Proceedings of the International Conference on Intelligent Systems: A Semiotic Perspective, Gaithersburg, MD, October 20-23, 1996.

Planning in the Hierarchy of NIST-RCS for Manufacturing

J. Albus, A. Lacaze, and A. Meystel
Intelligent Systems Division,
National Institute of Standards and Technology
United States Department of Commerce
NIST

ABSTRACT

Planning in the manufacturing environment can be approached consistently as a part of the general Behavior Generation process. In this paper, different planning strategies are compared, and a planner for the hierarchical reference architecture NIST-RCS (National Institute of Standards and Technology Real time Control System) is described. The suggested planner is simulated and the results are compared with other algorithms.

I. NIST-RCS Reference Architecture

NIST-RCS is a multiresolutional control system that has been previously presented in numerous papers [1, 2, 3, 4, 5, 6]. It divides the control problem (manufacturing) into a set of levels which has different range and resolution in time and space. This permits the definition of functional entities called control nodes at each level in such a way that they have a narrow focus of priorities and responsibilities to process. At all levels within the control hierarchy, entities receive goals and priorities from above, decompose them into subtasks, and distribute these subtasks to the lower level entities so that a cost function is minimized [5]. In NIST-RCS, each of these entities (human or machine) subscribes to a set of protocols that organize the flow of control commands.

Each control node of the NIST-RCS hierarchy has the following components:

Sensory Processing (SP) is where sensors at the highest level of resolutions, or virtual sensors at the other levels are observed. SP filters, detects and recognizes patterns with the help of the World Model to determine the state of the system.

World Model (WM) stores knowledge that comes from SP module. It also retrieves knowledge to help the Behavior Generation plan its action through simulation.

Behavior Generation (BG) uses the knowledge stored in WM module and the cost functions Value Judgment to find and execute plans and control actions.

Value Judgment (VJ) contains a set of cost functions and helps with the prediction of benefits and uncertainties of the generated plan.

Specifically, in this paper we are interested in the BG module because it creates and executes the plans. As Figure 1 shows, the BG module has the following components:

Job Assigner (JA) places possible distribution of tasks to the subordinators.

Scheduler (SC) assigns time tags to the distributions given by the JA.

Plan Selector (PS) chooses among the best possible plans jointly investigated by JA and SC.

Executor (EX) provides feedback compensation using (SP and WM) during the execution of the tasks selected by PS.

In turn, each task in a plan executed by EX is distributed to the lower level (higher level of resolution). Then, a set of similar BG modules decomposes the tasks, plans, and executes them. These lower level BG modules are similar. They have the same components as the higher level BG, with some differences:

1. The language used is of a higher resolution. It carries more details.
2. The time frame allocated for planning and executing is smaller.
3. The speed at which the WM is updated is faster.
4. The scope of interest (space of search) is smaller.

II. Planner within BG-module

The functions of a planner is to find "the best" possible plan:

- at the assigned level of resolution
- with respect to an assigned cost function

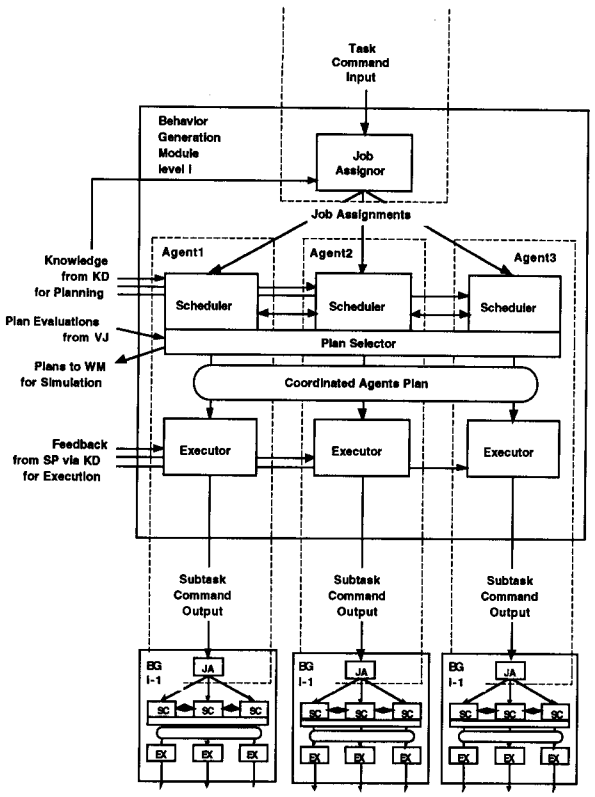

Figure 1: Behavior Generation

- following all the assigned constraints, and
- in the allocated computational time.

In general, planning consists of the following steps:

- Create (using combinatorics or other methods) sets of future decisions of space (resource-task) assignments
- Calculate the next state after each decision was made.
- Assign time tags to the decisions.
- Predict the cost of the strings of future decisions
- Simulate the strings to check for possible inconsistencies (or conflicts)
- Select the time tagged string of decisions (plan) that follows the constraints and better fits the assigned cost function.

To perform the assigned tasks, the NIST-RCS planner (PL) is composed of three modules: JA, SC, and PS. It receives a task from the upper level (lower resolution) and transforms this task into time tagged set of subtasks (plan) that is passed to EX for execution. These three modules were described in the previous section and shown in Figure 1.

The same modules (JA, SC, PS) can be identified in all planning algorithms. By modules we are implying that there are conceptually different functions that the planner must perform. This does not mean that they must be separated pieces of software, and we are not implying any interchangeability properties. Although there is a wide variety of planning algorithms, most planners mainly differ on the interaction between the JA and the SC.

1. *Tightly coupled JA-SC.* In these cases, the spatial and temporal search is performed jointly by the JA-SC couple. JA gives one possible assignment (one decision) at a time to the scheduler. Based on the scheduler's response, JA makes a new possible assignment. Dynamic programming fit this category.
2. *Loosely coupled JA-SC.* There are some cases where JA and SC can be loosely coupled. Most of them are just special cases of the tightly coupled case:
   - There are no "possible" alternative job assignments. The JA makes an actual instead of possible assignment, and gives it to SC. Because no alternatives exist, the complete assignment can be given to SC. There is no need for a JA-SC loop. All dispatching rules fit under this category.
   - The cost cannot be evaluated on a decision-by-decision basis. Complete strings of job assignments are created by JA and then given to SC. In this case, there is a JA-SC loop. It is done in plan-by-plan steps. These cases are rare and make the problem of finding "the best" plan much more cumbersome, because it is necessary to calculate the cost of the complete string. In most cases, it is possible to find decision by decision cost approximations that allow us to use more efficient tightly coupled JA-SC algorithms.

In some cases, task decomposition precedes SC. But, in most cases, only after scheduling is completed, the task can be considered decomposed.

III. Planner in the NIST-RCS Hierarchy

There is a common misconception that says that the structure of planners at each manufacturing level has to be radically different. We have found that at all the levels where we have a fairly well-defined WM representation, the planning problems can be transformed into multi-dimensional state space search that can be conceptually solved with the same algorithms. The difference between one level and the next, is that the type of information represented changes, but not the representational structure. So, the planner can search through a decision graph in a similar way although the information is conceptually different. We believe that the same planning algorithm can be applied to:

- Resource planning level
- Shop scheduling level
- Work cell planning level
- Machine control level
- Actuator control level

It can also be applied to the product and production planning level in cases where the World Model is thoroughly defined. This is rarely the case in production scheduling.

The planning problems are typically represented as follows:

- There are t tasks to be performed
- There are r resources that can perform those tasks
- There is a cost function to be minimized
- There is a set of ordering constraints that must be followed

Find the distribution of tasks among the resources (JA) and times of performing these tasks (SC) so that the the cost function is minimized and the constraints followed.

Assume that we have two resources (r_1 and r_2), and three tasks (t_1, t_2 and t_3) that must be processed. Let us say that t_2 and t_3 can only be performed if t_1 is finished. For simplicity say that I am trying to minimize time. PL needs to plan the distribution of jobs and calculate the time that should be processed so that the cost function is minimized. At $t = 0$ there are only two possible (fully allocated) assignments: t_1 assigned to r_1 (alternative 1) or t_1 assigned to r_2 (alternative 2). If PL explores the alternative 1, there are two different decisions that can be made after the m_1 is done: t_2 to r_1 and t_3 to r_2 (alternative 11) or t_3 to r_1 and t_2 to r_2 (alternative 12). Two other alternatives are possible if we explore alternative 2. All these alternatives can be stored in a tree form, the decision tree. The string of decisions between the root of the decision tree and a leaf is a plan. The decision tree is useful in comparing planning algorithms to see what decisions are explored. This model may not be correct for production scheduling where there are penalties for not processing tasks, but the "optimal" plan may not process all of them. We are assuming that all tasks must be processed.

IV. The algorithm of Planning

Our strategy has the following characteristics:

1. We assume that a decision-by-decision cost function is available. The cost of the complete course of action will be the addition of all the individual costs of all the decisions taken along that decision path. Decision-by-decision cost functions (incremental cost functions) are not always available, for example, the cost of not delivering a work within a given window for production planning.
2. The JA-SC couple weaves the decision tree as it generates and explores the alternatives instead of first creating the complete decision tree and then searching through it.
3. Only in worst computational cases the complete decision tree will be evaluated.
4. No decision path is studied more than once. It keeps track of previously explored paths, unlike most genetic planners.
5. The premises of the theoretic fundamentals have been thoroughly tested in different cases of path planning. Although path planning is not viewed as a classical scheduling problem, the algorithm that we present can be used for both cases.
6. Even if the algorithm is not finished, the first decisions (the ones that need to be executed the sooner) of the best incomplete plan have lower probability of being changed.

The algorithm that we propose for NIST-RCS is a mathematical programming technique rooted in Dijkstra or A*. It has the following steps:

1. If the decision tree is empty then $pl_{sel} = \emptyset$ go to Step 3
2. Find the cheapest leaf (le_{sel}) in the decision tree. The cost of the leaf is the cost of the decision path until the leaf (Dijkstra). Or the cost is the summation between the cost of the path until the leaf and the estimated cost to the end of the plan (A*). The second choice requires an estimate to the end of the plan that needs to be always smaller or equal to the actual path to warranty optimality. Label the plan created by the decisions made from the root (or roots) of the decision tree to the selected leaf as pl_{sel}
3. Identify the number of tasks for each task type $(nt_1, nt_2, \dots)$ that have all their constraint satisfied at t_{sel} given that all the decisions in pl_{sel} were taken. Where t_{sel} is the next time at which a resource becomes free or a task is finished (or both) along pl_{sel}
4. Identify the number of resources available for each task type $(nr_1, nr_2, \dots)$ at t_{sel} given that all the decisions in pl_{sel} were taken.
5. For each task type (i) :
   (a) Create all the combinations of the nr_i resources and the nt_i tasks. Or combinations of the nt_i tasks and the nr_i resources if $nr_i > nt_i$
   (b) Create all the permutations of each combination and store them in a list (L_i).
6. Create a new leaf in the decision tree for each possible assignment $l_{n,1} l_{m,2} l_{o,3} \dots l_{z,i}$ where $l_{n,1}$ is the an element of L_1, $l_{m,2}$ is an element from

L_2, etc. These leaves hang from le_{sel}

7. Find which is the cost before the next event in each of these new leaves of pl_{sel}. Where an event is a resource becoming free or a task being finished (or both). Store this cost and the time of the event in the leaf (to be used by Step 2)
8. [optional] prune the new leaves if they achieve a state that was achieved by another path in a cheaper manner. This step is optional because finding these same states may be more expensive than not pruning.
9. If all assigned tasks are finished, select the completed plan with the best cost backtracking through the decision tree; exit; else go to Step 2

To relate the previous algorithm to the material presented in Section II, steps 2 through 7 show a tightly coupled JA-SC algorithms, and step 9 can be considered a sketch of the the PS module. Step 7 implies that functions from VJ are invoked. This algorithm will find the optimal cost because all the paths that are shorter than the optimal cost will be explored by Step 3 both in the A* version and in the Dijkstra version. No path longer than the optimal one will be explored to the end.

Simon states in [7]

> ... while OR [operations research] has been able to provide a theoretical analysis of scheduling problems for very simple situations — a shop with a very small number of machines and processes — automatic scheduling of large shops with many machines and processes has been beyond the scope of exact optimization methods. [p 13]

This is true if we attack the problem of planning as a one level problem. But, NIST-RCS provides an architecture where each level has a reduced focus of attention (space of search) thus allowing the use of optimal algorithms like the one presented. Because PL creates plans only for one level, it should always be a limited amount of combinations to study.

It is not always clear that the planning problems can be decomposed into subproblems with constrained (preferably nil) interactions. An assumed decomposition does not necessarily warranty the best solution for that decomposition. If the decomposition is inappropriate, then the solution found will be a bad one. On the other hand, we know that complex planning problems are always subdivided into different levels. We have seen this divisions into control layers in cases spanning from the Roman empire army to current managerial structures. In the case of manufacturing, we can go from the Actuator control level where the tasks could be the application of a voltage to a motor, to a Machine control level where the tasks can be the milling of a part, and the resources are the different machines that can do that milling. In the case of the Work cell planning level, the tasks could be small batches of parts, and the resources, are the different work cells within the shop floor. In most cases, it is possible to identify the tasks and resources at each level of the manufacturing control hierarchy.

If the algorithm takes more time than the allocated to find the optimal plan, the following alternatives are always available:

1. A new intermediate NIST-RCS level to the control hierarchy, which makes the search space smaller and the search time reduced.
2. Allow the presented algorithm to search as far into the future as the time allocated, and if it is not completed when the result is needed, extend the best path found at that moment with a dispatching rule. Of course, we will not be able to warranty optimality. The addition of the dispatching rule will create a "just-in-time" planner. Still, the algorithm warranties to find a better plan (or equal in the worst case) than the dispatching rule by itself.

V. Present Views on Planning

In this section we will compare some planning and scheduling techniques that are currently in use.

V.A. Production Scheduling

Production scheduling is concerned with the effective allocation of the manufacturing resources over time for the manufacture of goods. These resources include the agents, the materials, and time. The agents are a combination of machines and manpower. Most organizations, both large and small, must solve scheduling problems on a recurrent basis. Creates considerable demand for good scheduling techniques. Starting in the 1950's, operations researchers began advocating formal optimization approaches to scheduling [8].

V.A.1. Production Scheduling from the AI Point of View

Knowledge Based Systems (KBS) provide the major AI approach to scheduling in general and production scheduling in particular. These employ domain specific information to derive schedules and in the process do not aspire nor guarantee optimality [8].

In turn, KBS can be divided between those that use dispatching rules (most) and those that don't. Among those that use dispatching rules we can find [9, 10, 11]. GENREG [12] can modify a set of dispatching rules to better suit the problem.

A different approach to scheduling within the AI paradigm is constraint based reasoning. Within this

category we can find ISIS [13]. ISIS creates a "beam" search where a few alternatives are created based upon some arbitrarily selected operators. ISIS 3 is "hierarchical" with the following levels predefined: order selection, resource selection, reservation selection. Please note that what is meant by hierarchy is very different from the NIST-RCS hierarchy. If none of the best-first itineraries satisfies all the constraints, then a "repair" algorithm is called to try to reschedule the "level" with a more relaxed set of constraints. ISIS evolved into OPIS [14]. Its methodology is almost identical to ISIS ("beam"). It is a reactive scheduling algorithm in the sense that it concentrates upon how to modify the schedules already found to satisfy new constraints and new rules. OPIS is composed of the following parts: order scheduler, resource scheduler, right shifter, left shifter, demand swapper. The current versions of OPIS are not suited for real time operations because of the time that it takes to schedule [8].

[15] presents a hybrid system KBSS that combines KBS (reactive) systems with search. It is claimed that KBSS finds schedules of good quality, usually within a couple of percentage points from the optimum.

V.A.2. Computational Intelligence (CI) Approach to Production Scheduling

CI methods for production scheduling are mainly based upon: genetic algorithm (GA) [16] , simulated annealing, and neural networks.

The genetic algorithm approach is basically a search technique that mimics the general idea of genetic evolution. GAs quickly identify possible solution areas, but may not converge to the optimal solution rapidly [17]. GAs require different operators to create the new "generation" of solutions. These operators are a set of heuristics that cross the current generation of solutions or schedules. [18, 19] show two different kinds of heuristical crossover operators.

Musser in [20] uses a simulated annealing technique that is currently used at the Texas Instruments PC-manufacturing facility. produces a weekly schedule for a process containing 50 operations. The average time to generate the schedule is 15 minutes which precludes it from operating real time [8]. Simulated annealing is another search technique that has its roots in the formation of crystals and is related to mechanical engineering. Laarhoven in [21] presents an argument where he concludes that simulated annealing is not as efficient as some other tailored heuristics, and thus showing large running times.

Vaithyanthan and Satake in [22, 23] present two Hopfield network schedulers. They have different methods to deal with the problem of early conversion of their networks. But both claim they have no way to ensure that the final results are optimal or reasonably close to the optimum.

In [24, 25] Willems and Aoki use neural networks in conjunction with integer programming to schedule.

V.B. Project Scheduling

Project scheduling allocates resources at specific times to tasks which together complete a project or accomplish a mission. Although production scheduling involves recurrent operational decisions, project scheduling normally involves a unique project. The project may be the construction of a house or a deep space expedition by an unmanned probe. The principles involved in project scheduling carry over from one project to another. But, we never schedule exactly the same objects for the same reasons as in production scheduling [8].

Two earlier methods of project scheduling became universally accepted: the critical path method (CPM) and the project evaluation and review technique (PERT). CPM considers activity times as a function of cost and seeks to trade-off cost with completion time. On the other hand, PERT models activity times as random variables and is used to analyze project completion time [26, 27, 28] explain extensibly both models.

V.B.1. Project Scheduling from the AI Point of View

In [29] project scheduling is done by way of an expert system where previous solutions to the problem are stored and the schedule is adapted by a set of heuristics that rearrange a set of operations that apply to the schedule.

Barber in [30] describes a prototype for an intelligent tool for project control and scheduling known as XPERT. It considers the problem of expediting selected activities if a project is to break some timing constraint. Again it is a set of heuristics used for "fixing" a previous schedule.

V.B.2. Computational Intelligence (CI) Approach to Project Scheduling

A number of classical algorithms (Simplex, network flow, Dijkstra) are used under the CPM approach in [27]. McKim in [31] presents three case studies in Neural networks and compares them to classical methods.

VI. Comparing DP and dispatching Rules

The following question should be addressed while creating an optimal planning algorithm: is the cost saved by the optimal planning algorithm significant enough to make the effort to look for more alternatives than a simple dispatching rule?

The following experiment was conducted to answer this question:

- A number of planning problems were created

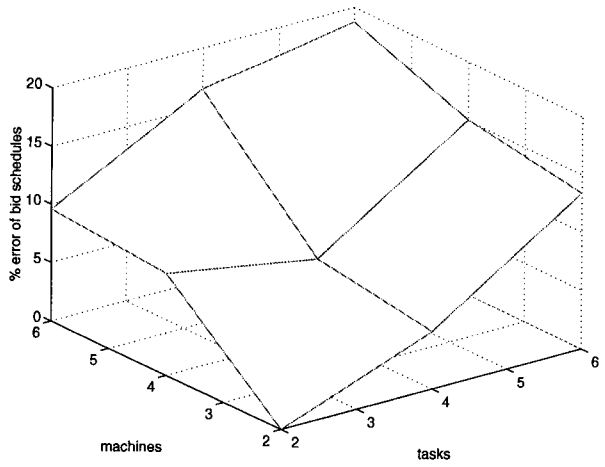

Figure 2: Error produced by the dispatching rule

with t tasks and r resources.

- the length of the tasks and the efficiency (speed) of the resources was assigned randomly.
- no precedence constraints were imposed
- Bidding (best first) was used to plan all the created planning problems. Approximately 20 planning problems were created for each combination of tasks and resources, $t = 2, 4, 6$ and $r = 2, 4, 6$.
- Dijkstra was used to plan the same problems.
- Results where compared

Figure 2 shows the average extra time (cost in this case) caused by the dispatching rule not picking the correct alternative. The extra cost becomes significant even with a small number of tasks and resources as shown in the example. The example was conducted only at one level of resolution. But we can expect that similar extra cost will be incurred by each level of the NIST-RCS hierarchy.

VII. Conclusions

1. Planning is realizable on line as a joint search procedure of JA, SC, and PS for a broad variety of numerical examples.
2. NIST-RCS allows us to use optimal algorithms by reducing the space of search.
3. Dynamic Programming (Dijkstra) is a good choice among the optimal algorithms
4. Optimal Algorithms give significant cost advantages over dispatching rules.
5. The proposed strategy and algorithm of planning allows for using the unfinished results ("just-in-time planning").

References

[1] J. Albus. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21:473–509, 1991.

[2] J. Albus. RCS: A reference model architecture for intellignet control. *COMPUTER, Special Issue: Computer Architecture for Intelligent Machines*, pages 56–59, May 1992.

[3] J. Albus. RCS: A reference model architecture for intellignet machine systems. In *Proceedings of the Workshop on Intelligent Autonomous Control Systems*, Haifa, Israel, 1992.

[4] J. Albus and A. Meystel. A reference model architecture for design and implementation of semiotic control in large and complex systems. In *Semiotic Workshop of the 10th IEEE Intenational Symposium on Intelligent Control*, pages 33–45, Monterrey, CA, 1995.

[5] J. Albus. An intelligent system architecture for manufacturing (ISAM). Technical report, NIST, 1996.

[6] J. Albus and A. Meystel. Behavior generation. Technical report, NIST, 1995.

[7] H. Simon. Two heads are better than one: the collaboration between AI and OR. *Interfaces*, 17(4):8–15, 1987.

[8] D. Brown, J. Marin, and Scherer W. A survey of intelligent scheduling systems. *Intelligent Scheduling Systems*, 1995.

[9] R. Conway. Priority dispatching and job lateness in a job shop. *Journal of Industrial Engineering*, 16:123, 1965.

[10] D. Elvers. Job shop dispatching rules using various delivery dates. *Prod. Inventory Management*, 14:61, 1973.

[11] J. Blackstone, D. Phillips, and G. Hogg. A state of the art survey of dispatching rules for manufacturing job shop operations. *International Journal of Prod. Res.*, 20(1):27–45, 1982.

[12] H. Pierreval and H. Ralambondrainy. A simulation and learning technique for generating knowledge about manufacturing systems behavior. *Journal of Operation Research*, 41(6):461–473, 1990.

[13] M. Fox. ISIS a retrospective. *Intelligent Scheduling*, 1994.

[14] Stephen Smith. OPIS: a methodology and architecture for reactive scheduling. *Intelligent Scheduling*, 1994.

[15] A. Kusiak. A knowledge optimization based apporach to scheduling in automated manufacturing systems. In D. Brown and C. White, editors, *Operations Research and Artificial Intelligence*, pages 453–479. Kluger Academic, 1990.

[16] J. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[17] S. Uckun, S. Bagchi, K. Kawamura, and Y Miyabe. Managing genetic search in job shop scheduling. *IEEE Expert*, pages 15–24, October

1993.

[18] D. Whitley, T. Starkweather, and D. Shaner. The travelling salesman and sequence scheduling. In L. Davis, editor, *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[19] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[20] K. Musser, J. Dhingra, and G. Blakenship. Optimization based job scheduling. *IEEE Transactions in Automatic Control*, 38(5), May 1993.

[21] V. Laarhoven. Job shop scheduling by simulated annealing. *Operation Research*, 40(1):243, 1990.

[22] S. Vaithyanthan and P Ignizio. A stochastic network for resource constrained scheduling. *Computers Ops. REs.*, 19(3/4):241–254, 1992.

[23] Y. Satake, K. Morikawa, and N. Nakamura. Neural-network approach for minimizing the makespan of the general job shop. *International Journal Prod. Econ.*, 33:67–74, 1994.

[24] T. Willems and J. Rooda. Neural networks for job-shop scheduling. *Control Engineering Practice*, 2(1):31–39, 1994.

[25] K. Aoki, M. Kanesashi, M. Itoh, and H. Matsuura. Power generation scheduling by neural networks. *Int. J. Systems Sci.*, 23(11):1977–1989, 1992.

[26] S Elmaghraby. *Activity Networks: Project Planning and Control by Network Models.* Wiley, 1977.

[27] J. Moder, C. Phillips, and E. Davis. *Project Management with CPM, PERT and Precedence Diagramming.* Van Nostrand Reinhold, 3rd edition, 1983.

[28] S. Gupta and L. Taube. A state of the art survey of research on project management. In *Project Management: method and studies.* Elsevier Science Publishers, 1985.

[29] Y. Zong, T. Yang, and J. Ignizio. An expert system using an exchange heuristic for the resource constrained scheduling problem. *Expert Systems with Applications*, 8(3):327–340, 1993.

[30] T. Barber and J. Boardman. Knowledge based project control employing heuristic optimization. *IEE Proc*, 135(8):529–538, 1988.

[31] R. McKim. Neural network applications for project scheduling: Three case studies. *Proj. Mgmt. J.*, 24(4), 1993.