Intelligent Command Control for VLSI CAD Systems

Motohide OTSUBO, Satoru FUJITA and Toru YAMANOUCHI
C&C Research Laboratories, NEC Corporation
1-1, Miyazaki 4-Chome, Miyamae-ku, Kawasaki, Kanagawa 216 Japan
{otsubo,satoru,yamano}@swl.cl.nec.co.jp

Abstract

High-performance CAD systems help designers of VLSI logic to synthesize optimal circuits. With these, users are able to repeatedly select and invoke the most appropriate-looking command (which corresponds to an algorithm). They are hard for novice users to operate, however, because efficient command-selection requires experience. In the past, novices used a "command script with heuristics" for selecting commands and automating command invocation, but the heuristics needed to be rewritten for any updated version of the CAD system being used, and users were unable to place their own deadlines on the time within which the design results had to be obtained.

To cope with these problems, we have developed an Intelligent Command Control Shell (ICCS) which performs logic synthesis tasks by automatically selecting and executing multiple sequences of commands within a pre-set time limit or until it obtains an adequate circuit.

ICCS uses easily updatable statistical data as its knowledge base. ICCS also features "time-constrained control", which takes imposed deadlines into account in its selection of commands, so as to produce the best possible circuit within a given time limit. When applied to the design of large-scale practical circuits, the use of ICCS resulted in circuits with 6% shorter delay on average (and 30% shorter delay in the best case) than those obtained with simple optimization command.

Introduction

High-performance CAD systems in VLSI logic synthesis have various circuit optimization commands corresponding to various algorithms. With these, users are able to synthesize optimal circuits by repeatedly selecting and invoking the most appropriate-looking command. They are hard for novice users to operate, however, because it takes experience to select commands efficiently. In the past, novices commonly used "command scripts with heuristics", which, while helping novices use VLSI CAD systems, had two distinct disadvantages: 1) the heuristics needed to be rewritten for any updated version of the CAD system being used, and 2) users were unable to place their own deadlines on the time within which design results had to be obtained.

To cope with these problems, we have developed an Intelligent Command Control Shell (ICCS) which performs logic synthesis tasks by automatically selecting and executing commands (Fujita, Otsubo and Watanabe 1994). In addition to its "command tree", which defines the range of available commands, ICCS uses "statistical data" representing the performance characteristics of all the circuits that can be synthesized using the tree and the properties of circuits in the process of being designed. When the ICCS is to be applied to an updated version of a CAD system, the statistical data can easily be revised from a set of typical circuits. During actual use, ICCS automatically requests property values for circuits in the process of design from the CAD system and invokes command sequence(s) in accord with its command tree. When ICCS reaches a "non-deterministic branch" of the tree, indicating the existence of several candidates, it selects and invokes the most promising one by estimating command execution results using its statistical data and the history of the property values for the circuit in the process of being designed. ICCS also features "time-constrained control", which takes imposed deadlines into account in its selection of commands, so as to produce the best possible circuit within a given time limit.

When applied to the design of large-scale practical circuits, the use of ICCS resulted in circuits with 6% shorter delay on average, and 30% shorter delay in the best case, than circuits obtained by a built-in simple optimization command, a specific type of "command script with heuristics".

ICCS uses two technologies, which are statistical expectation and real-time searching. The Magellan system (Gadient 1993) developed at CMU also uses statistical expectation to control CAD systems, though it has not yet reached the stage of being able to be

applied to practical logic synthesis tasks. Real-time searching has been the subject of extensive studies. Results include real-time searching algorithms (e.g. RTA* (Korf 1990)) and anytime algorithms for time-critical problem solving (Dean and Boddy 1988, Zilberstein and Russell 1996, Wellman, Ford and Larson 1995). ICCS is one of only a very few practical systems using real-time searching.

Description of Problem

First of all, let us consider how a VLSI logic synthesis CAD system works. It first receives input in the form of (1) the functional description of a target circuit and (2) such circuit specifications as size, delay, etc. It then synthesizes a gate level circuit out of specific parts. A semi-customized LSI with a gate-array, for example, would be designed from the basic prepared parts of both primary gates (e.g. NAND and NOR gates) and complex gates (combinations of the primary gates). Such basic parts, known collectively as a "library", differ from vendor to vendor, and most VLSI CAD systems use algorithms universal enough to handle libraries from any vendor. Some more powerful VLSI CAD systems have circuit optimization commands, each of which corresponds to a specific algorithm. In these systems, designers first input the functional description, then transform this to a number of intermediate states according to successive command invocations, and finally obtain a gate-level circuit, as shown in Figure 1. Because there are several command candidates at each successive transformation, the designers must repeatedly choose an adequate command sequence, which is assumed to be the best, by investigating the intermediate property values with built-in reporting commands and then invoke this.

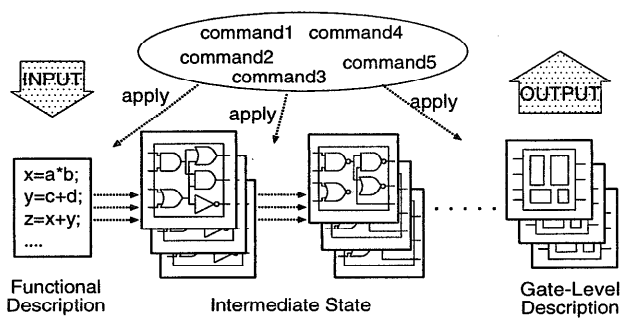

Figure 1: VLSI Design using the CAD System

Let us consider, for example, a CAD system with only two commands, flatten and compact. The flatten command transforms the intermediate states of a circuit into two-level logic form by expanding the circuit's internal nodes. The compact command extracts the identical parts from the circuit and makes these up into a new internal node. Though it takes a long time to execute the flatten command, it may produce a more optimal form for applying the compact command, because the flattened circuits are in more regular forms and they have many chances to achieve compaction. On the other hand, if the circuit is large, simply applying the compact command to the circuit may cause better compaction and takes a shorter time than the former method. Such a trade off makes it difficult for novices to operate them straight away.

In the past, VLSI CAD developers have tried to solve this problem by automating command invocation in the following ways.

- **Using command scripts:** They supplied a command script written by an expert user, or introduced a command script generator, which coordinated an effective command script according to the requirements of the circuit to be synthesized.
- **Using command scripts with heuristics:** They introduced a conditional command script which switched the remainder of its sequence into a more effective one based on the property values of circuits in the process of being designed. The conditions for the branches were produced from the expert users' heuristics.

Although these approaches were effective for automating command invocation, the first one was unable to extract maximum performance from VLSI CAD systems with many commands. This was because of "static command selection" which did not consider any properties for circuits in the process of being designed. Despite using "dynamic selection of commands", the second one had the following problems:

- **Maintenance:**
  It was expensive to modify condition parameters for every updated version of the CAD system.
- **Time limit considerations:**
  There was no way to take account of the limited execution time in selecting commands.

ICCS Mechanism

To cope with the problem of maintenance and time limit considerations, we developed the Intelligent Command Control Shell (ICCS), which dynamically selects commands in the light of expectations generated by multivariate analysis. ICCS uses two sets of given knowledge – the command tree which defines the range of available commands and the statistical data which is the accumulated empirical data on synthesis (Fujita, Otsubo and Watanabe 1994) (Fig. 2). ICCS treats the process of invoking commands towards a VLSI CAD system as a search through the command tree. We adopted a kind of real-time searching to ICCS to make full use of the limited time allowed and to obtain the circuit closest to the objective specifications.

Command Tree and Statistical Data

The command tree: is a set of command sequences described in the form of a tree. Each arc in the tree

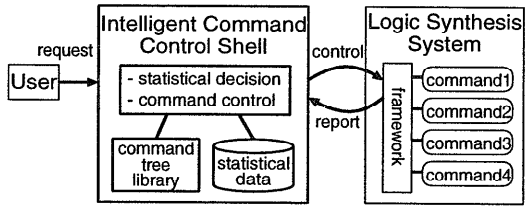

Figure 2: Basic Architecture of ICCS

is associated with a command in the LSI CAD system. There are two kinds of branch in the tree – a non-deterministic branch where ICCS can select any one of the arcs based on its own decision, and a conditional branch where ICCS selects one of the arcs according to a condition labeled on the branch. The command tree is usually formed by an expert user or the developer of the LSI CAD system, because its formation requires the enumeration of several command sequences that are effective with specific circumstances of synthesis and which are expected to lead to the best results under any circumstances.

The statistical data: is the accumulation of performance characteristics (for example, circuit size and delay) of all the circuits that can be synthesized using the tree , historical property values of circuits in the process of being designed (for example, the number of literals when describing a design object as a Boolean expression) and the execution time for each invocation. They are collected by the experimental execution of all branches in the command tree and stored on the tree where they were obtained. The statistical data should be created by a special tool with given typical circuits before using ICCS.

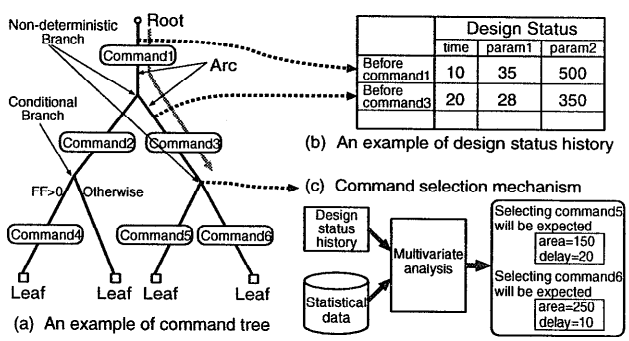

Figure 3: Example of Command Tree and the Way to use It

Figure 3(a) shows an example of a command tree. ICCS starts to search the command tree from the root towards one of the leaves. On its way towards a leaf, ICCS controls the LSI CAD system by invoking commands associated with arcs that it has passed. During the tree search, ICCS requests the property values of design objects and the execution time from the VLSI CAD system before each invocation of commands, and stores these in its design status history (Fig. 3(b)). When the execution point on the tree reaches a non-deterministic branch, ICCS predicts the performance characteristics of resultant circuits at each leaf computed by multivariate analysis using the design status history and the statistical data (Fig. 3(c)), and selects and invokes the optimal one. Because the execution point proceeds towards the leaves, the amount of information in the design status history increases monotonically. Consequently, by making predictions at leaves whenever its execution point reaches the branches, ICCS is able to maintain more possible expectations at leaves which are calculated from more information.

As mentioned above, ICCS utilizes knowledge on the sequences of commands assumed to be effective, which is part of the experts' knowledge that they can describe relatively explicitly, and knowledge on the situations where each sequence is assumed to be effective,which is determined using expectations computed from real values obtained from typical examples, i.e., results learned from typical examples. In ICCS, because the latter unverbalized knowledge is collected by a tool automatically, it is easy to update its knowledge even when the capability of the LSI CAD system has changed.

Time-constrained Control

To take account of and make better use of the limited time allowed, and to try to execute several command sequences to obtain better circuits, ICCS has a time-constrained control. The time-constrained control consists of two technologies, intelligent backtracking and real-time scheduling.

In following sections, we substitute the action of ICCS controlling the LSI CAD system as a search through the command tree. Based on this substitution, we call the leaves of the command tree "goals", the route through the command tree to obtain a circuit from the root towards a leaf "path", the circuit resulting from the execution of one path "result", and the degree of sufficiency on the resulting circuit "quality".

The Intelligent Backtracking ICCS utilizes the statistical data collected from typical examples, not from the circuit to be synthesized, so its predictions do not always come true. In addition, the goals are re-evaluated as the execution point moves, as described above, so the expected quality (or merit) of the selected path is renewed as searching the path. Therefore, to get better results, we need an enhanced search algorithm that changes the execution point dynamically according to the merits and searches several paths that are assumed to lead to the best result. Carbonell proposed the delta-min algorithm (Carbonell 1980) to

search for second-best goals, by backtracking to and restarting the search from any arc of branches assumed to be best if any time remains after reaching one of the goals. The delta-min algorithm differs from traditional backtracking algorithms, such as those used in Prolog, in that it backtracks to any of the unselected arcs passed previously, not just to the one immediately before. When traversing the tree, it records the merits on the unselected arcs when it passes a branch.

ICCS contains an improved version of the delta-min search algorithm, called intelligent backtracking. This differs from the delta-min algorithm where the backtracks may occur at any time, not only at the goal, whenever the merits of another arc are better than the current one. The outline for the searching algorithm to achieve intelligent backtracking is as follows (Fig. 4).

1. Set the root of the command tree as the current search point.
2. If the current point is at a non-deterministic branch, save the current design object on a file.
3. Store the property values obtained from the VLSI CAD system into the design status history.
4. Predict the quality of results at each goal using the design status history and the statistical data.
5. If any other arcs are expected to obtain better quality than the current one, then
   (a) save the current design object on a file and
   (b) change the current point to the best-expected arc and load the design object from the file.
6. Move the current point one step towards the goal that is expected to achieve the best quality.
7. If a result that best satisfies the requirements is obtained or the given time limit has passed, then output the best result from the ones already obtained. Otherwise go back to 2.

Even if the ICCS misses the optimal result and gets a semi-optimal one (a result with next-best quality to the optimal one), this increases the possibility of eventually obtaining the optimal result by continuing to search for several paths that it thinks might be better until the given time limit expires. In actual logic synthesis tasks, designers sometimes repeat synthesizing tasks until they obtain a circuit which fulfills requirements. Therefore, the idea of searching several path in intelligent backtracking is not very different from practical situations.

Real-time Scheduling Here, we consider maximizing the expectation of surpassing the probability to obtain better quality results than those already obtained, using intelligent backtracking.

Suppose that the expectations regarding unsearched paths are those in Table 1; the expectations of surpassing are independent of each other and a time of 10 remains. If path 4, on which the expectation of surpassing is greater than on the others, is selected, then no other paths will be selected because of the limited time; therefore, the total expectation of surpassing is 0.6. However if paths 1 to 3, all of which may be executable within the remaining time, are selected, then the total expectation of surpassing becomes $1-(1-0.2)*(1-0.3)*(1-0.5) = 1-0.28 = 0.72$, which is higher than that for path 4.

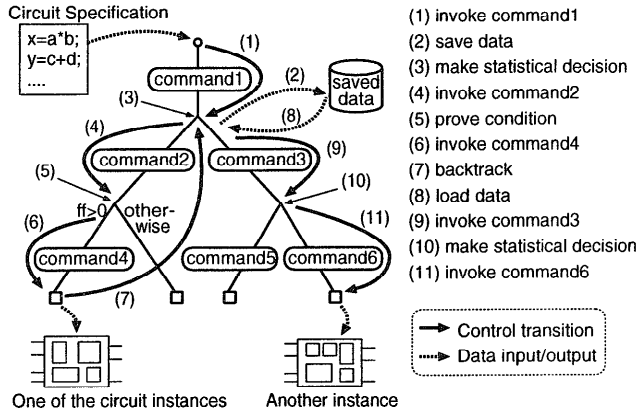

Figure 4: An Execution Example of Intelligent Backtracking

	Path 1	Path 2	Path 3	Path 4
expected time for execution	2	3	5	10
expectation of surpassing	0.2	0.3	0.5	0.6

Table 1: A Sample of Expectations

Similarly, there are some cases where selecting several paths with shorter times is better than selecting one path with the biggest expectation of surpassing.

We incorporated this idea into ICCS together with intelligent backtracking and developed real-time scheduling, which schedules the combination of paths to be executed within the remaining time to raise the probability of obtaining a better circuit. The problem of scheduling paths, which are executable within the remaining time and whose total expectation is maximal, is a kind of knapsack scheduling problem, and is known to be an NP-complete problem. For ICCS, it is not necessary to solve this problem exactly because of its frequent rescheduling and erroneous expectations. Therefore, we developed the following approximate algorithm for real-time scheduling.

1. **Preparation:** Compute the expected time for rest execution and the expectation of surpassing for each unreached goal.

2. **Get the path with the biggest expectation of surpassing:** Select one path that can be executed within the remaining time and that is directed towards the goal with the best expectation of surpassing.
3. **Get a set of paths:** Select a set of paths that can be executed within the remaining time in the following way:
   (a) Select a path with the shortest expected time.
   (b) Subtract the expected time for this path from the remaining time.
   (c) If there are no paths with shorter expected times than the remaining time, then finish selection. Otherwise, go back to (a).
4. **Select a better path:** Compare the expectation of surpassing for the path selected in step 2 and the total value for expectation of surpassing for the set of paths selected in step 3, and choose the larger one.

The expected time for execution and the expectation of surpassing computed in step 1 are updated to more probable ones as the search progresses, so the scheduling process is executed every time the expectation is updated. Using the real-time scheduling shown above leads to ICCS being able to obtain a better circuit within the limited design time.

Improvement in Searching Strategy

Although intelligent backtracking gradually corrects erroneous path selections by backtracking its search, this might produce frequent backtracking at the beginning because of poor information, and this could lengthen the time required to obtain the first circuit. To prevent this, ICCS changes its search strategy according to the number of results obtained, as described below, to ensure it obtains a result within a rather short time.

1. **Until the first result has been obtained**, it searches the tree in depth-first style, without any backtracking, towards a goal expected to give the best quality for each command selection.
2. **After it has obtained the first result**, it uses intelligent backtracking and real-time scheduling.

Evaluation of ICCS

To evaluate the practical performance of ICCS in LSI CAD, we used it to design various kinds of circuits and investigated the quality of the circuits obtained under several different time limits. All of the examinations were conducted under the following conditions;

- Using a command tree having 30-way paths.
- Using statistical data without the target circuits' data.
- Required a circuit as short delay as possible.

We measure the performance of the ICCS by the shortness of the delay in its obtained circuit. We defined the delay in the circuit as the longest time for logical changes to appear in the circuit's output pins, where all the changes in the input pins were made at time 0.

The LSI CAD system we used had a Simple Optimization Command (SOC), which was a meta-command and which acted like the command script with heuristics described "Description of Problem" section. Because SOC was created by the developers of the LSI CAD system who knew the behavior of each command, its competence in synthesizing circuits is comparable to that of expert designers. Consequently, we chose SOC as a reference for evaluating ICCS.

Evaluation Using Small-scale Practical Circuits

Using a set of 8 small-scale practical circuits, we examined ICCS under several time limits, that is, 1, 2, 5, and 10 times the normalized execution time calculated for each circuit. The normalized execution time was the mean time required to execute each of the 30-way paths in the command tree for each circuit.

The results are shown in Fig. 5, which is the mean for the 8 circuits. The horizontal axis is the time limit in units of normalized execution time and the vertical axis is the delay of the resultant circuits, normalized by that of SOC. The solid line shows the quality of circuits obtained by ICCS for each time limit, and the thick dotted lines are the shortest delay (lower line in the graph) and the longest delay (upper line in the graph) among resultant circuits when executing each of the 30-way paths, which means the best and the worst result ICCS can obtain, respectively. The thin dotted horizontal line shows the mean delay of the resultant circuits obtained from SOC and the thin dotted vertical line shows the mean time required to generate circuits using SOC.

In the case of small-scale circuits, ICCS obtained circuits having quality which was almost equal to that of SOC, within the same time limit which is the mean time SOC spent. We can see from the graph that the more time we allowed, the better circuits ICCS obtained. In this experiment, some 30-way paths had the same quality in each circuit, especially paths with differences in a few commands. Here, ICCS seemed to be unable to perform its command selection fully.

Evaluation using Large-scale Practical Circuits

We examined ICCS using 7 large-scale practical circuits, which consisted of about from 500-3000 gates. (Fig. 6 and Table 2). In Table 2, the "ratio" means the delay ratio for the resultant circuits obtained from ICCS compared to that of SOC.

In large-scale circuits, ICCS obtained a better result at a time of 1 (normalized time) than in small-scale cir-

Circuit Name	Resultant from ICCS								Resultant from SOC	
	1 unit of time		2 units of time		5 units of time		10 units of time			
	delay (ns)	ratio (%)	delay (ns)	ratio (%)	delay (ns)	ratio (%)	delay (ns)	ratio (%)	delay (ns)	exec. time (hour)
Circuit A	7.19	69.6	7.19	69.6	7.19	69.6	7.19	69.6	10.33	1.65
Circuit B	9.16	113.6	8.95	111.0	8.85	109.8	8.85	109.8	8.06	3.62
Circuit C	11.20	79.0	11.20	79.0	11.20	79.0	11.14	78.6	14.17	0.45
Circuit D	13.41	100.0	13.19	98.4	13.15	98.1	12.73	95.0	13.40	1.28
Circuit E	15.89	105.4	15.89	105.4	15.56	103.2	13.94	92.4	15.08	6.42
Circuit F	29.07	91.9	29.07	91.9	29.07	91.9	29.07	91.9	31.64	10.56
Circuit G	40.76	94.7	39.72	92.3	39.72	92.3	39.72	92.3	43.02	9.76
Average		93.5		92.5		92.0		89.9		

Table 2: Comparison of ICCS and SOC Capability

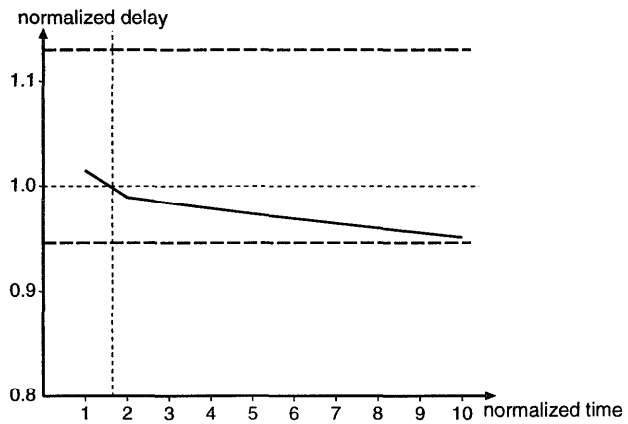

Figure 5: Evaluations of ICCS using Small-scale Practical Circuits

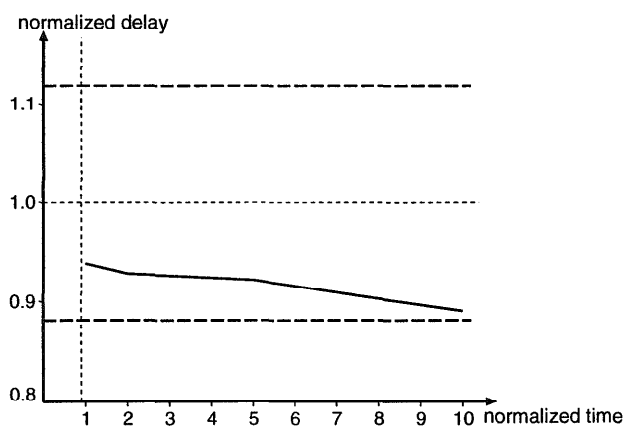

Figure 6: Evaluations of ICCS using Large-scale Practical Circuits

cuits, above. ICCS had a 6% shorter delay on average, and a 30% shorter delay in the best case (Circuit A in Table 2) than circuits obtained by SOC at the same time limit (1 unit of time). This means that ICCS achieved better control by taking into account various property values, especially in complex cases like large-scale circuits where even the difference in a few commands between paths affects the quality of synthesized circuits.

Application Status

The idea of ICCS was initially proposed in October 1992, and a beta version was developed in August 1993. An ICCS prototype was implemented mainly using LISP, and after, one was implemented using C language. ICCS currently runs on several different workstations, including SUN (Solaris and SunOS) and NEC (SVR4). After the beta version, two main changes and several minor improvements were made which resulted in the current version, which is version 2.4. It took nine person-years to develop the current ICCS.

At the beginning of development, ICCS used simple statistical expectation, which made predictions using only one property value of the design objects. This was replaced by the multivariate analysis, which is used now, in version 1.1. Also, at the beginning of development, ICCS used backtracking, which repeated the execution of a certain part of the command sequence in the tree by changing the parameters of commands. This was replaced by the current intelligent backtracking at version 2.0.

The use of ICCS is limited within our company. We are intending to apply it to practical VLSI design in the VLSI design divisions of our company, and to support them in its use. One of these divisions is eager to introduce ICCS into their process, and they reported that they got better circuits during their own experimental evaluation of ICCS. They are incorporating ICCS into their own framework for VLSI designers to use, and plan to finish doing this by the end of the first quarter in 1997.

Conclusion

We developed an Intelligent Command Control Shell (ICCS) for VLSI CAD systems which allowed us to obtain better circuit designs within a limited time.

ICCS uses statistical data as its knowledge base, which is easily updatable by collecting the property values of design objects and the quality of synthesized circuits using a set of typical circuits. ICCS also features time-constrained control, which takes imposed deadlines into account in its selection of commands, so as to produce the best possible circuit within a given time limit.

When applied to the design of large-scale practical circuits, the use of ICCS resulted in circuits with 6% shorter delay on average (and 30% shorter delay in the best case) than those obtained with simple optimization command.

To eliminate the updating task for statistical data and to enable ICCS to be distributed more quickly, we intend to develop a learning technique for statistical data, by which ICCS gradually adapts to the new LSI CAD system by collecting data during execution. We also intend to develop a mechanism for the parallel control of CAD systems so that several command sequences using several processors (or workstations) can be searched simultaneously (Fujita and Lesser 1996).

References

Carbonell, J. G. 1980. DELTA-MIN: A Search-Control Method for Information-Gathering Problems. In Proceedings of the First National Conference on Artificial Intelligence, 124–127. AAAI/MIT Press.

Korf, R. E. 1990. Real-Time Heuristic Search. *Artificial Intelligence* Vol. 42:189–211. Elsevier Science Publishers.

Gadient, A. J. 1993. A Dynamic Approach to CAD Tool Control, Research Report No. CMUCAD-93-13, Dept. of Electrical and Computer Engineering, Carnegie Mellon Univ.

Fujita, S., Otsubo, M. and Watanabe, M. 1994. An Intelligent Control Shell for CAD Tools. In Proceedings of the Tenth Conference on Artificial Intelligence for Applications, 16–22. IEEE Computer Society Press.

Fujita, S. and Lesser, V. R. 1996. Centralized task distribution in the presence of uncertainty and time deadlines. In Proceedings of the Second International Conference on Multiagent Systems, 87–94. AAAI Press.

Dean, T. and Boddy, M. 1988. An Analysis of Time-Dependent Planning. In Proceedings of the Seventh National Conference on Artificial Intelligence, 49–54. AAAI/MIT Press.

Wellman, P. R., Ford, M., and Larson, K. 1995. Path planning under time-dependent uncertainty. In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, 532–539.

Zilberstein, S. and Russell, S. 1996. Optimal composition of real-time systems. *Artificial Intelligence* vol. 82:181–213. Elsevier Science Publishers.