columns C1 and C6, but in total, they cover C1, C3, C4, C5, and C6. Hence, it is only necessary to add a single row term to cover C2. C2 is covered by the expression r1 + r3 + r5. Hence, either R1 or R3 or R5 may be used to cover C2 and any of these terms may be paired with the necessary terms. Hence the solutions are {r2,r4,r1}, {r2,r4,r3}, and r2,r4,r5}.

Note that this is a very difficult application for a GA or a Greedy Algorithm, since application specific rules are not considered. Using dominance and other rules, together

with the many smart (human heuristic) methods that already exist, can produce much better solutions for this common logic problem.

Slide 14:

The preliminary results of testing the CPS over a set of various tables gave the following results:

- The Equivalence/Dominance/Indispensable conditions each reduce the search space 2-3 times. (This is an example of the human logic methods/rules introduced into the automatic algorithm.)
- Further, the experimental results compared with the initial random vector:
  - > Reduces the generated space by 50-200 times,
  - > Increases the computation time in the learning phase by 20-30%. (This is the price that must be paid for reducing the search space.)
  - > The vector of coefficients obtained, decreases the generated solution space size by 15-20% from that of the initial vector.

So, the results from combining human "logic" rules together with that of search mechanisms were quite good.

Slide 15:

Another example of the preliminary results of the application of the CPS algorithm is given.

Herein, the CPS is applied to the Productivity (or Assignment) Problem. This problem type has many CAD applications, including encoding.

Given are *n* machines and *n* workers, the productivity of the worker *i* on machine *j* is denoted wij.

The goal is to maximize productivity. For this, the assignment matrix $W_{nxn} = [W_{ij}]$ is given from which *n* elements must be selected in such a way that when any two of them are taken from different rows and columns, that the sum of the elements is maximum.

Slide 16:

The productivity problem can be naturally demonstrated as that of work assignment in a factory for maximizing production. Here is a small example, in which at the Toy Widget Workshop, the workers' production rates were tested for the assembly of various products. From this chart, it is apparent that to simultaneously maximize all product lines, Woozle production should be assigned to Paul, Wizzles to Mark, and Weazels to Susan. While these workers are all of about the same overall quality, (summing their skills for each product line), each job is assigned to the individual that demonstrates the highest skill at assembling that product.

Slide 17:

The process of utilizing the CPS in solving problems requires the numerical specification of the following parameters. The quantification of these parameters differs for each application.

- Quality Function for Operators,
- Cost Function for States, and

- Heuristic Quality Function for Nodes.

Here these parameters are shown for a "Runaway Robot" guidance problem. Let us suppose robot "R" is located at the blue "x" on our map of the engineering building. The goal is to get him back to the lab as quickly as possible, before it is noticed that he is missing. Thus, for the CPS the parameters in this problem can be defined as follows:

Quality Function for Operators – The measure of cost of operations needed to reach the goal. (For this example, it is which turn gets the robot closer to the destination.)

Cost Function for States – The total efficiency measure of reaching a particular state. (For this example, since all instructions are of equal cost, it is simply the number of instructions needed to get to a particular state.)

Heuristic Quality Function for Nodes – Considering all conditions, (traffic, allowable speed, etc.), which route is more efficient.

Slide 18:

Process

Returning to the Productivity Problem, the process for utilizing the CPS is the numerical specification of the:

- Quality Function for Operators: measure of cost of operations for production,
- Cost Function for States: measure of total efficiency for reaching a particular state,
- Heuristic Quality Function for Nodes: measure of most efficient production method, considering current conditions.

Results

The preliminary application of the CPS for this problem, over different sets of data gave the following results:

- The Ordered-Search Strategy was most efficient.
- The Stopping Moment learning technique (described later) was effective.
- The entire space generated by the system included:
  - > average of 600 nodes,
  - > 530 expansion nodes were sufficient for the method, and
  - > 420 nodes produced the optimal solution.

Slide 19:

The testing results illustrating the Productivity Problem using the CPS with different Learning Methods is illustrated in these two graphs. Graph (a) on the left details the dependence of the number of nodes, and Graph (b) the processing time on the problem size for strategies.

Here we can see that for this problem, the Ordered-Search Strategy is the most efficient.

Slide 20:

Now that the CPS has been introduced with a history of its origin and preliminary experimental results, a closer examination of the theory and methodology are next presented.

Again, the purpose of the CPS is to automatically design application specific solutions for classes of problems, in both binary and multi-valued logic.

The CPS takes a state-space approach to solution derivation for learning problem classes. Herein an assumption is made that any combinatorial logic problem can be solved by searching some space of known states (for example, the netlists being optimized).

The CPS solutions are based on an intelligent strategy; a problem analysis is made followed by decisions for deriving solutions. In contrast to tradition, machine learning methods (both human designed heuristics and state-space search) are combined with an optimized strategy.

Slide 21:

The internal organization of the CPS Algorithm has the following features:

- The standard input/output is a netlist,
- The internal data representation uses: trees, expressions, and netlists,
- Two types of trees are utilized:
  > The Search Strategy Tree, which is:
    -- Managed with tree properties and transformations,
    -- Described by conditions, relations, sorting functions, and strategy parameters
    -- Utilizes in-program learning, which is the dynamic modification of flags/coefficients
  > Solutions/Data Tree
    -- Customized to application, as solutions must be fit for the problem
    -- Subset of the "best" results; tree maps out the directed/instructed search
- The output data can be visualized with trees, decision diagrams, K-maps, or algebraic forms.

Slide 22:

A diagram of the CPS system architecture is shown here. First, a brief overview is given and then each of the components will be presented.

Most simply, input data is given to the general problem solver and a particular, derived problem solver search strategy (chromosome) gives the algorithm instructions as to how to perform a search for good solutions. The quality of solutions is evaluated with the cost vector. Following this evaluation, each component of the strategy and cost vectors are given as feedback to their respective learning schemes. The learning schemes make adjustments, following which, new partial vectors for the strategy vector are formed. The user may specify the strategy or (temporarily) over-ride the search with expert information. The various learning schemes here, compete for application to the problem (as the type of search strategy).

Shown here in bold, is the problem solver search strategy. This chromosome (bit string) is a vector of parameters and specifies a tree searching technique, over the set of all search methods. The chromosome contains the best learning strategy. This methodology is different than a GA or GP because it specifies a general program for a problem class and is not tailored to a specific problem instance.

Slide 23:

The learning schemes are shown in bold in this diagram. These learning schemes indicate the use of various learning strategies, such as the GA, Perceptron (NN), Functional Decomposition, Rule-Based, etc. (Note that the learning schemes are not limited to four methods.)

The strategy vector, providing feedback to the learning schemes, consists of a number of strategy sub-vectors for analysis of the learning methods. These sub-vectors may contain:

-Evaluating functions for operators,
-Evaluating functions for nodes of the solution tree,
-Evaluation of the Stopping Moment, and
-Probabilities of calling various subroutines or using parameters.

Slide 24:

Here, the strategy vector generation is shown. This component solves the conflict or competition between the learning/search schemes with game concepts, for example, best cost, voting, etc., to determine the winner. The search strategy is encoded in a chromosome or bit string.

Slide 25:

The selection of the search techniques (creating the search tree) involves game concepts and is a two-stage process.

First the different search strategies compete for efficiency, in building a new tree branch.

Within this process:

- Search methods may change during the "game", as strategy parameters are altered to improve future behavior.
- Performance (between the state-space search strategies) is compared over partial trees, reiteratively, until tree completion.

Second, the search strategies cooperate to obtain a high quality solution and thus build the whole tree.

- The final tree construction for a general algorithm for a class of problems, occurs after many learning examples.

Slide 26:

The cost vector component of the CPS learning architecture evaluates the proposed solutions as a vector of cost functions. It applies to the solution for each type of learning scheme. This cost function is for a multi-criteria optimization.

The Problem Class Understanding module indicates that knowledge is gained through the evaluation of a number of search strategies for finding a good method of solving a class of problems. This is based on a large sampling of training problems.

Slide 27:

The CPS Learning Methods are applicable for any state-space search. The methods employed are:

1. Weighted learning of the Evaluation Function

2. Learning the Stopping Moment
3. Learning through Functional Decomposition

Two examples and the basic theoretic framework for each of these methods will be given. First, the Set-Covering Problem utilizes Method 1. Next, the Productivity (Assignment) Problem is shown with Method 2.

Slide 28:

The first method is the Weighted Learning of the Evaluation Function. In this learning method, an evaluator constructor is formed. The system learns the criteria of selecting good operators by using a weighted and decomposed (multi-level) evaluation function and its weight coefficients. The search space is divided into good and bad regions with various methods.

So, total cost is the summation of the individual weights and their respective costs.

For each new problem, the program automatically learns and selects w_i's on the basis of part of the solution tree that has already been searched and the information gathered from previous problems.

Slide 29:

The second CPS learning method is determining the Stopping Moment. This process of determining the Stopping Moment is deciding when to stop the learning process. It is the point when search backtracking can be terminated, because a better solution is very unlikely to be obtained in the future.
This is determined by:
--Normalized shape diagram/signature analysis, or
--Calculating the actual vs. ideal conditions, for example the estimated probability of not loosing the optimal solution.

This diagram plots the number of subsequently generated solutions (number of expanded nodes) versus the function cost. It is assumed that shapes of these diagrams are the same for combinatorial data of the same type.

Slide 30:

Continuing with the learning of the Stopping Moment… The method effectiveness is based on the construction of a learning system that predicts the effects of further solution space expansion.

The <u>assumptions</u> in constructing such a system are that combinatorial problem data sets have:
- Similar improvement curve shapes, and
- Statistical dependency between the number of nodes expanded to find better solutions

What is <u>Given</u> is:
- Sufficiently large number of examples tested, and
- Normalization for comparison between improvement curves.

Slide 31:

As recounted at the beginning of this presentation, the CPS originated due to problems with pure heuristic searches, in our GA research. So, now more information about this research is given.

First, as discussed earlier, with the development of the GA-GRM, we utilized a Genetic Algorithm for the minimization of GRM forms.

As some background, a GRM is a Generalized Reed-Muller form for 2-level AND-XOR logic. It is a canonical expression, with complete polarity freedom allowed for all variables, in every term. An example is given as: $1 \oplus x_1 \oplus x_2' \oplus x_1'x_2$, as a GRM. Note that there exists only one term for each subset of variables.

The number of literals in a GRM is $n2^{(n-1)}$ and the total number of unique GRMs is $2^{n*2\wedge(n-1)}$, for *n* variable functions, so the search space for minimization is very large.

Slide 32:

Traditional GRM minimization is performed with complementary expansions and substitutions. These are the facts that $x = x' \oplus 1$ and $x' = x \oplus 1$.

For example, given f, if we change the polarity of the last term from $x_1x_2'x_3'x_4'$ to $x_1x_2'x_3'x_4$, the following mathematics result.

Slide 33:

By substituting the complementary expansions, $x = x' \oplus 1$ and $x' = x \oplus 1$, into f_2, the following mathematics results.

Solving $f = f_1 \oplus f_2$, the example is reduced from eight to five terms.

Slide 34:

For our research, developing the GA-GRM code, we utilized a Genetic Algorithm encoding for the GRM logic minimization problem. Given a GRM, each term must be unique in name and polarity. For example, a GRM for a three variable function, f(a,b,c) has the following possible terms and polarity vector encodings shown in the table. By definition, only one term selection is allowed per row. This problem formulation creates a good match between the GA and the logic minimization problem.

Slide 35:

The "bit string" polarity vector representation for a three variable function, f(a,b,c) is encoded as shown. This representation and its particular encoding describes the valid GRM given. Note that the partial bit vector abc=110 in the bit string represents the abc' term and the remainder of the encodings correspond similarly.

Slide 36:

The results of our research produced competitive minimizations for small benchmarks. However, the results were non-competitive for large functions, as the chromosomal encoding string was too long and cumbersome.

Further, the shortcomings of the GA approach (for both the GA-GRM and <u>all</u> GA applications) are that:

- The application of a pure GA doesn't incorporate logic rules/methods; instead, using the evolutionary mechanisms, it must automatically "re-invent" them.

- The approach is highly dependent on the cost function for all behavioral reinforcement.
- The GA considers all possible, rather than just reasonable solutions.

Slide 37:

The GA-GRM minimizer was compared with that of GRMIN2 (Debnath/Sasao). In contrast to our approach, the GRMIN2 is entirely a rule-based system.

An outline of the GRMIN2 is taken from Debnath/Sasao's paper.

(i) The input is a PSDRM or a SOP.
(ii) Simplifies multiple-output functions.
(iii) Uses eight rules iteratively to reduce the number of products.
(iv) Modifies the cubes repeatedly by replacing a pair of cubes with another one, while keeping the array to represent a GRM.
(v) When reduction of the number of products becomes impossible in the iterative improvement mode, it temporarily increases the number of products and simplifies again.
(vi) Obtains lower bounds on the number of products in GRMs and often proves the minimality of the solution.

Slide 38:

The GRMIN2 Simplification Rules 1-4 are given.

Slide 39:

The GRMIN2 Simplification Rules 5-8 are given.

Slide 40:

This small example is here given to illustrate the GRMIN2 Rule 4, "X-EXPAND-1". (This rule is usually applied to binary logic, but is also applicable to multi-valued logic, as shown here for brevity.)

The other GRMIN2 rules are applied in a similar manner.

Slide 41:

In creating the CPS, we can learn lessons from that of the GRMIN2. The GRMIN2 Rules are a subset of ESOP rules for the GRM. These rules are human formulated, based on experience with logic. But, the CPS may be an improvement over the GRMIN2 by: (1) automatically selecting other rule subsets and/or strategies, (2) combining these human rules with automated search processes, and (3) determining an optimal application/combination of these rules.

Further testing of the CPS is necessary to determine the results and make comparisons to other software.

Slide 42:

Let us now summarize our experience with genetic operators versus human rules. This illustrates why the CPS combines automatic learning with human experience.

Genetic Operators are:

- A simple and practically blind mechanism of Nature.
- Have an elegance of design.
- Have universal applicability.

- Understand Nature as pure theory of Darwin; Although, incorporation of Baldwinian or Lamarckian Learning is possible.

In contrast,

Human Rules (Thought):

- Operates at a higher level, with multiple goals and much complexity.
- Logic algorithms are optimal and/or mathematically sophisticated.
- From human rules a high quality of learning results: this includes knowledge generalization, discovery, no over-fitting, and small learning errors.

In short, we have an immense amount to understand before we can duplicate human learning/thought in machines. So, for now the best option is to incorporate both automated search and human expertise into our processes.

Slide 43:

In conclusion, based on our past research with GA/GP applications and experience with logic synthesis/minimization, the CPS (Cybernetic Problem Solver) was developed.

- The CPS is a new approach for general problem solving and learning, for classes of binary and MVL optimization problems.
- It combines search strategies, through both competition and cooperation.
- The CPS is essentially an intelligent, superset "toolbox" of methodologies.
- It utilizes game concepts that build on the strengths, while reducing the weaknesses of different learning methodologies.

Slide 44:

- New learning methods are utilized in the CPS. These are:
  - > Weighted learning of the evaluation function, and
  - > Determination of the Stopping Moment.
- Initial CPS representative applications on combinatorial logic included:
  - > Set covering problem,
  - > Productivity (Assignment) Problem,
- The results were that substantial improvement was shown with CPS, as compared with traditional search strategies.

Slide 45-49:

References