ELSEVIER

Computers and Chemical Engineering 26 (2002) 239–267

Computers & Chemical Engineering

www.elsevier.com/locate/compchemeng

Algorithm architectures to support large-scale process systems engineering applications involving combinatorics, uncertainty, and risk management

Joseph F. Pekny *

School of Chemical Engineering, Purdue University, West Lafayette, IN 47907-1283, USA

Received 18 April 2001; received in revised form 17 September 2001; accepted 17 September 2001

Abstract

Increased information intensity is greatly expanding the importance of model-based decision-making in a variety of areas. This paper reviews applications that involve large-scale combinatorics, data uncertainty, and game theoretic considerations and describes three related algorithm architectures that address these features. In particular, highly customized mathematical programming architectures are discussed for time-based problems involving significant combinatorial character. These architectures are then embedded into a simulation-based optimization (SIMOPT) architecture to address both combinatorial character and significant data uncertainty. Multiple SIMOPT objects are combined using a coordinating architecture to address game theoretic issues. A key focus of the paper is a discussion of the algorithm engineering principles necessary to mitigate the NP-complete nature of practical problems. The life cycle issues of algorithm delivery, control, support, and extensibility are important to sustained use of advanced decision-making technology. An inductive development methodology provides a means for developing sophisticated algorithms that become increasingly powerful as they are subjected to new constraint combinations. Implicit generation of formulations is crucial to routine large-scale use of mathematical programming based architectures.

Keywords: Risk management; Algorithm architectures; Combinatorics

1. Introduction

The global economy continues to undergo massive changes. In contrast to the industrial revolution of the early 20th century, which primarily effected society's ability to manipulate the physical world, these changes are being induced primarily by the explosive evolution of information generation, management, and dissemination. The World Wide Web and the Internet revolution are a popular and well-documented example of the self-catalytic nature and speed of these changes. As the Web and Internet technologies permeate society, they are changing how people collect and use information. Indeed, businesses are beginning to learn to use the Web and Internet in concert with other information technology (e.g. personal computers, databases, etc.) for rapid response to customer needs and opportunities. As such businesses are becoming acutely aware that they exist in a highly dynamic environment where speed and effectiveness of response is a matter of profitability and survival. This trend towards the speed at which business is conducted and important decisions have to be made has significant implications for the need for Process Systems Engineering (PSE). For purposes of this paper PSE is defined to be the coupling of engineering, physics, software, mathematics, and computer science principles to address business and industrial applications (see Grossmann and Westerberg (2000) for a more detailed discussion).

The Internet revolution to date has involved significant first order use of information. That is, this initial phase of the revolution made the same large body of information available to anyone who could access the Internet. Of course the existence of and access to this large body of information generates value for society

* Tel.: +1-317-494-7901; fax: +1-317-494-0805.
E-mail address: pekny@ecn.purdue.edu (J.F. Pekny).

0098-1354/02/$ - see front matter
PII: S0098-1354(01)00744-X

and prepares a foundation for next generation advances (Magnusson, 1997; Phelps, 2002). A critical next generation advance, and one that the PSE community is poised to contribute to, is the systematic processing of information into the knowledge of which actions to undertake to achieve specific goals. In particular, the PSE discipline has developed formalism for systematically translating information into decisions in a goal-oriented fashion. In fact the speed of the economy in general and the speed at which profitability of a market attracts competition makes the adoption of systematic means of transforming information to knowledge essential. This follows because the speed of the economy lessens the value of 'steady-state' experience since companies now rarely operate in a static environment and companies need other means for generating a competitive advantage since mere possession of information is a less significant advantage in an information rich environment.

In general terms, the PSE formalism for transforming information into knowledge involves (i) the analysis of a problem to understand its essential features, (ii) the construction of a model which captures these essential features and provides a concise statement of the goal and constraints to achieving the goal, (iii) a means of obtaining answers from the model, and (iv) interpreting the answers to understand the range over which they are valid and how to implement them in practice. A model is an expression of how various pieces of information relate to one another and can take on many different forms, for example a spreadsheet, a neural network, an expert system, a mathematical program, a regression equation, an x–y plot, etc. In virtually every successful application of PSE technology models undergo iterative refinement that involves adapting the model to reflect continually improving understanding and an evolution of needs. Obtaining a solution to a model requires an algorithm for which the input is the specific data driving the model and whose output is a solution to the problem implied by the data. Insight is often generated through repeated use of a model to understand how output changes with input and this feedback loop makes speed of solution an important consideration. Because of the theoretical difficulty in obtaining answers from models to most PSE problems (see Pekny & Reklaitis, 1998), the discipline of algorithm engineering is critical to meeting the problem solving challenges.

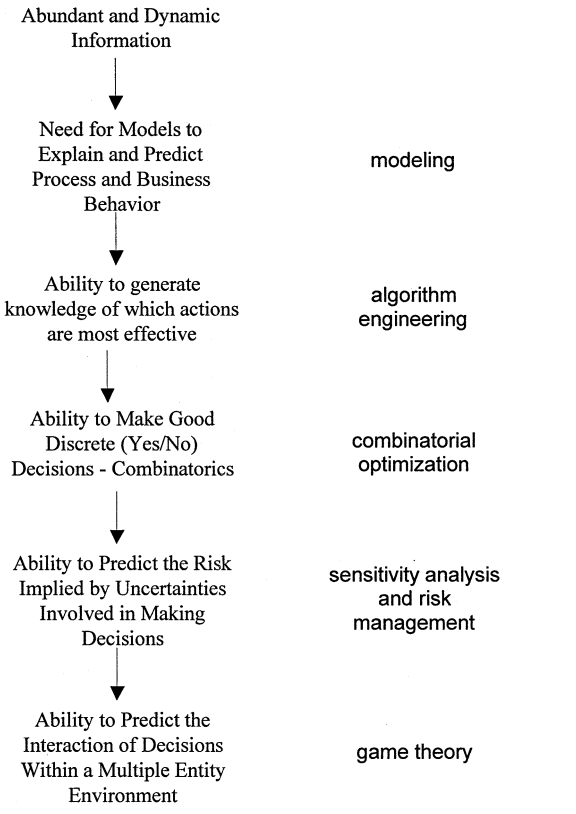

Fig. 1. Hierarchy of modeling issues motivated by abundant and dynamic information.

Fig. 1 summarizes the motivation for this paper. As the progression of boxes shows, rapidly changing and abundant information induces a need to use more sophisticated models to explain behavior. In process and business applications these models often require making a number of discrete decisions. Given real world uncertainties there is a natural need to understand how decisions and risk depend on key information. Many environments in which information is used involve multiple entities so that game theoretic considerations are important to key decisions. The remainder of this paper discusses three related algorithm architectures for addressing the bottom three boxes of Fig. 1. In particular, we review the use of highly customized mathematical programming technology for problems involving large-scale combinatorial optimization. This technology is in its infancy, but promises to systematically improve decision-making in strategic, tactical, and real-time environments. The next section reviews examples where large-scale combinatorial optimization is important. The following section describes a Mathematical Programming Based Architecture (MPA) for solving models with a large combinatorial component. Subsequent sections describe the Simulation-Based Optimization (SIMOPT) architecture for addressing risk management and data uncertainty and an electronic Process Investigation and Management Analysis (*e*PIMA) architecture for addressing game theoretic applications. The role of algorithm engineering is also discussed, especially as it applies to making the PSE modeling formalism practical.

Table 1
Feature based scheduling problem classification

Description	Industry sector
Multiple small molecule products, similar recipe structure, campaigning/cleanout/setup, common production equipment	Pharmaceutical and specialty chemicals
Long chain process, fixed batch sizes, process vessel storage, dedicated equipment	Protein manufacture
Identical recipe structure, large no. of SKUs, single equipment processing followed by packaging, labor constraints	Specialty blending
Limited shared storage, convergent process structure, campaigning/cleanout/setup, parallel equipment	Poly-olefins manufacture
Convergent recipe structure, shared storage, renewable resource constraints, batch size limitations	Food and personal care
Asymmetric parallel equipment, implicit task generation, unlimited storage, multiple steps	Large-scale printing and publishing

2. Combinatorial nature of many process systems applications

Many PSE applications have a strong combinatorial character. In principle, this combinatorial character can be thought of as a series of yes/no questions that when answered in conjunction with specifying some related continuous variables values defines a solution to a problem. This paper focuses on applications in the process management domain, including:

- process scheduling and planning;
- process design and retrofit;
- model predictive decision-making;
- warehouse management;
- supply chain design and operation; and
- product and research pipeline management.

The problems defined by these application domains are difficult in the theoretical sense in that they are virtually all NP-complete (see Garey & Johnson, 1979; Pekny & Reklaitis, 1998). In the intuitive sense they are intractable to solve using explicit enumeration algorithms for answering the yes/no questions since there are typically hundreds to millions of yes/no decisions in problems arising in practical applications. From the standpoint of this paper, these application domains are related in the sense that they can utilize the same underlying technology for transforming information to knowledge. In particular the discussion of MPA, SIMOPT, and *e*PIMA is best motivated by understanding the key features of these related application areas. One key feature that unifies all the problems in these application domains is the need to manage time. Much of the combinatorial complexity of these applications is attributable to the management of time since many types of yes/no decisions can be thought of as recurring periodically throughout the horizon of interest. This recurrence of yes/no decision type can increase problem size by one or more orders of magnitude depending on how finely grained time must be managed. From a practical point of view the applications differ considerably in terms of data quality, organizational interest, end user skills, tool implementation issues, etc. However, the purpose of this paper is to discuss the common features of problems involving large-scale combinatorics and algorithm architectures that may be engineered for their solution. A summary of each application domain and their relationship to each other is as follows.

2.1. Process scheduling and planning

The problems in this domain are conveniently described using the resource-task-equipment framework (Pantelides, 1993). Resources can be considered renewable or consumable according to whether they are reusable or are consumed upon use and must be replenished. Tasks can be thought of as actions that are applied to resources. For example a reaction task can input raw material resources at the beginning and then emit product after a certain amount of time. Together a set of tasks defined to utilize a set of resources can define an entire production network. Equipment is a special kind of renewable resource that is required in order for a task to execute. In principle the resource-task description of a process is sufficient, but the redundancy of explicitly defining equipment provides a great deal more information to algorithms for solving process scheduling problems. In general the scheduling problem is to determine a time-based assignment of tasks to equipment so that no process constraints are violated. Typical constraints that must be satisfied are resource balances, inventory limitations (minimum and maximums), unit allocation constraints, renewable resource limitations, and restrictions on how equipment may be used in time. The need to manage time greatly complicates the solution of process scheduling problems because the interaction and tightness of these various types of constraints can vary greatly, even in a single problem instance. For example the bottleneck piece of equipment may change several times over the course of the problem horizon or the shortage of labor may constrain the schedule at particular times even though labor may be abundant at other times. In fact, the various ways in which process scheduling constraints may interact over time gives rise to well-defined classes of problems. Table 1 summarizes the physical characteristics of a few of these classes of process scheduling problems that are encountered in practice. The widely

different nature of these problem classes has critical implication on how to approach their solution as will be discussed below. Goals in solving process scheduling problems include minimizing cost, minimizing the number of late orders, or maximizing process throughput (Applequist, Samikoglu, Pekny & Reklaitis, 1997; Reklaitis, Pekny & Joglekar, 1997).

2.2. Process design and retrofit

The process design and retrofit problem is a generalization of the process scheduling problem in that all the same decisions are required along with decisions as to when and how much equipment to add to a process. Process design and retrofit problems necessarily involve much longer time horizons than typical planning and scheduling problems because equipment purchase and decommissioning plans are typically carried out over several years. This longer time scale of interest necessarily introduces significant uncertainty, especially in the prediction of demands that will be placed on the process. Thus process design and retrofit is typically considered a stochastic optimization problem that seeks to optimize probabilistic performance. This performance usually involves maximizing expected net present value, minimizing expected cost, controlling risk of capital loss, or maximizing an abstract measure of process flexibility (Subrahmanyam, Bassett, Pekny & Reklaitis, 1995; Subrahmanyam, Pekny & Reklaitis, 1996; Epperly, Ierapetritou & Pistikopoulos, 1997).

2.3. Model predictive decision-making

In practical applications decision problems are never solved in isolation, rather decision problems arise periodically and must use as their starting point plans previously in place as part of their solution. For example, many processes typically are scheduled once a week or every few days. In determining a current schedule, the execution in progress of a previous schedule, the available labor patterns, the arrival of raw materials, and the transportation schedule for products are fixed by decisions made from having solved previous scheduling problems. Thus decision problems are frequently experienced as a related sequence in time. The solution of the current problem strongly depends on the previous members of the sequence and will affect the solution of future problems. Thus in solving a current problem, consideration must be given to the types of uncertain events that will perturb the ability to follow a prescribed solution so that future problems can be effectively solved. Model predictive scheduling, planning, or design problems have a close analogy to model predictive process control (see the summary of Qin & Badgwell, 2002).

2.4. Warehouse management

In abstract terms, the warehouse management problem can be considered as a special kind of scheduling problem. In particular, a warehouse consists of a number of storage locations with capacities; there are a number of resources that must be scheduled (e.g. laborers, fork trucks, etc.) to accomplish activities; and there is a 'raw' material and 'product' delivery schedule. Of course in the warehouse management problem, the 'raw' materials are goods moved into the warehouse and the 'product' materials are goods moved out into a transportation network. Historically the warehouse management domain has been served by highly specialized software. However, as upstream and downstream production becomes more tightly integrated to warehouse management, the problems to be solved take on characteristics of both. The goal in warehouse management is to store material in such a way that the time to retrieve a set of material is minimized or that the cost of warehouse operation is minimized. This goal is affected by how incoming material is assigned to storage locations and how cleverly resources are used to accomplish activities (Rohrer, 2000).

2.5. Supply chain design and operation

The supply chain operation problem extends the scheduling problem in the spatial dimension and considers the coordinated management of multiple facilities and the shipment of materials through an associated transportation network. Because of the enormous size of supply chain management problems, significant detail is necessarily suppressed in their solution. Similarly the supply chain design problem addresses questions such as where a warehouse or manufacturing facility should be located and to which existing site additional capacity should be added. The increased scope of supply chain management problems and their long time horizons imply that they must be solved under significant uncertainty. Another dimension involved in supply chain management are the game theoretic aspects of cooperation, competition, and market factors. In particular many supply chain management problems involve multiple entities, for example some of whom are suppliers, some of whom are customers, and some of whom are competitors. This added dimension raises significant additional strategic questions such as those concerning the pricing of products, when to offer promotions, which type of incentives to offer customers to provide accurate demand forecasts, and how much inventory to hold to counter market or competition shifts (see for example Tsay, 1999). These strategic and game theoretic issues interact with the capabilities and scheduling of the facilities involved in the supply chain.

2.6. Product and research pipeline management

The development of new products provides a competitive advantage for many companies and offers the possibility of superior profitability when done consistently well. For example, the top-tier companies of the pharmaceutical industry possess billion-dollar drug development pipelines whose goal is to turn out blockbuster products that provide substantial returns and underwrite the cost of many candidates that fail to make it to market. The product and research pipeline management problem has much overlap both with supply chain management and process scheduling problems. In particular, correctly answering many management questions depends on insight into how competitors will behave. This follows because the first company to market with a particular product type often reaps the vast majority of market share and return relative to competitors that successively enter. Besides game theoretic issues, the many resources in a pipeline must be coordinated to maximize the development throughput. Unlike manufacturing scheduling problems, the scheduling of a product and research pipeline involves a great deal of uncertainty as to whether all the tasks associated with the development of a particular product will be necessary. This is particularly true of the pharmaceutical industry when candidate drugs can be eliminated from consideration based on safety and efficacy testing. The anticipated failure of a significant fraction of all candidates leads to the concept of 'overbooking' a pipeline whereby more tasks are scheduled for possible completion than could be achieved if all the candidates successfully survived. The difficulty in addressing practical problems is achieving the proper amount of overbooking so that resources are fully utilized when attrition occurs but are not overtaxed (Honkomp, 1998).

2.7. Summary of features common to applications

As mentioned, all the above applications involve the management of activities over a time horizon. However, the discussion above also indicates several other commonalities. In particular successful solution of any of these problems involves utilizing information that is not known with great precision, for example demand forecasts or which tasks will fail in a product and research pipeline. This uncertainty implies that there will be risk in however the answers to the yes/no questions are made. One challenge is to determine which pieces of information are the most important to a good solution. Another challenge is to select an answer to a problem that is the most effective over a wide range of possible values of the most critical pieces of information. This last issue necessarily implies that risk management is an important issue in many practical applications and that steps must be taken because of uncertainty in underlying information that would be unnecessary if the information where completely known. Put succinctly, the above applications involve (i) management of activities on a timeline, (ii) significant combinatorial character because of the large number of yes/no decisions implied by practical applications, (iii) uncertain information which can have a significant impact on the best answer, (iv) risk because the uncertainty might be realized in a way that is detrimental, and (v) game theoretic aspects because in many problems multiple entities interact.

3. Mathematical programming approach for process management problems

A practical approach to process management problems must be able to address each of the technical issues summarized in the previous section as well as a variety of business process and human factors issues (see Bodington, 1995). The paper by Shobrys and White (2002) reviews typical and best practices for addressing many process management problems using traditional technologies. This paper focuses on three related technologies that are general enough to address the applications described above, but which are amenable to the significant customization necessary for practical use. The paper by Pekny and Reklaitis (1998) reviews various technologies and classifies them according to how they address the intrinsic computational difficulty of most process management problems. As they discuss virtually all process management applications are NP-complete, the practical implication of which is that there is an uncertainty principle that must be addressed in the development of any solution approach. In particular, a given algorithm for a process management problem cannot guarantee both the quality of the answer and that the worst case performance would not be unreasonable in terms of execution time or computational resources. That is an algorithm might take an unreasonable amount of time to get a provably good answer or might get an arbitrarily poor, or no answer, in a predictable and reasonable amount of time. Another useful way of interpreting the uncertainty principle is that any given algorithm that performs well on one process management problem will exhibit poor performance on some other process management problem. This last statement sometimes seems to contradict intuition, but can be understood by considering the great variety of process management problems and the fact that dominant features in some problem will represent a combination of constraints on which previously known solution strategies are ill-suited. The importance of the uncertainty principle to users, developers, and researchers of process management software cannot be overstated and indeed not

appreciating the practical implications of the uncertainty principle is a key cause of failure when process management software does not work. The following example illustrates the implications of the uncertainty principle on a simple problem.

3.1. Example: how an algorithm can fail when the problem changes character

To understand the implications of the uncertainty principle on a simple problem, consider a Traveling Salesman Problem (TSP) when all the cities to be visited lie on a circle of a certain radius (circle-TSP). An optimal solution to the TSP involves specifying the order in which cities are to be visited, starting and ending at the home city, so that the travel distance is minimized. For the circle-TSP an algorithm guaranteed to find an optimal solution involves starting at the home city, choosing the closest unvisited city as the next city in the sequence, and repeating until all the cities have been traversed and then returning to the home city. Of course this 'greedy' algorithm traces out a route in the order in which cities appear around the circle (see Fig. 2). Now, when this same greedy algorithm is applied to a TSP whose cities appear at random locations in a Euclidean plane its performance is very poor because the greedy strategy does not take steps to prevent having to traverse a long distance after most cities have been visited (see Fig. 3). The greedy algorithm is an example of an algorithm whose execution time is reasonable but whose solution quality can be very bad. An alternative to the greedy algorithm is to exhaustively enumerate all possible solutions and choose the one with the minimum travel distance. Such an approach will guarantee arriving at a best possible answer, but the execution time will be unreasonable for all but the smallest problems. The papers by Miller and Pekny (1991) and Pekny and Miller (1991) discuss the tradeoff between solution quality and performance in more detail for versions of the TSP relevant to several process scheduling applications. The TSP is one of the most well known problems in the combinatorial optimization literature and many highly engineered solution approaches have been developed that perform well on many known classes of TSP instances-achieving optimal or near optimal solutions with reasonable effort and probability (Applegate, Bixby, Chvátal & Cook, 1998). However, even for these highly engineered algorithms, instances can be constructed that cause them to produce poor answers or take unreasonable amounts of effort. When confronted with such a problematic instance, existing TSP solution approaches can be adapted to provide better performance. Indeed this kind of iterative challenge of TSP approaches has been a motivating factor in their improvement. Thus TSP algorithms are an example of a research area that has benefited from iterative refinement motivated by the challenge of problematic cases (for more information see the online bibliography by Moscato, 2002). This need for iterative refinement is central to the engineering of algorithms for any NP-complete problem.

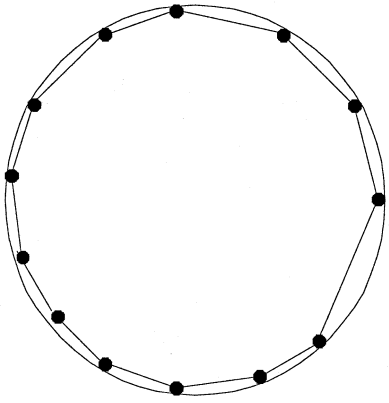

Fig. 2. A 'Greedy' nearest neighbor algorithm works well on a circle-TSP.

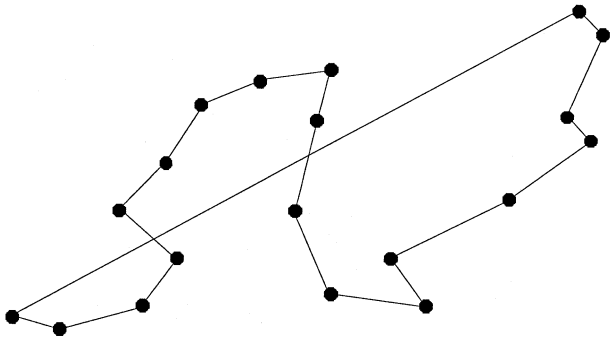

Fig. 3. A 'Greedy' nearest neighbor algorithm works poorly on a more generalized TSP.

3.1.1. Life cycle issues associated with the use of algorithms for practical problems

The key implication of the uncertainty principle is that the development and use of process management solution algorithms is an engineering activity. The starting point of this engineering activity is that the objectives of the solution algorithm must be clearly stated. These objectives often include:

1. guarantee of optimal solution,
2. guarantee of feasible solution,
3. guarantee of reasonable execution time,
4. permits (requires) a high degree of user input in obtaining an answer,
5. high probability of obtaining a good answer in reasonable time on a narrow and well-defined set of problem instances with limited user input,
6. easy to modify, adapt, and improve as new applications or unsatisfactory performance is encountered,
7. low cost of development, and

8. provides an intuitive connection between solutions and input data, that is explains why a given solution was obtained with the specified input data.

The theory of NP-completeness shows that insistence on (1) or (2) as an objective precludes achieving (3) and vice versa. Experience shows that objectives (4) and (7) are highly compatible, but put much of the burden of problem solution on the user. Experience also shows that objectives (5) and (6) seem complementary, but inconsistent with objective (7). Given the nature of NP-completeness and the existence of the uncertainty principle, objective (5) is a practical specification of algorithm engineering success and objective (6) speaks to how well an algorithm engineering approach supports the life cycle encountered in applications. In this context life cycle means the *delivery* of a solution technology for use in practical applications, providing for user *control* of the nature of answers to allow for incorporating considerations that are difficult to formally capture, *support* of a technology in the field when deficiencies arise, and *extension* of a technology both to make an individual application more sophisticated and for use in a broader range of applications. An important part of user control is embodied in objective (8) whereby a solution can be rationalized and implemented with confidence because it is intuitively understood to be valid. This is especially critical for large-scale applications that involve enormous amounts of data, large solution spaces, and where an effective solution might represent a significant departure from past practices. A key problem implied by (8) is the resolution of infeasible problems. In those applications where tools are most useful the chance for specifying an infeasible problem is greatest (e.g. too much demand for the given capacity, insufficient labor to execute all tasks, etc.). In addition to being practically valuable, addressing objective (8) is an interesting research problem given that there are sometimes many ways to resolve infeasibilities and discovering and presenting these many options in a meaningful way is itself a very difficult theoretical and computational problem (see Greenberg, 1998). The paper by Shobrys and White (2002) suggests that objective (8) is crucial to the sustained acceptance of sophisticated modeling tools. Given the speed with which information is being generated, the associated opportunities for improving efficiencies, and the need to rapidly reduce research advances to practice, all these life cycle considerations (delivery, control, support, and extension) are becoming an important part of PSE research. The remainder of this section discusses how mathematical programming approaches to process management problems promote support of life cycle issues and the remaining sections discuss their extension into risk management and game theoretic applications.

3.1.2. Mathematical programming framework to address time-based problems

From an engineering standpoint, mathematical programming based approaches to process management problems offer a number of advantages. In particular a mathematical programming based approach can deliver many combinations of the objectives (1)–(8), depending on how it is implemented and used. For example, straightforward mathematical programming based approaches will deliver objectives (1) and (2) at the expense of ignoring (3). Experience also shows that mathematical programming approaches can be used to pursue objectives (5) and (6) at the expense of relaxing objective (7), see for example Miller and Pekny (1991). Furthermore, mathematical programming approaches can easily support objective (4) and, as with other approaches, effective user input can make problems much easier to solve. As our goal is the treatment of application life cycle issues, the remainder of this section will discuss the use of mathematical programming approaches to achieve objectives (5) and (6) with the understanding that the same approaches can be used to achieve objective (1) and (2) or utilize objective (4), when necessary. With regard to addressing objective (8) mathematical programming methods have the advantage of a systematic framework for relating parameters and constraints, but the combinatorial complexity and ambiguities inherent in explaining solution structure and resolving infeasibilities remains an important issue to be addressed by research (see Greenberg, 1998).

The starting point of a mathematical programming based approach is a formulation of a process management problem:

Maximize or Minimize $f(x,y)$

Subject to:

$h(x,y) = 0$

$g(x,y) \geq 0$

$x \in \{0,1\}^m$, $y \subseteq R^n$

The objective function of a process management problem is typically to minimize cost, maximize profit, minimize tardiness in the delivery of orders, etc. The equality constraints typically implement material balance or resource assignment constraints. The inequality constraints typically put restrictions on inventory and resource usage (e.g. labor). The binary variables (x) represent the discrete choices available when solving a process management problem, e.g. should an activity be executed on a given piece of equipment at a given time. The variables (y) represent the values of continuous quantities such as inventory levels or the amount of material purchased. There are two strategies for formulating process management problems. One strategy creates time buckets and associates variables and

constraints with these buckets. Another strategy controls the sequencing of activities and only implicitly represents time. Regardless of approach, the management of activities in time almost always makes the size of mathematical programming formulations for practical problems quite foreboding.

3.1.3. Implication of the use of time bucketed formulations

The way that time is managed is critical to the success of solving the process management problems discussed above. We primarily use formulations where time is divided into buckets and constraints are then written on variables that are associated with the time buckets. For example, the material available in time bucket k is equal to the material available in time bucket $k-1$ plus the material that becomes available in time bucket k due to production minus the material that is consumed. Such material balance constraints must be written for every time bucket and material. As another example, consider that restrictions on the allocation of equipment yield constraints that have the following form:

$$x_{\text{mixing\_activity\_A,mixing\_equipment,12 noon}} + x_{\text{mixing\_activity\_B,mixing\_equipment,12 noon}} = 1$$

This simple example shows that only one of mixing activity A or B must be started on a piece of mixing equipment at 12 noon and shows how easily intuition may be expressed.

The work of Elkamel (1993), Kondili, Pantelides and Sargent (1993), Pekny and Zentner (1994), and Zentner, Pekny, Reklaitis and Gupta (1994) discusses time bucketed formulations in detail. Time bucketed formulations offer many advantages in applications and support life cycle needs: (i) they are readily extensible to account for additional problem physics, (ii) they are easy to understand since they are based on straightforward expressions of conservation principles and other problem physics, and (iii) solution algorithm engineering is often facilitated because reasoning can be localized due to the fact that variables and constraints only directly affect small regions of time. The chief drawback to time bucketed formulations is the size of formulation when practical problem detail is required. For example, consider that the need to provide 2 weeks worth of timeline management with 1-h accuracy implies 336 time buckets and can easily translate to thousands or tens of thousands of binary variables for practical applications. Furthermore, 1-h accuracy is insufficient for scheduling many critical activities, e.g. the need to use labor for 10 min at the start and end of an activity. In fact the time buckets must be chosen to be as small as the smallest time interval of interest. This type of reasoning quickly leads to the conclusion that explicit formulation of problems using a time bucketing strategy leads to enormous formulations and long solution times for most practical problems. This is because these large formulations take significant time to generate and an enormous amount of memory to store them, irrespective of the cost of actually obtaining a solution. Fortunately, most of the constraints in time bucketed formulations of practical problems do not contribute meaningful information to problem solution because they hold trivially, e.g. zero equals zero. In practice, generation of these trivially satisfied constraints is not necessary. However, one does not realize which constraints matter until after a solution has been obtained. Exploiting the large number of trivially satisfied constraints and avoiding the generation of large explicit formulations is an engineering necessity if a time bucketed mathematical programming formulation is to be delivered for use in practical applications.

3.1.4. Implicit techniques to avoid large formulation sizes

The discussion in the preceding paragraph illustrates the importance of considering all aspects of algorithm engineering when deciding how to approach solution of an important class of practical problem. Whereas a common means of using mathematical programming approaches is to generate the formulation and then solve it, this explicit generation of a formulation is out of the question for many practical process management problems. Instead, we choose to pursue implicit generation of only the nontrivially satisfied constraints and nonzero variables that are necessary for defining the solution. Polyhedral cutting plane techniques also use this strategy by utilizing separation algorithms to detect and iteratively enforce violated members of an exponentially large constraint family (Parker & Rardin, 1988 and see the tutorial work of Trick, 2002). The b-matching paper of Miller and Pekny (1995) illustrates this implicit generation of both constraints and variables and their approach is summarized in the following example.

3.2. Example: implicit formulation and solution of the b-matching problem

The b-matching problem with upper bounds may be formulated as an integer program on an undirected graph $G=(V,E)$ with integer edge weights c_{ij} and integer edge variables x_{ij} as follows:

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1}$$

subject to:

$$\sum_{j|(i,j)\in E} x_{ij} = b_i, \quad \forall i \in V \tag{2}$$

$$0 \le x_{ij} \le d_{ij} \tag{3}$$

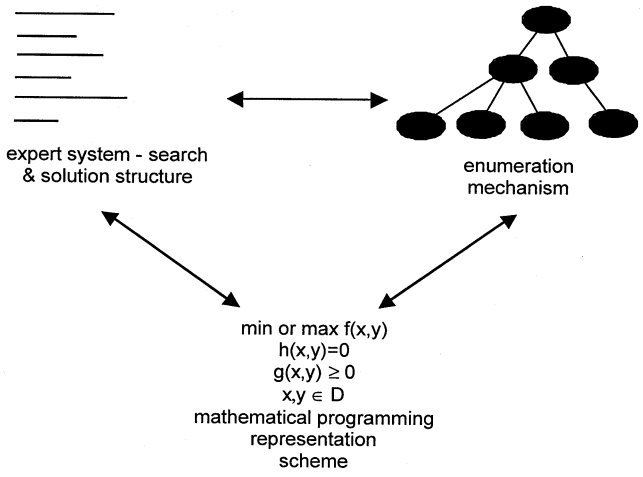

Fig. 4. Mathematical programming architecture for solving large-scale process management problems.

$$x_{ij}, d_{ij} \in Z^{+} \tag{4}$$

Relaxing the integrality constraints yields a Linear Program (LP) that admits only integral and half-integral solutions. The convex hull of the b-matching polytope as given by Padberg and Rao (1982) is useful for developing a highly customized solution algorithm for the b-matching problem. Let $R \subset V$ with $|R| > 2$, $\delta(R)$ be the set of all edges with exactly one end in R, and T be a non-empty subset of $\delta(R)$. A parity of R,T is defined to be even or odd depending on the parity of:

$$b(R) + d(T) = \sum_{i \in R} b_i + \sum_{e \in T} d_e \tag{5}$$

For this example $\forall R,T$ is taken to mean all combinations of R and T with odd parity. Following Padberg and Rao (1982), the facet defining inequalities may be written as:

$$\sum_{(i,j) \in \delta(R) \setminus T} x_{ij} + \sum_{(i,j) \in T} (d_{ij} - x_{ij}) \geq 1, \quad \forall R,T \tag{6}$$

The implicit solution strategy of Miller and Pekny (1995) operates on the LP implied by (1)–(3), and (6) to obtain a provably optimal solution in a number of steps polynomial in the cardinality of V. Note that (6) implies a number of constraints that scales exponentially in the cardinality of V. The basic idea behind the implicit solution strategy is to use a network flow algorithm, an implicit and highly customized linear programming solution technique, to solve (1)–(3). The resulting solution will have integral and half-integral values. The half-integral values are eliminated by growing a tree in graph G whose nodes represent either members of V or individual constraints from (6). Cycles may be found during growth of the tree, which identify other nontrivial constraints from (6) which are then directly enforced. The root of the tree is a violated member of (6) consisting of a cycle of half-integral variables. Tree growth terminates when another violated member of (6) is encountered and then the half-integral variables can be made integral by modifying the variables values of the half-integral cycles and in the tree according to a well-defined pattern (see Miller & Pekny, 1995). Thus growth of the tree identifies the nontrivial constraints of (6) which must be enforced. The nonzero variable values are identified by the network flow algorithm and tree growth. The highly customized implicit solution strategy summarized in this example has successfully solved problems with up to 121 million integer variables in less than 10 min of personal computer time (1 GHz, 128 Mb of RAM). Of course most of these variables actually take on a value of zero in the solution and only a few of the constraints in (6) actually have to be enforced. See Miller and Pekny (1995) for a complete description of the algorithm and a discussion of computational results.

Fig. 4 illustrates the basic concepts behind a mathematical programming architecture for solving process management problems. The three main components are (i) implicit formulation—for example based on a time bucketed strategy (bottom box), (ii) an expert system which controls the pivoting strategy used to solve the linear programming relaxation of the formulation (top left box), and (iii) an expert system that controls the branching rules used during implicit enumeration (top right box). The b-matching example given above provides a simple example of implicit formulation generation and corresponding linear programming solution. Unlike the b-matching problem, process management applications produce linear programming relaxations that are arbitrarily integer infeasible and much less structured. In these cases the expert system chooses pivots to provide for rapid linear programming solution and to prevent the introduction of unnecessary infeasibilities. The work of Bunch (1997) provides an example of how pivot rules (i.e. an expert system for pivoting strategy) can be used to avoid unnecessary infeasibilities. Avoiding such infeasibilities dramatically speeds solution since unnecessary implicit enumeration is avoided to remove these infeasibilities. For the b-matching problem described above the pivot rules described in Bunch (1997) reduce the computational complexity of the guaranteed detection of violated constraints from $O(n^4)$ to $O(n)$, where n is the number of nodes in the graph. The luxury of using a computationally cheaper algorithm to detect violated constraints results in order of magnitude speedup in problem solution both because of faster convergence and faster iterations. The advantage of the work of Bunch (1997) is that conventional linear programming solvers that allow control of entering and exiting basic variables may be used to implement the technique.

Whereas implicit generation of mathematical programming formulations is crucial to the delivery of the

technology on practical applications, the architecture of Fig. 4 promotes extension of applications to encompass new features. In particular, the solution logic is distributed across the implicit generation of the formulation, the implicit enumeration expert system, and the expert system for solving the LP. Both expert systems are designed to operate on the formulation and therefore are one level of abstraction removed from the problem details. The expert system logic is coded to obtain solutions to the formulation and is not directly designed to deal with particular problem instances. Thus when the formulation is changed to encompass more details or features, the basic expert system strategies for driving out infeasibilities are still valid. Of course with due consideration to the uncertainty principle described above, significant departures in problem features from those that the expert systems have been well-designed to handle may cause a dramatic rise in solution time. This is especially true on large problems where even minor inefficiencies can result in unreasonable solution times due to the extremely large search spaces implied by a large number of integer variables. When problematic cases occur, the abstraction of the solution logic underlies a systematic approach for associating the cause in the solution logic to an undesirable solution or unreasonable execution time. That is, the features of the problem can be rationalized against the particular sequence of pivots and the search tree path, which promotes physics based reasoning for failures and expert system modifications that prevent encountering them.

3.2.1. Mathematical programming architecture and life cycle considerations

The paper by Pekny and Zentner (1994) discussed the basic mathematical programming *delivery* architecture shown in Fig. 5 for process management applications. The basic components are (I) graphical user interface, (II) object oriented and natural language problem representation language, (III) mathematical programming formulation, and (IV) solution algorithm (see also Zentner, Elkamel, Pekny & Reklaitis (1998)). Obviously a major benefit of the architecture of Fig. 5 is insulating the user from the details of the mathematical programming formulation and algorithm. With respect to life cycle considerations and keeping in mind the need to use implicit formulations, the graphical user interface shown in Fig. 5 (I) is the means by which user *control* is provided. In particular, the elementary and fundamental nature of the variables in time bucketed mathe-

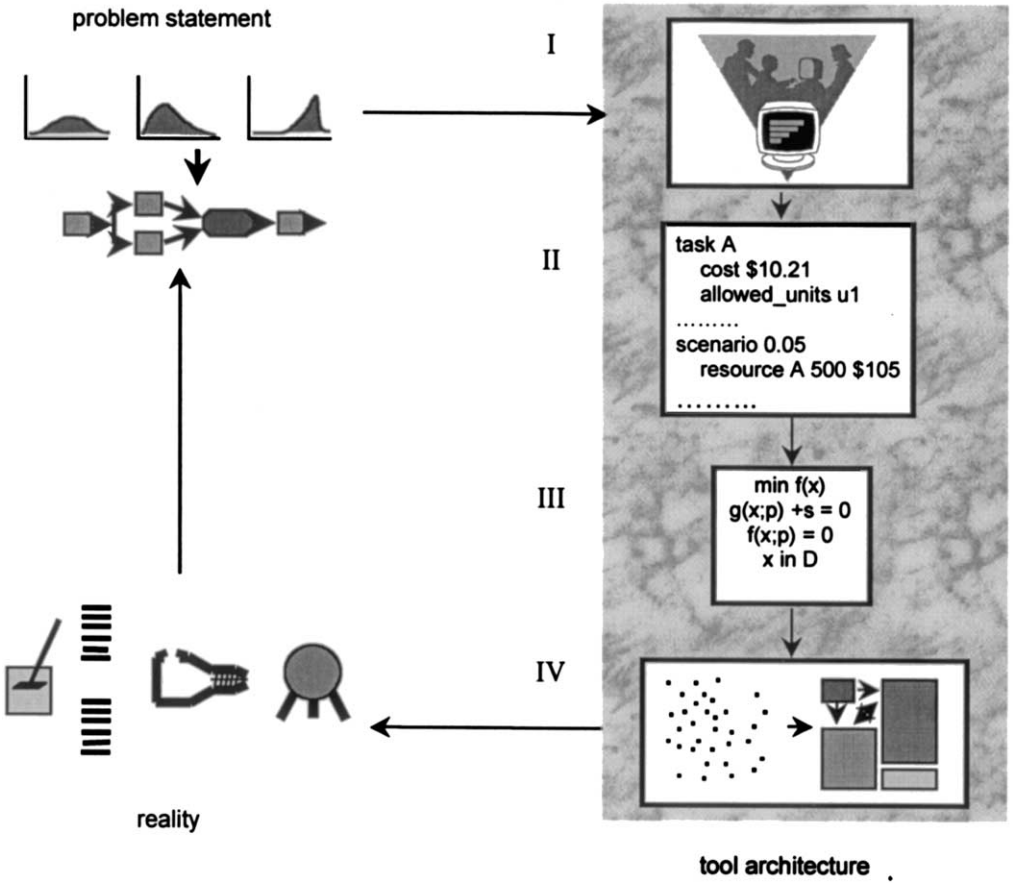

Fig. 5. Tool architecture promotes support of life cycle considerations.

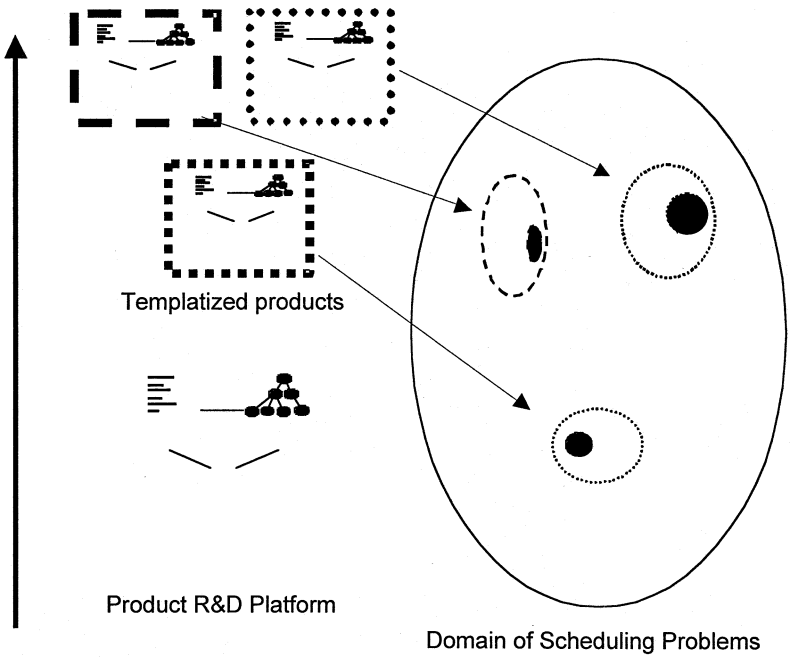

Fig. 6. Robust deployment through tool templatization.

matical programming formulations provides for many classes of user control. For example the simplest class of user control is to restrict a variable or group of variables to a particular value. With respect to management of time and equipment this class of user control can represent restricting (preventing) an activity to (from) a group of equipment or set of resources. Alternatively, this class can restrict activities to a particular time interval. Another class of user control possible is to direct the way in which the formulation is solved. For example, in a scheduling application a user can specify that a particular demand be assigned resources before any other demand. This has the effect of scheduling activities to satisfy the demand in a more left most position on the time line than demands that are scheduled subsequently. An interrupt driven class of control is also possible whereby whenever an expert system considers manipulating a group of variables the user can be prompted for direction as to how to proceed.

The architecture of Fig. 5 also promotes the *support* of applications. Firstly, deciding where components of Fig. 5 execute in a client/server-computing environment can be made on the basis of the needs of the application. Second, the component represented in Fig. 5 (II) supports communication of problematic instances since the object oriented language description can be electronically communicated to facilitate recreation of the problem in a support environment. Perhaps most importantly from the standpoint of support, the architecture of Fig. 5 can be restricted to particular classes of problem instances where the probability of failure can be kept low. Fig. 6 illustrates the concept of a 'templatized' tool whereby the graphical user interface and problem description component are adapted to only permit the creation of instances that fall within a class for which the tool has a high probability of solution success in terms of quality of answer and reasonable performance, e.g. one of the problem types described in Table 1. In terms of product support there may only be one solution engine underneath all templatized tools, but the templates prevent users from applying the tool outside the testing envelope. This eliminates many potential support problems, but still allows for *extension* of the tool by relaxing the template restrictions. In terms of the circle-TSP example given above, templatization would amount to restricting the input to the greedy algorithm to problems that are circle- or near circle-TSPs. By avoiding arbitrary Euclidean TSP instances a tool based on the greedy algorithm would not exhibit poor behavior. For any given algorithm architecture, one qualitative measure of its versatility is the number of different problem types and the size of problem it can effectively address without additional algorithm engineering. Experience shows that the promise of the algorithm abstraction of Fig. 4 is that by having the solution logic operate on the formulation instead of the problem information directly that the versatility is enhanced.

3.2.2. A qualitative rating system for process management applications

Given the uncertainty principle implied by NP-completeness, a system for rating the effectiveness of solu-

tion algorithms provides a useful way of differentiating different approaches and implementations. Table 2 describes a qualitative rating system for process management solution algorithms. The first column of Table 2 shows the various classifications in the rating system, the second column provides a short description of the classification, and the third column shows the resources/techniques that are likely to be required to move an algorithm from one classification to a more capable classification. The qualitative rating system of Table 2 embodies the uncertainty principle, is suggestive of the different ways that solution algorithms can be used and suggests that more robust behavior requires greater algorithm investment. One of the major advantages of mathematical programming approaches is that, with suitable investment, they can support the entire range of capability depicted in the first column on a variety of problems through a formal and well-defined series of iterative refinements (also see the discussion on problem generators in Kudva & Pekny, 1993).

3.3. Example: detailed process scheduling application and discussion of mathematical programming approach

This example discusses a mathematical programming approach to a practical example from the modeling perspective through a model predictive application. The example is based on the Harvard Applichem case study (Flaherty, 1986) with the extensions described in Subramanian (2001). The first part of the discussion summarizes the processing details and the second part describes the application.

3.3.1. Recipe network

The Release-ease family of products moves through a similar recipe, which consists of four tasks. The first task involves the all-important reaction where four raw materials A, B, C and D are added in a precise sequence, along with miscellaneous substances including water (termed as 'Other') to yield a formulation of Release-ease particles. The Release-ease particles formed move over a conveyer belt mesh in a cleaning task where the liquid waste falls through. The filtrate particles are further dried and packed in different sizes. The recipe is represented in Fig. 7.

The Gary, Indiana Applichem plant makes eight formulations in two grades, commercial and pharmaceutical, and packs them in 79 packaging sizes selling a total of 98 products.

3.3.2. Equipments/machines

There are four categories of equipment used for the production: reactors, conveyer belts, driers and packaging lines. The first task is executed in a reactor in batch mode. Three reactors with capacities 1000 and 8000 lbs along with four more of an intermediate capacity of 4000 lbs are available, totaling 10 in all. The processing times and Bills of Materials (BOM) depend on the type of the formulation of Release-ease and on the batch size. The reactors have to be washed out every time a batch is executed. Two identical conveyer belts filter the formed Release-ease. This is modeled as a batch task with processing times increasing proportionally with amount to be filtered. Three identical driers dry the filtered Release-ease in batches. Eight packing lines pack the eight formulations into 79 sizes making 98

Table 2
Algorithm support classifications

Algorithm capability classification	Description	Examples of resources required to improve
Unsupported	Algorithm is incapable of even small test problems with all constraint types	Extension of algorithm framework, fundamental algorithm modifications, intense R & D activity
Demonstration	Algorithm is capable of demonstrating solutions but solutions are not implementable in practice, need sculpting, and key constraints are not acceptably handled	Extension of algorithm framework to shape schedules according to user desires, interactive user interface
Engineering	Algorithm is capable of generating useful results, but model solution is not sufficiently robust to use in practice or some less important features are neglected which are of practical interest for routine use	Extension of algorithm framework to shape schedule according to user desires, programmatic user interface, interactive user interface
Routine	Algorithm routinely generates useful schedules that are implemented in practice through human interpretation, routine usage periodically results in the need for bug fixes or slight algorithm enhancements	'Designed' testing problems and generators to test algorithm robustness, periodic minor coding fixes or slight algorithm enhancements
Online model predictive	Algorithm is extremely robust in generating schedules that are faithful to shop-floor operations and can be used to predict shop-floor activity	Generators to test algorithm robustness in an advancing time window fashion (SIMOPT based testing—see later section)

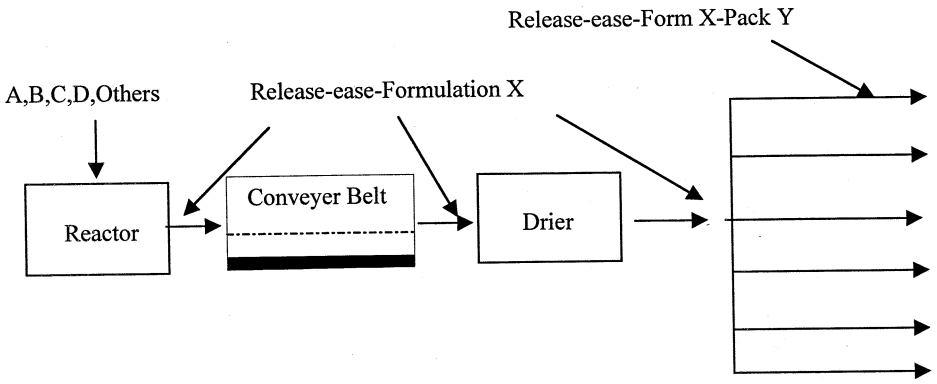

Fig. 7. Release-ease recipe.

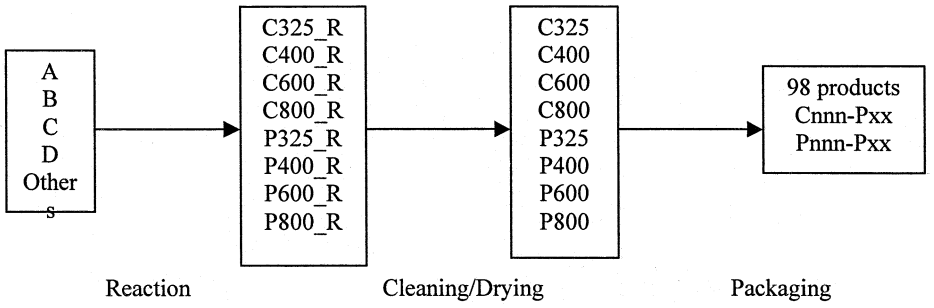

Fig. 8. Material flow in the plant.

Stock Keeping Units (SKU) in all. It is assumed that the eighth packing line is dedicated to packaging commercial grade formulations, and hence packages faster than the other packing lines. There is no intermediate storage prescribed after the cleaning task. It is assumed that intermediate storage vessels are available after the reaction and drying tasks.

3.3.3. Products

The flow of materials (raw materials, intermediates and final products) in the system is shown in Fig. 8. The eight formulations are classified by grade (commercial (C) and pharmaceutical (P)) and particle mesh size (325, 400, 600 and 800). Fig. 9 lists the final 98 products sold, differentiated by the packing type or size in which a formulation (such as C325) is packaged. P-xx refers to a packing type, and is assumed to correspond to a particular size or shape.

3.3.4. Processing characteristics

3.3.4.1. Reaction task. The BOM for this task involves initial addition of B, C and Other together and upon completion of this reaction, addition of A and D. The processing times are dependent on the grade and batch size. The processing times, in minutes, are provided in Table 3. It is assumed that making 600 and 800 mesh size particles takes M times (see below) longer than 325 and 400. It is further assumed that making P grade takes N times (see below) longer than C grade. For $M = N = 1.2$, processing times for all the grades are provided in Table 3. The addition of A and D is assumed to take place after 10% of the overall processing time has elapsed.

3.3.4.2. Reaction wash-outs. The reactors have to be cleaned after every batch execution. Washout times are longer for larger reactors and after making P grade, due to stricter environmental regulations. Washout times are presented in Table 4.

3.3.4.3. Cleaning task. The cleaning times over the conveyer belt increase proportionally with the amount being filtered. The processing times are given in Table 5.

3.3.4.4. Drying task. Drying times are slightly longer for larger batches. The drying times are listed in Table 6 as a function of output dried Release-ease quantities.

3.3.4.5. Packaging task. There are 79 distinct packing types based on size, and the packing times depend on these types. The packing times are assumed to be independent of the formulation being packaged. Table 7 lists the packing times in hours, for all the 98 final products along with the size they are sold in, in kilograms.

Table 3
Processing times in minutes for the 8 grades

Batch Size (lbs)	C325_R, C400_R
1000	50
4000	75
8000	125

Batch Size (lbs)	P600_R, P800_R
1000	72
4000	108
8000	180

Batch Size (lbs)	C600_R, C800_R,P325_R,P400_R
1000	60
4000	90
8000	150

C325-P11	C325-P79	P800-P27
C325-P14	C400-P42	C325-P18
C325-P17	C400-P46	C325-P19
C325-P22	C400-P50	C325-P20
C325-P27	C400-P54	C325-P21
C325-P37	C400-P60	C325-P23
C325-P71	C400-P62	C325-P24
C400-P17	C400-P64	C325-P25
C400-P27	C400-P68	C325-P26
C400-P37	C400-P70	C325-P28
C400-P67	C400-P74	C325-P30
C600-P47	C400-P78	C325-P32
C600-P57	C600-P41	C325-P34
C800-P17	C600-P45	C325-P36
C800-P37	C600-P49	C325-P73
P325-P17	C600-P53	C325-P77
P325-P3	C800-P27	C400-P38
P400-P10	C800-P72	C400-P40
P800-P37	C800-P76	C400-P44
C325-P12	P325-P11	C400-P48
C325-P13	P325-P14	C400-P52
C325-P15	P325-P5	C400-P56
C325-P16	P325-P7	C400-P58
C325-P29	P325-P9	C400-P66
C325-P31	P400-P1	C600-P39
C325-P33	P400-P3	C600-P43
C325-P35	P400-P4	C600-P51
C325-P57	P400-P5	C600-P55
C325-P62	P400-P8	C600-P65
C325-P67	P600-P11	C800-P59
C325-P69	P600-P7	C800-P61
C325-P75	P600-P9	C800-P63
		P400-P2
		P400-P6

Fig. 9. Final products made by the Gary Applichem plant.

Table 4
Wash-out times for reactors

Material Grade	Reactor Size (lbs)	W/O Time (min)
C-Grade	1000	10
P-Grade	1000	10
C-Grade	4000	30
P-Grade	4000	40
C-Grade	8000	45
P-Grade	8000	60

Table 5
Cleaning times for conveyer belts

Batch (lbs)	Filtration Times(min)
1000	5
4000	20
8000	40

Table 6
Drying times for release-ease formulations

O/P Release-ease (lbs)	Drying Times(min)
100	40
400	50
800	60

3.3.4.6. Packing setups. Before a packing line executes the packaging task, a setup time is incurred. This is required because there exists a need to arrange the appropriate packing type in the packing line. These setups are reduced by 50% for the dedicated packing line 8. The setup data is provided in Table 8.

3.3.5. Material requirements and flow

For manufacture of 100 lbs of Release-ease, the raw material requirements, as provided in the original Ap-

plichem case study (Flaherty, 1986), are presented in Table 9. The reactor batch is assumed to contain 10% Release-ease. This 10% figure represents the yield of the process. Therefore, 100 lbs of Release-ease would be produced in a 1000 lbs batch with Other (1000–156.92) making up the total material amount to 1000. Because no intermediate storage is prescribed between the cleaning and drying tasks, they are modeled as a single task, which outputs the dried Release-ease particles in amounts of 100, 400 and 800 lbs (10% of the Reactor batches). The rest of the material is discarded. These amounts of final product Release-ease are stored in containers and packed into various sizes depending on the demand.

Table 7
Packing size in kg and packing times in hours for all final products

SKU ID	Size	Packing time	SKU ID	Size	Packing time
C325-P11	20	0.166666667	C400-P60	130	1.5
C325-P12	22	0.166666667	C400-P62	150	1.5
C325-P13	24	0.166666667	C400-P64	170	2
C325-P14	25	0.166666667	C400-P66	190	2
C325-P15	26	0.333333333	C400-P67	200	2
C325-P16	28	0.333333333	C400-P68	225	2
C325-P17	30	0.333333333	C400-P70	275	2.5
C325-P18	31	0.333333333	C400-P74	375	3
C325-P19	32	0.333333333	C400-P78	475	3
C325-P20	33	0.333333333	C600-P39	55	0.5
C325-P21	34	0.333333333	C600-P41	60	0.5
C325-P22	35	0.333333333	C600-P43	65	0.75
C325-P23	36	0.333333333	C600-P45	70	0.75
C325-P24	37	0.333333333	C600-P47	75	0.75
C325-P25	38	0.333333333	C600-P49	80	0.75
C325-P26	39	0.333333333	C600-P51	85	0.75
C325-P27	40	0.333333333	C600-P53	90	0.75
C325-P28	41	0.416666667	C600-P55	95	0.75
C325-P29	42	0.416666667	C600-P57	100	1
C325-P30	43	0.416666667	C600-P65	180	2
C325-P31	44	0.416666667	C800-P17	30	0.333333333
C325-P32	45	0.416666667	C800-P27	40	0.333333333
C325-P33	46	0.416666667	C800-P37	50	0.416666667
C325-P34	47	0.416666667	C800-P59	120	1
C325-P35	48	0.416666667	C800-P61	140	1.5
C325-P36	49	0.416666667	C800-P63	160	1.5
C325-P37	50	0.416666667	C800-P72	325	3
C325-P57	100	1	C800-P76	425	3
C325-P62	150	1.5	P325-P3	1	0.016666667
C325-P67	200	2	P325-P5	5	0.016666667
C325-P69	250	2	P325-P7	10	0.083333333
C325-P71	300	3	P325-P9	15	0.166666667
C325-P73	350	3	P325-P11	20	0.166666667
C325-P75	400	3	P325-P14	25	0.166666667
C325-P77	450	3	P325-P17	30	0.333333333
C325-P79	500	3	P400-P1	0.5	0.016666667
C400-P17	30	0.333333333	P400-P2	0.75	0.016666667
C400-P27	40	0.333333333	P400-P3	1	0.016666667
C400-P37	50	0.416666667	P400-P4	2	0.016666667
C400-P38	52.5	0.5	P400-P5	5	0.016666667
C400-P40	57.5	0.5	P400-P6	7.5	0.083333333
C400-P42	62.5	0.75	P400-P8	12.5	0.083333333
C400-P44	67.5	0.75	P400-P10	17.5	0.166666667
C400-P46	72.5	0.75	P600-P7	10	0.083333333
C400-P48	77.5	0.75	P600-P9	15	0.166666667
C400-P50	82.5	0.75	P600-P11	20	0.166666667
C400-P52	87.5	0.75	P800-P27	40	0.333333333
C400-p54	92.5	0.75	P800-P37	50	0.416666667
C400-P56	97.5	0.75			
C400-P58	110	1			

Table 8
Packing setup times for packing lines, X:{1…7}

Product Grade	Packing Line	Setup time(min)
C-Grade	PLX	20
P-Grade	PLX	30
C-Grade	PL8	10

Table 9
Raw materials consumed/100 lb release-ease produced

Raw material	Amount (lbs)
A	20.75
B	53.8
C	53.6
D	28.77
Total	156.92

3.3.6. Product demand

There are two components to how demands are estimated for use in the application below. The first is the product mix for which demands are specified, and the second pertains to the actual demand quantity specified.

3.3.6.1. Product mix. The 98 final products have been classified into three categories with high (0.6), medium (0.3) and low chance (0.1) of demands being specified. The number of products split between these categories is 19, 46, and 33, respectively. If demands for ten products were to be specified, six would be from the high category, three from medium and one from low. Even though 98 products could be made and there is no restriction on how many products can have demands specified, it is assumed that during normal plant operation only about 20–40 products have demands specified in any one time period. This assumption stems from usual industrial practice of scheduling campaigns to smooth out noisy long-term demand signals, and rarely does a plant make 100 different products in a short period of time. For the application described below, the number of SKUs for which demands would be generated is chosen and a particular list of SKUs is obtained by sampling from the list of 98 products.

3.3.6.2. Demand generation. Once the product mix is obtained, a total demand profile is created with the help of a generic demand generator. The total demand over all the SKUs is generated using a sum of different demand signals. These signals are base demand, growth or decay with time, cyclic demands, seasonal demands and random noise. The parameters that control each of the demand signals are set based on the desired characteristics in the total demand profile over time. The demand, $D(t)$, for a period t is given as follows:

$$D(t) = B + Gt + C\sin(\hat{a}t) + S\cos(\hat{a}t) + \text{Noise}$$

Here, B is the base demand, G is the growth or decay of demand, C is the coefficient for cyclic demand with periodicity ã, S is the coefficient for seasonal demand with periodicity ã, Noise is the normally distributed noise, $\sim N(0,\sigma)$, σ is 5% of B.

Given such a demand profile over a specified horizon, the total demand quantity is apportioned amongst the sampled SKUs for every period in the horizon, based on a fixed ratio, thus generating the individual demands for every SKU for every period.

3.3.7. Application

The purpose of the application was to study scheduling algorithm performance over a typical year of Gary plant operation assuming that the plant was restricted to manufacture a specified number of SKUs (5, 10, 20, 40, 80) in any one 2 week period at various possible demand levels. The Virtecs® scheduling system (version 4.021) with the Elise® MILP scheduling solver was used to conduct the study (Advanced Process Combinatorics, Inc., 2001). The plant is assumed to produce in a make-to-stock environment. Given the existence of the model, the study could have investigated the effect of additional equipment, labor, technical improvements to reactor or other equipment performance, the marginal impact of adding SKUs to the product mix, the effect of forecast noise on inventory levels, etc. To conduct the study, the timeline was divided into 52 1-week periods with a reasonable initial inventory condition at the start of the timeline. A 2-week schedule was generated starting with week number 1, the inventory at the end of the week was computed, and then the frame of interest was advanced 1 week and the process was repeated until 52 overlapping 2-week schedules were generated for the 1 year timeline. A typical 2-week Gantt Chart is shown in Fig. 10 and a portion of this Gantt Chart is shown in detail in Fig. 11. The scheduling problem used to generate Fig. 10 contained 178 tasks (reaction, etc.), 129 resources (products, intermediates), and 28 equipment items. The Gantt Chart in Fig. 10 contains 6710 scheduled activities (task instances = boxes on Gantt Chart). The problem in this example is sufficiently large that a time bucketed (5 min buckets) mathematical programming based approach is only possible using an implicitly generated formulation. Table 10 shows the execution time statistics for the Elise scheduling solver on a 400 MHz Pentium II with 256 Mb of memory after approximately 20 h of algorithm engineering effort. Table 11 shows the failure rate of the Elise scheduling solver on problem instances at the beginning of the study prior to any algorithm engineering work. In this case failure is defined to be any 1 year timeline in which the Elise solver required an unacceptably large amount of time to obtain a solution on some scheduling problem along the timeline. For this

study unacceptably large was defined to be more than 1 h of execution time. As Table 11 shows, for schedules involving certain numbers of SKUs the failure rate was as high as 50% and as low as 0%. Upon completion of the algorithm work, the failure rate was 0% as measured over 1300 2-week scheduling problems and the execution time statistics of Table 10 show that the standard deviation of execution times is small relative to the mean indicating reliable performance. In order to conduct the algorithm engineering work several problematic scheduling problems were studied to determine the cause of the unreasonable execution time. In all cases the root cause of execution time failure was determined to be due to numerical tolerances involved in LP solution that had not failed on other problem types. The strategy used in applying the tolerances had to be modified because of significant formulation degeneracy encountered in this application. The nature of these Applichem derived scheduling problems is such that very little enumeration is required so almost all the computational effort is spent on LP solution.

The scheduling literature contains several examples of applications similar in size or larger that are solved using highly customized heuristics (Thomas & Shobrys, 1988; Elkamel, Zentner, Pekny & Reklaitis, 1997). However, these heuristics are usually applicable only to the specific problem for which they were designed. Many of these heuristics lack the framework to express, let alone solve, such problem extensions as adding another level of processing, parallel equipment, sequence dependent changeover, shared or process vessel storage, or combinations of these features. Some heuristic algorithms cannot address the addition of such a commonplace constraint as binding finite capacity storage. From an algorithm engineering point of view these heuristic algorithms can be developed inexpensively relative to customized mathematical programming based approaches. However, they lack extensibility since their underlying specialized representations cannot describe features beyond the problem for which they were designed. Lack of extensibility has significant practical implications. Processes typically evolve over time due to changing prices of raw materials, new products, pressure from competitors, etc. An application that is wholly acceptable when installed, will certainly fall into disuse if the underlying solution technology proves too brittle to follow process evolution. Mathematical programming based approaches have the advantage that the solution methodology is built on an abstract problem representation. The representation can often be changed easily to accommodate process evolution (Zentner et al., 1994). Sometimes the solution methodology will work as is, but NP-completeness guarantees that this cannot always be the case. This leads to the

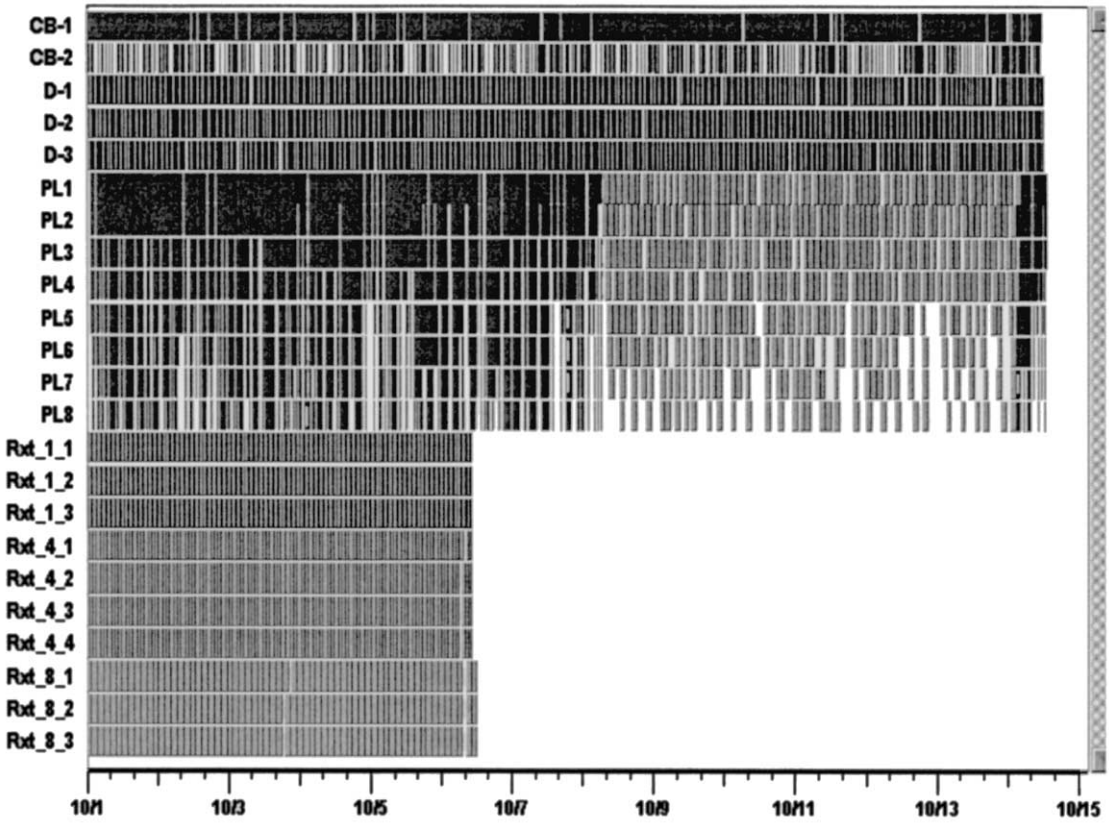

Fig. 10. Two-week schedule for applichem case study.

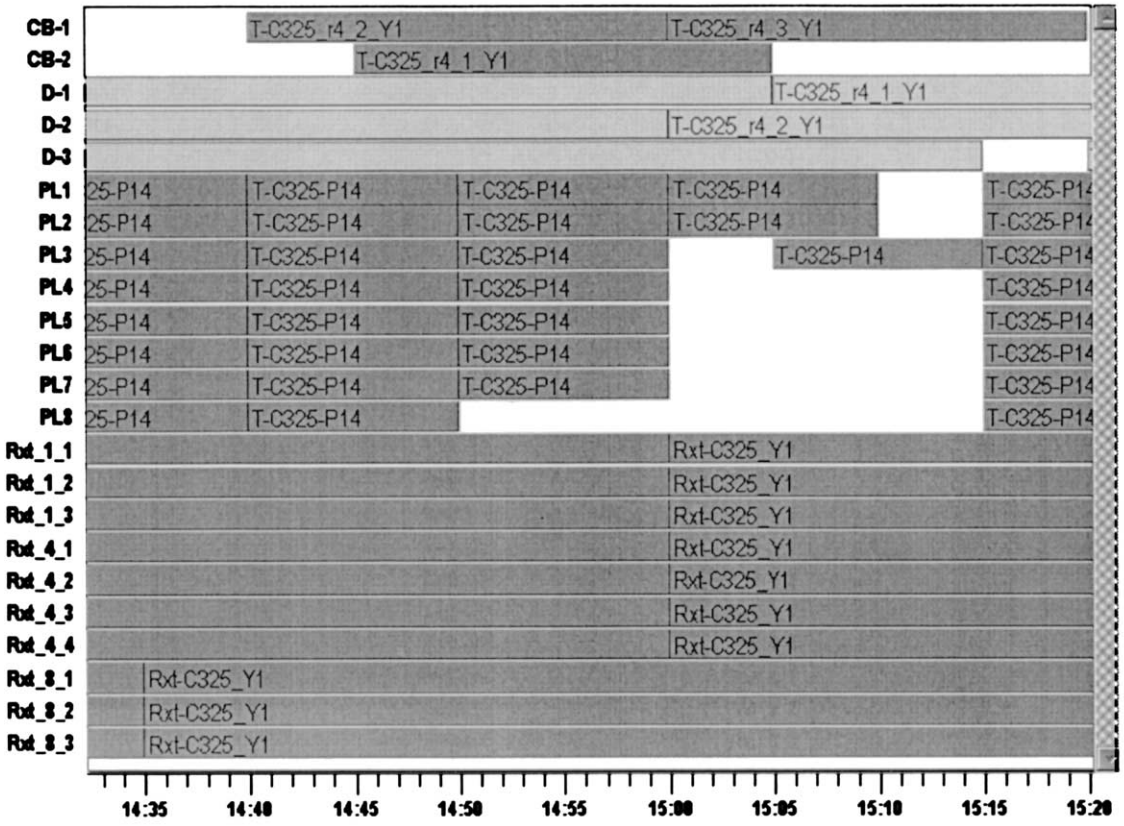

Fig. 11. Grantt chart showing schedule detail.

notion of inductive development. A solution algorithm based on an abstract representation will have been engineered to solve a certain set of problems. When the representation is extended to address a new problem feature, NP-completeness often requires additional algorithm engineering work. The inductive development process involves maintaining a set of problems representative of the neighborhood of success for the algorithm, and the methodical extension of this set as problematic cases are encountered. The key point of the inductive development process is that the improved algorithm will solve not only the new problematic cases, but also the entire set of representative problems. Mathematical programming based approaches can support the inductive development process through the use of implicit formulation techniques. In fact the above scheduling example was a problematic case that was brought into the solved set by mechanistically analyzing the cause of failure and adapting the solution approach.

4. Simulation-based optimization architecture

As discussed above, process management problems involve two issues that make their solution computationally difficult:

Uncertain data: Much of the data used to drive process management is highly uncertain, for example marketing forecasts, the true cost of lost sales, and equipment failures. As such management strategies must be designed to work under a range of possible conditions. Developing rational risk management policies for a process involves exploring the tradeoffs between the robustness of proposed plans to uncertain events and the costs to achieve this robustness. For example, determining an appropriate inventory level of a key intermediate involves trading off a carrying cost against unexpected customer demand for any downstream products or upstream production disturbances. The inventory carrying cost may be viewed as an insurance premium for protecting against particular types of risks. Understanding the protection afforded

Table 10
Final execution time statistics (in min) for 1300 2-week scheduling problems

SKU	Min.	Max.	Average	S.D.
5	1.983333	25.05	11.93962	5.038296
10	2.883333	57.78333	30.91132	14.40239
20	3.883333	25.4	12.28382	3.957422
40	4.516667	8.45	5.992521	0.939776
80	2.95	14.73333	9.098911	3.03401

Table 11
Solver failure rate prior to any algorithm engineering effort

SKU	Number of yearly runs made	Num of yearly runs not solved	Failure rate
5	14	7	0.5
10	10	0	0
20	12	6	0.5
40	5	1	0.2
80	7	0	0

by various levels of inventory helps keep costs to a minimum.

Combinatorial character: Even the most innocuous process management problems often have significant combinatorial character due to the need to manage resources over time. This combinatorial character arises from the need to address a series of yes/no questions whose answers, for example, specify product sourcing, the use of different contract manufacturing options, transportation modes, raw material purchase alternatives, marketing strategies to promote demand, etc.

The need to simultaneously address both the combinatorial aspect and the impact of uncertain data is a key technical challenge in approaching process management problems. While there exist deterministic methods that address the combinatorial aspect (see the previous sections) and simulation methods that use Monte Carlo sampling to investigate the impact of uncertainty, most existing methodologies do not effectively address the interaction of both to determine the best process management strategy. The approach described in this section is a computationally tractable approach for simultaneously considering combinatorial aspects and data uncertainty for industrial scale problems. The approach builds on well known simulation methods and deterministic methods for large-scale problems, such as those described in the previous section.

The SIMOPT architecture, as shown in Fig. 12, uses a customized mixed integer linear programming solver to optimize process behavior in conjunction with a discrete event simulator to investigate the effect of uncertainty on the plans output from the optimizer. The optimizer solution provides the initial input to the simulation and the simulation returns control to the optimizer whenever an uncertain event causes infeasibility that necessitates a new plan. Thus the iteration between the simulation and optimization continues until a timeline is traced out for a sufficiently long horizon of interest. The information generated along this timeline represents an example of how the future could unfold. As this process is repeated along different timelines for different simulation samplings (see Fig. 13), the evolution of behavior can be visualized in many possible realities. From a risk management perspective, the goal is to optimize the here-and-now decisions so that they perform well across a large fraction of the possible timelines. Of course the here-and-now optimization problem embodies the set of choices faced by planners and the SIMOPT approach allows them to gauge whether their decisions involve an acceptable amount of risk. One objective that can be used in this stochastic optimization is the lowest cost to achieve a certain level of risk. Risk is specified in terms of limiting the probability of certain goals not being achieved or

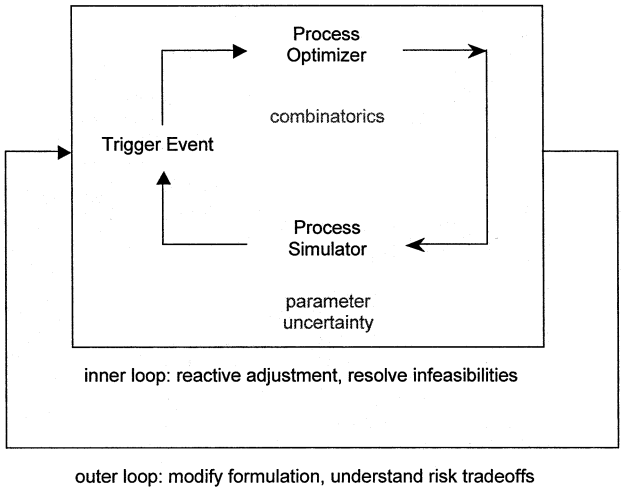

Fig. 12. Architecture of simulation-based optimization algorithm for combinatorial optimization problems under parametric uncertainty and for risk analysis.

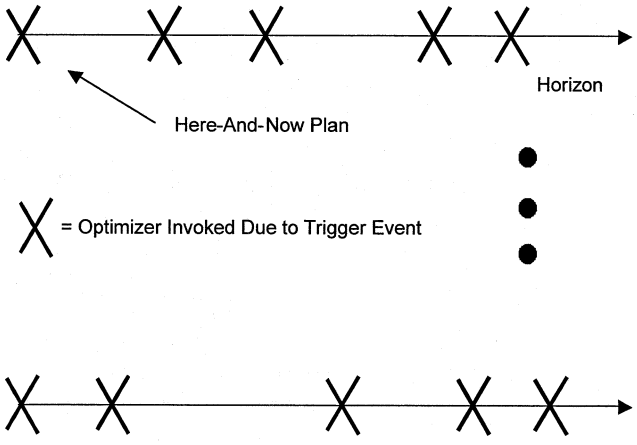

Fig. 13. Timelines traced by simulation-based optimization architecture.

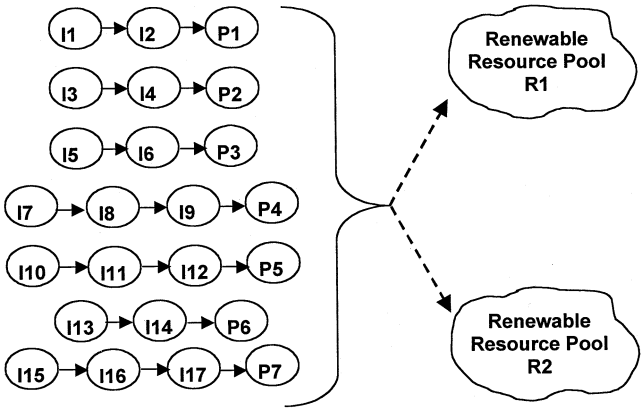

Fig. 14. Seven project example.

limiting the probability that a process will not achieve a certain level of performance.

A key challenge of the basic scheme shown in Fig. 12 is maximizing the rate at which meaningful samples can be collected so that the overall procedure completes in a reasonable time, even for realistic problem sizes. This challenge can be met with two basic strategies: (1) maximizing the speed at which the optimizer executes (as in the previous section), and (2) biasing the parameter sampling procedure so that the simulation focuses on 'critical events' and avoids a large number of simulations which are not particularly insightful. Strategy number (2) essentially requires using a procedure for identifying which uncertain events are problematic for a given plan and then biasing the sampling to explore them. A correction must be applied to accurately estimate the system behavior probabilities from the biased sampling (see Kalagnanam & Diwekar (1997), Diwekar & Kalagnanam (1997)).

A related challenge of the SIMOPT approach is determining when control should pass from the simulator to the optimizer. In particular, the response to an uncertain event can be to make local changes at a given time point in the simulator or to invoke the optimizer to effect more global changes. Obviously the more frequently the optimizer is invoked the slower the overall procedure, but the better the quality of the reaction. Practically the key to meeting this challenge is to determine an optimization frequency whereby if the optimizer is invoked more frequently the overall result does not change by an appreciable measure. The paper by Subramanian, Pekny and Reklaitis (2001) describes the issues involved with selecting reasonable trigger events and their associated computational expense.

4.1. Example: simulation-based optimization applied to an R & D pipeline

As an example of the SIMOPT approach consider management of a research and development pipeline where the goal is to maximize the rewards from successfully completing tasks over a 19 week horizon. This example is closely related to the scheduling example in the previous section in that the tasks in the pipeline must be scheduled so as not to utilize more resources than are dynamically available. As such a time bucketed scheduling formulation is used by the optimizer. In the pipeline there are the seven projects shown in Fig. 14 whose activity details are given in Table 12 and expected reward data are given in Table 13. Each activity has a duration, requirements for two different resources, and a probability of success. Each of these parameters is represented by a probability distribution as shown in Table 12. The duration and resource requirements are represented by a custom distribution that specifies a value and its associated probabilities. The activity success probabilities are represented by a triangular distribution. In practice the custom distributions could be specified simply as high, medium, and low values since estimating more detailed distributions is too tedious for many purposes. For this example there are 16 units of resource R1 and 8 units of resource R2 available. The distributions associated with each parameter are used by the simulation component to trace out timelines of possible pipeline behavior. The data given in Tables 12 and 13 are also used by the optimization component to schedule pipeline activities. An example schedule is given in Fig. 15 after the simulation component has sampled the task duration and resource requirement distributions. Note that the schedule shown in Fig. 15 only includes a subset of the projects because resource requirements limit the number of activities that may be simultaneously executed. In generating the schedule of Fig. 15, the optimization component was executed three times, to develop an initial schedule, to develop a schedule after activity I2 (project 1) failed, and after activity I6 (project 3) failed. Note that Fig. 15 only illustrates the initial portion of a possible timeline and that the simulation could be extended until all projects either fail or are executed. A more realistic simulation could also introduce new projects at later times. Each time the simulation is executed a new timeline is generated corresponding to one possible outcome of the pipeline. After many executions, a distribution of pipeline behavior is obtained. Fig. 16 shows the distribution of pipeline performance after 20 000 timelines are generated. Two distinct classes of performance are illustrated in Fig. 16. One class of performance is associated with timelines that experience an average reward of about $8000. Another class of performance is associated with timelines that experience an average reward of about $34 000. The more valuable class of performance is associated with the success of a blockbuster project. A close inspection of Fig. 16 shows that several timelines also exhibit behavior between the two dominant classes. Thus Fig. 16 illustrates that even a relatively small problem involving only a few projects exhibits complex behavior that arises because of uncer-

Table 12
Data for seven-project example

Activity	Duration (weeks), custom distribution		R1 (units), custom distribution		R2 (units), custom distribution		Probability success triangular distribution		
	Value	Probability	Value	Probability	Value	Probability	Min.	Most likely	Max.
I1	1	0.295	4	0.29	2	0.28	0.74	0.80	0.86
	2	0.375	5	0.44	3	0.44			
	3	0.190	6	0.21	4	0.28			
	4	0.110	7	0.06					
	5	0.030							
I2	2	0.10	4	0.06	1	0.22	0.7	0.75	0.8
	3	0.18	5	0.21	2	0.56			
	4	0.44	6	0.46	3	0.16			
	5	0.18	7	0.21	4	0.06			
	6	0.10	8	0.06					
P1	3	0.32	10	0.06	2	0.29	0.8	0.85	0.9
	4	0.40	11	0.12	3	0.44			
	5	0.18	12	0.55	4	0.21			
	6	0.10	13	0.21	5	0.06			
			14	0.06					
I3	1	0.335	3	0.05	1	0.23	0.7	0.8	0.85
	2	0.415	4	0.20	2	0.52			
	3	0.166	5	0.45	3	0.20			
	4	0.084	6	0.25	4	0.05			
			7	0.05					
I4	3	0.123	4	0.23	1	0.23	0.55	0.6	0.65
	4	0.203	5	0.52	2	0.55			
	5	0.335	6	0.20	3	0.16			
	6	0.228	7	0.05	4	0.06			
	7	0.111							
P2	4	0.32	10	0.23	2	0.23	0.75	0.8	0.85
	5	0.44	11	0.55	3	0.53			
	6	0.16	12	0.16	4	0.16			
	7	0.08	13	0.06	5	0.08			
I5	2	0.26	2	0.23	1	0.23	0.7	0.8	0.85
	3	0.54	3	0.55	2	0.53			
	4	0.14	4	0.15	3	0.17			
	5	0.06	5	0.07	4	0.07			
I6	3	0.10	4	0.23	1	0.225	0.7	0.75	0.8
	4	0.20	5	0.53	2	0.565			
	5	0.36	6	0.17	3	0.155			
	6	0.28	7	0.07	4	0.055			
	7	0.06							
P3	4	0.32	12	0.225	2	0.225	0.85	0.9	0.95
	5	0.40	13	0.555	3	0.555			
	6	0.18	14	0.160	4	0.160			
	7	0.10	15	0.060	5	0.060			
I7	3	0.335	3	0.23	2	0.23	0.85	0.9	0.95
	4	0.415	4	0.55	3	0.55			
	5	0.165	5	0.15	4	0.15			
	6	0.085	6	0.07	5	0.07			
I8	1	0.335	5	0.23	1	0.23	0.55	0.6	0.65
	2	0.415	6	0.57	2	0.57			
	3	0.165	7	0.15	3	0.15			
	4	0.085	8	0.05	4	0.05			
I9	1	0.32	4	0.23	1	0.22	0.65	0.7	0.75
	2	0.40	5	0.57	2	0.56			
	3	0.18	6	0.15	3	0.16			
	4	0.10	7	0.05	4	0.06			
P4	4	0.07	9	0.23	2	0.23	0.75	0.8	0.85
	5	0.17	10	0.53	3	0.53			
	6	0.52	11	0.17	4	0.17			
	7	0.17	12	0.07	5	0.07			
	8	0.07							

Table 12 (*Continued*)

Activity	Duration (weeks), custom distribution		R1 (units), custom distribution		R2 (units), custom distribution		Probability success triangular distribution		
	Value	Probability	Value	Probability	Value	Probability	Min.	Most likely	Max.
I10	3	0.33	4	0.23	3	0.23	0.7	0.85	0.85
	4	0.41	5	0.53	4	0.54			
	5	0.20	6	0.17	5	0.17			
	6	0.06	7	0.07	6	0.06			
I11	2	0.297	6	0.23	3	0.23	0.38	0.38	0.48
	3	0.456	7	0.54	4	0.53			
	4	0.197	8	0.17	5	0.17			
	5	0.05	9	0.06	6	0.07			
I12	1	0.32	5	0.23	1	0.23	0.38	0.38	0.48
	2	0.42	6	0.53	2	0.53			
	3	0.18	7	0.17	3	0.17			
	4	0.08	8	0.07	4	0.07			
P5	3	0.325	11	0.23	2	0.23	0.7	0.75	0.8
	4	0.375	12	0.53	3	0.53			
	5	0.150	13	0.17	4	0.17			
	6	0.100	14	0.07	5	0.07			
	7	0.050							
I13	3	0.305	4	0.23	3	0.23	0.7	0.75	0.8
	4	0.440	5	0.53	4	0.53			
	5	0.200	6	0.17	5	0.17			
	6	0.055	7	0.07	6	0.07			
I14	3	0.32	6	0.23	3	0.23	0.4	0.45	0.5
	4	0.42	7	0.53	4	0.53			
	5	0.18	8	0.17	5	0.17			
	6	0.08	9	0.07	6	0.07			
P6	3	0.088	13	0.20	3	0.22	0.6	0.65	0.7
	4	0.200	14	0.57	4	0.54			
	5	0.380	15	0.23	5	0.17			
	6	0.280			6	0.07			
	7	0.052							
I15	3	0.325	4	0.22	3	0.22	0.7	0.85	0.85
	4	0.425	5	0.54	4	0.54			
	5	0.175	6	0.17	5	0.17			
	6	0.075	7	0.07	6	0.07			
I16	1	0.283	6	0.22	5	0.22	0.45	0.5	0.55
	2	0.433	7	0.54	6	0.54			
	3	0.284	8	0.17	7	0.17			
			9	0.07	8	0.07			
I17	2	0.29	6	0.22	2	0.22	0.3	0.35	0.4
	3	0.44	7	0.54	3	0.54			
	4	0.21	8	0.17	4	0.17			
	5	0.06	9	0.07	5	0.07			
P7	3	0.29	13	0.20	3	0.22	0.45	0.5	0.55
	4	0.44	14	0.57	4	0.54			
	5	0.21	15	0.23	5	0.17			
	6	0.06			6	0.07			

tainty and resource contention. SIMOPT type approaches provide insight into this range of behavior, but there are several issues that must be addressed in practical applications. In particular, the nature of how rewards are computed has a significant effect on results. This example assumes that 10% of the reward is received after completion of the initial steps of the project and the remaining 90% of the reward is received after completion of the final step. Another strategy is to assume that all rewards are received only after the completion of the final step. Which assumptions are used for the rewards depends on the purposes of the study. If proper care is exercised in setting up the project parameters, this example shows that SIMOPT can be used to develop intuition about particular events that can occur (Fig. 15) and the classes of behavior that may be encountered in operating the pipeline (Fig. 16). The combined use of optimization and simulation provides a great deal of realistic insight because the schedules that are generated satisfy all the resource constra-

Table 13
Project reward data

Project	Reward $
P1	30 000
P2	20 000
P3	15 000
P4	40 000
P5	50 000
P6	40 000
P7	60 000

ints and the distribution information can be used to manage risk. More details about SIMOPT and this example may be found in Subramanian, Pekny and Reklaitis (2001).

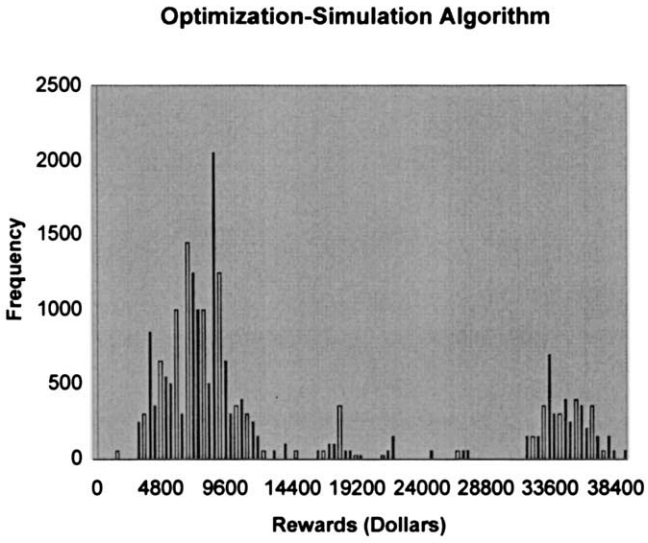

Fig. 16. Distribution of rewards for the seven-project example after 20 000 timelines are generated.

5. *e*PIMA: architecture for investigating game theoretic applications

In addition to combinatorial and uncertainty considerations, many practical problems involve multiple entities that must cooperate or compete in some environment. These types of game theoretic issues are most often found at the supply chain management level, but are important in any process management problem where significant behavior can only be modified indirectly. Traditionally, game theoretic issues are only addressed with relatively coarse approaches that neglect details to more fully focus on how cooperation or competition takes place. However, to accurately address many applications requires connecting game theoretic issues to process details. For example, effectively setting the proper pricing or timing of a consumer product promotion can depend on manufacturing operations (e.g. inventory levels, changeover costs, waste, etc.) and a competitor's manufacturing operations. Ideally, a promotion must be timed when inventory levels and manufacturing capacity can accommodate additional demand and puts a competitor at a disadvantage. Available to Promise (ATP) dates for potential product delivery are another example of where connecting game theoretic issues to process detail offers significant opportunity. The most effective ATP dates must take into account inventory levels, available process capacity, the importance of the customer, and current process operations. If the wrong ATP date is given this can result in a lost product sale or requiring extra cost to satisfy the order. Effective ATP dates can involve different product pricing levels or sharing demand forecasts between suppliers and customers so that delivery dates can be cooperatively determined. Examples of other process management questions where both detail and game theoretic issues are important include:

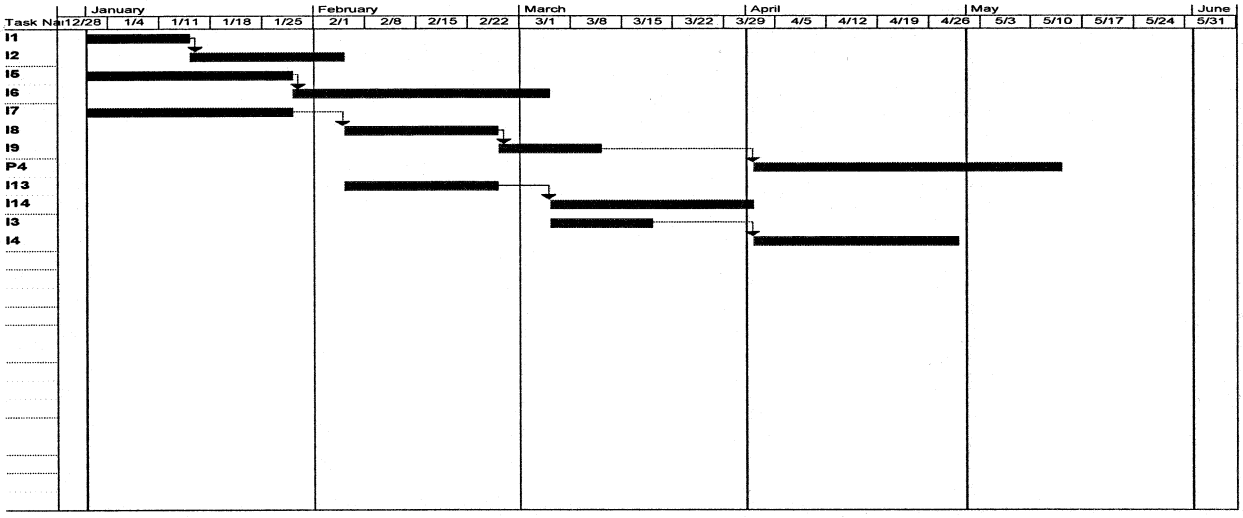

Fig. 15. A possible schedule for the seven-project example.

- What is the effect of forecast quality on performance?
- What is the effect of an increase in supplier or customer inventory levels on performance?
- What is the effect of re-distribution of inventory levels among the various entities in a supply chain?
- How much extra inventory is needed in the supply chain to counter the effect of forecasting error?
- What is the value of information sharing between the manufacturer and customers?
- What contracts, incentives, or penalties for bad forecasts should be negotiated with the customers to ensure better performance?

The SIMOPT architecture described in Section 4 represents a single locus of control. The premise is that the optimization model solution may be realized in practice because all the people responsible for its implementation have incentives to follow it. In a multiple entity environment, each entity has a separate locus of control and follows objectives specific to that entity. The *e*PIMA architecture shown in Fig. 17 incorporates one SIMOPT 'agent' per entity to model a multiple entity environment. Each SIMOPT agent is responsible for developing plans and simulating them to address uncertainty. Events that involve the interaction of more than one entity will appear on both entities simulation event stacks. The functionality associated with the 'Agent Mediation and Timeline Recording' box in Fig. 17 is responsible for informing all entities involved in an event once it is accepted by both agents (e.g. an order must be accepted by a supplier and placed by a customer). This functionality is also responsible for coordinating the event stack of each SIMOPT simulation and advancing the simulation time to correspond to the event on the stack with the next earliest event. Thus this functionality also has the information necessary to record a master time line for later analysis. As part of processing an event, a SIMOPT agent may invoke its optimization model to develop a new plan. Of course this can change the event stack of its simulation and the agent mediation function will inform other SIMOPT agents involved in the events so that their event stacks may be appropriately updated by any change in plans. More detail concerning the *e*PIMA architecture and its use to address the types of questions listed above is discussed in Bose (2001) and Subramanian (2000). Use of the *e*PIMA architecture is illustrated in the following example.

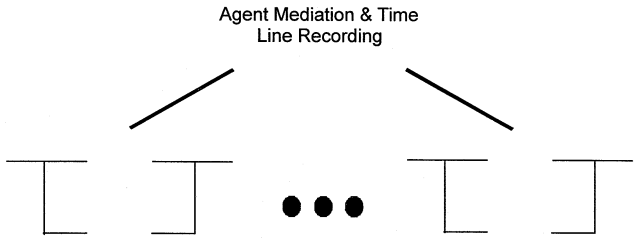

Fig. 17. The electronic process investigation and management analysis (*e*PIMa) environment for investigating game theoretic interactions under uncertainty.

5.1. Example: tradeoff between forecast quality, capacity, and customer service level in a manufacturer-retailer system

The *e*PIMA framework creates a virtual supply chain by modeling each of the entities and their interactions. The entities are represented as self-interested objects, working to control their business processes in an effort to optimize their own objectives. In this example a supplier, manufacturer, and retailer are modeled. This example is written from the perspective of the manufacturer. The other entities are modeled so as to evaluate the manufacturer's decisions under realistic conditions. While the manufacturer is modeled in the greatest detail, the other entities are modeled to a level of detail that is required to perform studies involving coordination and information sharing. The business processes modeled for the three-tiered supply chain under study are: forecasting and ordering model for the retailers, planning, scheduling and ordering model for the manufacturer, and planning-scheduling model for the supplier. The mathematical models for the business processes, in each of the entities, are given below. A demand module generates many different 'true-demand' scenarios for which experiments are performed with a separate *e*PIMA timeline associated with each experiment. Distorting the true-demand controls the level of forecasting accuracy. This introduces uncertainty in the system (like an external disturbance in the case of process control). The effect of this uncertainty and ways to combat the uncertainty can be evaluated. The optimization model of the manufacturer is implemented using a two-stage model: (i) a Set-point Setting (SPS) model whose output is fed to (ii) a Set-point attaining (SPA) model. The SPS model is solved over a longer time scale and hence cannot address all the necessary detail in the manufacturing facility. The costs parameters in the SPS model are updated based upon the solution of the SPA model, which is more detailed. Additional details may be found in Bose (2001).

The math-programming formulations for the SPS and SPA models are summarized next.

Economic model for the manufacturer (SPS model)

V_{ij}	inventory of variant i in period j
S_{ij}	inventory of SKU i in period j
SS_{ij}	safety stock of SKU i in period j
H	scheduling horizon
B_{ie}^{V}	batch size of production of variant i on equipment e

B^S_{ie} batch size of production of SKU i on equipment e
t^V_{ije} process time for batch of variant i in period j on equipment e
t^S_{ije} process time for batch of SKU i in period j on equipment e
n^V_{ije} number of batches of variant i produced in period j on equipment e
n^S_{ije} number of batches of SKU i produced in period j on equipment e
N^V number of variants
N^S number of SKUs
E^V set of equipments for producing variants
E^S set of equipments for producing SKUs
$\underline{M}^V$ matrix representing the bill of materials for stage I (purification of variants)
$\underline{M}^S$ matrix representing the bill of materials for stage II (packing of SKUs)
c cost of raw materials
P^V production costs for stage I (purification of variants)
P^S production costs for stage II (packing of SKUs)
P^M price mark-up for SKUs
h_{RM} holding cost of raw materials per unit weight per period
h_v holding cost of variants per unit weight per period
h_s holding cost of SKUs per unit weight per period
p_s penalty cost of missed demands for SKUs
S^+_{ij} dummy positive variable representing overage of SKU i in period j
S^-_{ij} dummy positive variable representing underage of SKU i in period j
SS_{ij} safety stock of SKU i in period j
D_{ij} demand of SKU i in period j
t^D_{je} downtimes and changeover-time on equipment e in period j
t^T total time available in a period

The SPS model seeks to maximize profit that is defined as revenue minus costs. The costs have three components, which are holding costs, penalty costs and production costs. The objective function is given below. For succinct matrix representation, the index i pertaining to the resource (SKU or variant) is dropped.

$$\text{Max (profit)} = \sum_{j=1}^{H} [(\text{revenue})_j - (\text{costs})_j] \quad (7)$$

where,

$$(\text{revenue})_j = (c\,\underline{M}^V\underline{M}^S + P^V\underline{M}^s + P^S)(D_j - S_j^-)(1 + P^M) \quad (8)$$

$$(\text{costs})_j = h_s S_j^+ + h_{RM} R_j + p_s S_j^- + P^V \sum_{e=1}^{E^V} n^V_{je} + P^S \sum_{e=1}^{E^S} n^S_{je} \quad (9)$$

The modeling of penalty and holding costs becomes easy by introducing two dummy independent variables, representing the underage (S^-_{ij}) and overage (S^+_{ij}) of inventory. The actual inventory in any week for a SKU is given by the following equation.

$$S_{ij} = S^+_{ij} - S^-_{ij} \quad (10)$$

The variables S^+_{ij} and S^-_{ij} cannot take negative values and both S^+_{ij} and S^-_{ij} cannot be positive at the same time.

The above objective function is subject to various constraints. The following sets of equations are the mass balance constraints, which link the inventory from one period to the next.

$$S_{ij} = S_{i(j-1)} + \sum_{e \in E^S} [n^S_{ije} B^S_{ie}] - D_{ij} \quad \forall i \in [1,N_S],\ \forall j \in [1,H] \quad (11)$$

$$V_{ij} = V_{i(j-1)} + \sum_{e \in E^V} [n^V_{ije} B^V_{ie}] - \sum_{e \in E^S} \{\underline{M}^S_{ie}[n^S_{ije} B^S_{ie}]\} \quad \forall i \in [1,N_S],\ \forall j \in [1,H] \quad (12)$$

The constraint determines the raw material inventory that should be available at the beginning of each period.

$$R_j \geq \sum_{e \in E^V} [\underline{M}^V (n^V_{je} B^V_e)] \quad \forall j \in [1,H] \quad (13)$$

The next two equations enforce the condition that the total production time, downtime and changeover-times do not exceed the total available time in a period.

$$\sum_{i=1}^{N_S} [t^S_{ije} n^S_{ije}] + t^D_{je} \leq t^T \quad \forall j \in [1,H],\ \forall e \in E_S \quad (14)$$

$$\sum_{i=1}^{N_V} [t^V_{ije} n^V_{ije}] + t^D_{je} \leq t^T \quad \forall j \in [1,H],\ \forall e \in E_V \quad (15)$$

The lower and upper bounds for the independent variables are given below.

$$0 \leq S^+_{ij} \leq S^+_{\max} \quad \forall i \in [1,N_S],\ \forall j \in [1,H] \quad (16)$$

$$0 \leq S^-_{ij} \leq S^-_{\max} \quad \forall i \in [1,N_S],\ \forall j \in [1,H] \quad (17)$$

$$0 \leq V_{ij} \leq V_{\max} \quad \forall i \in [1,N_V],\ \forall j \in [1,H] \quad (18)$$

$$0 \leq R_{ij} \leq R_{\max} \quad \forall i \in [1,N_V],\ \forall j \in [1,H] \quad (19)$$

$$y^S_{ije} \in \{0,1\} \quad \forall i \in [1,N_S], \forall j \in [1,H],\ \forall e \in E_S \quad (20)$$

$$y^V_{ije} \in \{0,1\} \quad \forall i \in [1,N_V],\ \forall j \in [1,H],\ \forall e \in E_V \quad (21)$$

5.1.1. Scheduling model for manufacturer (first SPA model)

The manufacturer uses the SPA (scheduling) model to attain the target inventory levels set by the SPS

model. In order to avoid being myopic the manufacturer plans for several periods ahead in advance in the SPS model. However, the SPA model is only invoked for the first period in order to obtain a detailed schedule for that period. The SPA model is defined and solved with the VirtECS® scheduler (Advanced Process Combinatorics, Inc., 2001). Before the SPA model is formulated, the solution of the SPS model is processed to compute the demands for the scheduling problem. Based on the solution of the SPA model, the product costs in the SPS model are updated and the SPS model is resolved. This updating is done to reflect the actual values of the products cost parameters.

5.1.2. Ordering model for the manufacturer (second SPA model)

The model is based upon *dynamic lot-sizing* where demands are known but vary over time. This single item model is solved for each SKU.

Let, H is the planning horizon (the number of periods for which demands are known in advance), D_j is the forecast demand in period j, $j=[1, H]$, c_j is the unit purchase cost in period j, $j=[1, H]$, K is the fixed (set-up) cost incurred when an order is placed in period j, $j=[1, H]$, h is the holding cost incurred to carry an unit of inventory from period j to $(j+1)$, $j=[1, H]$, I_j is the amount of product inventory at the start of a period t (before ordering), $j=[1, H]$, and Q_j is the order quantity for period t (to be determined), $j=[1, H]$.

The formulation seeks to determine Q_j in order to minimize the total cost (holding cost plus fixed-ordering cost) while satisfying all demands (no back orders) (see Bose, 2001 for details). It is assumed that the purchase cost of items is constant over time. Thus $c_j=c$ for all $j=[1, H]$.

The objective function tries to minimize the sum of holding costs and ordering costs. Since it is assumed that the ordering models will order so as to fulfill all demands for raw materials, the penalty cost for missing any demand is not considered in the model. The objective function is given as follows:

$$\text{Min} \sum_{j=1}^{H} [hI_j] + \sum_{j=1}^{H} [K\delta(Q_j)] \tag{22}$$

Here the delta function (δ) takes the value 1 if Q_j is positive, 0 otherwise. The above objective function is subject to several constraints. The first constraint is a material balance constraint.

$$I_j = I_{j-1} + Q_j - D_j \quad \forall j \in [1,H] \tag{23}$$

The upper and lower bounds for the variables are specified in the problem. They are given as:

$$0 \le I_j \le I_{\max} \quad \forall j \in [1,H] \tag{24}$$

$$0 \le Q_j \le Q_{\max} \quad \forall j \in [1,H] \tag{25}$$

Let the lead-time between the manufacturer and the retailers is L periods. Thus, quantity, Q_j ordered in period j will be received in period $j+L$. Eq. (23) changes to:

$$I_{j+L} = I_{j+L-1} + Q_j - D_{j+L} \quad \forall j \in [1,H] \tag{26}$$

We provide below a brief description of the other models used in this example.

5.1.3. Planning model for the supplier

In order to produce the variants from the raw material, the supplier uses a planning model. This planning model is simpler since there are no parallel equipments for producing the bases and variants. The formulation is similar to the planning model used for the manufacturer. We assume that the raw materials for the supplier are in abundance and hence his forecasting and ordering processes are not considered.

5.1.4. Demand generation model

The customer demands for each of the SKUs at the retailers is generated using a sum of different demand signals. These signals are base demand, growth or decay with time, cyclic demands, seasonal demands and random noise. The parameters that control each of the demand signals are obtained from the historical demand data of the industrial case study. The many instances that can be generated using these parameters are representative of the actual demands that can occur in this industry. The demand, $D_i(t)$, for a period t and SKU i is given as follows:

$$D_i(t) = B_i + G_i t + C_i \sin(\hat{a}_i t) + S_i \cos(\hat{a}_i t) + \text{Noise}_i \tag{27}$$

Here, B_i is the base demand of SKU i, G_i is the growth or decay of demand of SKU i, C_i is the coefficient for cyclic demand with periodicity $\tilde{a}_i$, S_i is the coefficient for seasonal demand with periodicity $\tilde{a}_i$, Noise_i is the normally distributed noise, $\sim N(0, \sigma_i)$, $\sigma_i = 5\%$ of B_i.

5.1.5. Forecasting model for retailers

The demand observed by the retailers in the supply chain is stochastic. Hence, the retailers can predict the demands for the future weeks to a certain degree of accuracy. We will define this degree of accuracy as the *forecast quality*. This accuracy is measured as the deviation of the forecasted demands from the true demands. The deviation is normalized between 0 and 1, 0 being the ability to predict future demands perfectly, and 1 denoting poor prediction of the future demands.

The forecast, $F_i(t)$, for a period t and SKU i is calculated as follows:

$$F_i(t) = [D_i(t)]^* N_i^F \tag{28}$$

Here, N_i^F is the norm ally distributed noise, $\sim N(1, \sigma_i)$, σ_i is a parameter that can be varied to obtain forecasts of different accuracy.

Table 14
Forecast quality description

Forecast quality	σ_i (%)
Good	5
Average	20
Bad	50

Table 14 lists the values of σ_i, which were used to obtain the various forecast qualities of 'good', 'average' and 'bad'. The retailer orders SKUs based on the forecasts using a similar ordering model as described for the manufacturer.

This example represents the situation when the manufacturer can make an investment towards improving the forecast quality and/or increase the production capacity. Under such conditions, the manufacturer must choose the best investment decision, which is one of following three options.

1. (Forecast Improvement) Invest in better information-systems, like Electronic Data Interchange (EDI), or Point of Sales (POS) data, or marketing and product sampling surveys.
2. (Capacity Expansion) Invest in increasing plant capacity, or purchasing technologically improved and efficient equipment.
3. A combination of the above investments.

In such situations, the *e*PIMA architecture can reveal which of the options will yield the maximum improvement in the service level. Results are shown in Fig. 18 as a contour plot of service level against forecast quality and increase in plant-capacity. Clearly, high quality forecasting can allow the same level of customer service to be reached with less process capacity. Another of the important observations is that if the manufacturer has 'bad' forecast quality, then an increase in the plant capacity will not be effective. The results argue for a combination of both options. For example, in order to increase service level to 95%, the manufacturer can improve the forecast quality to an 'average' level from a 'bad' level and then increase the capacity by 10%. To further increase the service level, to say 97%, from 95% the manufacturer can either improve the forecast quality to a 'good' level, or increase the capacity to 20%. At different points on the contour plot, the marginal increase in service level varies depending upon what investment option is chosen at that point.

6. Conclusions

This paper has discussed three algorithm architectures for addressing decision problems arising in process management applications: (i) mathematical programming architecture for problems involving large-scale combinatorics, (ii) SIMOPT architecture for problems involving combinatorics, uncertainty, and risk

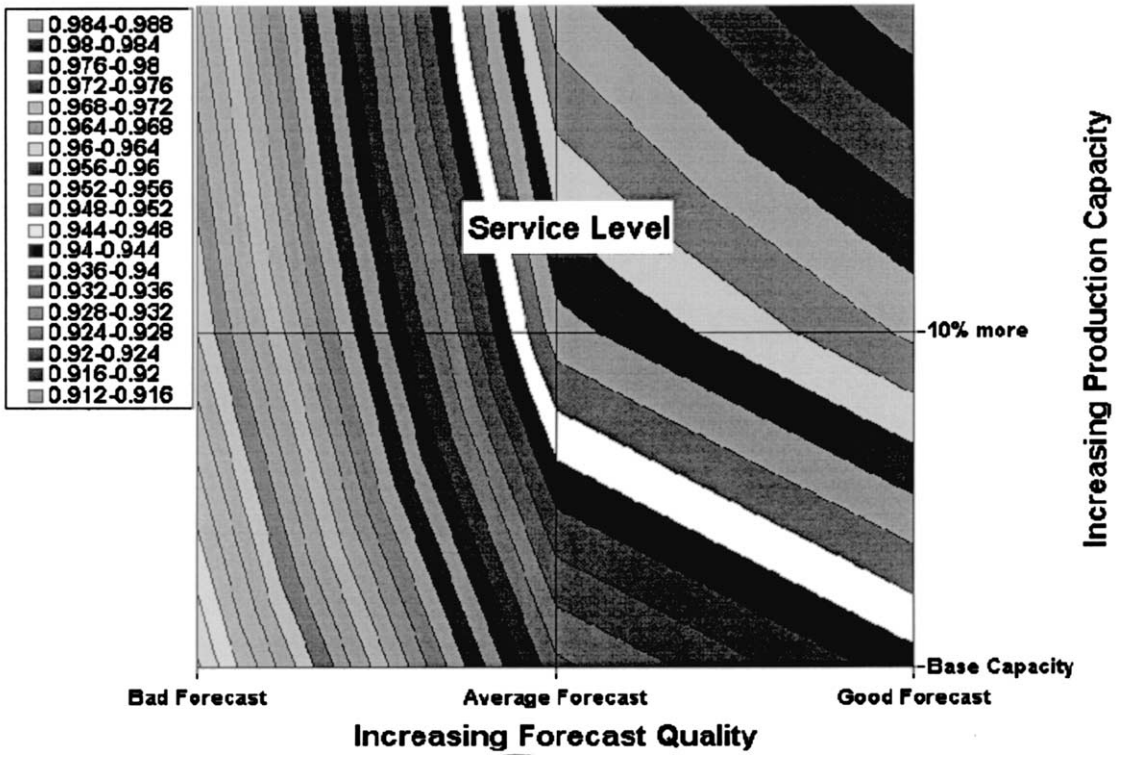

Fig. 18. Contour plot of service level.

management, and (iii) an *e*PIMA architecture for problems involving game theoretic issues. A key factor underlying the design and engineering of these architectures is the uncertainty principle implied by NP-completeness. Since no single algorithm can guarantee both the solution quality and a reasonable execution time over the range of all problems of interest, consideration of life-cycle issues is of paramount importance. In particular the delivery, control, support, and extension of a given architecture and implementation are critical research and development issues and ones that will control the sophistication with which practical problems can be addressed. Mathematical programming based architectures offer several advantages with regard to life-cycle issues, although the intellectual hurdles are high since the size of practical problems dictates sophisticated techniques to achieve acceptable performance (e.g. implicit generation of formulations). For several application areas, this sophistication is justified since the ready availability of information is putting a premium on the ability to quickly reduce information to effective actions, identify key information driving an answer, and develop strategies that manage risk. In this regard the formalism of mathematical programming offers the potential for cheaper delivery with regard to the effort required for an application, detailed algorithm control, rapid support in the event of problematic problems, and extensibility to new constraint combinations and application areas.

References

Advanced Process Combinatorics, Inc., (2001). 3000 Kent Avenue, West Lafayette, IN, 47906, www.combination.com.

Applegate, D., Bixby, R., Chvátal, V., Cook, W. (1998). On the solution of traveling salesman problems, *Documenta Mathematica Journal DMV Extra Volume ICM III*, 645–656.

Applequist, G., Samikoglu, O., Pekny, J., & Reklaitis, G. (1997). Issues in the use, design, and evolution of process scheduling and planning systems. *ISA Transactions*, *36*(2), 81–121.

Bodington, C. E. (1995). *Planning, scheduling, and control integration in the process industries*. New York: McGraw Hill.

Bose, S. (2001). PhD thesis dissertation. Purdue University, West Lafayette, IN.

Bunch, P. (1997). PhD thesis dissertation. Purdue University, West Lafayette, IN.

Diwekar, U. M., & Kalagnanam, J. R. (1997). Efficient sampling techniques for optimization under uncertainty. *AIChE Journal*, *43*(2), 440–447.

Elkamel, A. (1993). PhD thesis dissertation. Purdue University, West Lafayette, IN.

Elkamel, A., Zentner, M., Pekny, J. F., & Reklaitis, G. V. (1997). A decomposition heuristic for scheduling the general batch chemical plant. *Engineering Optimization*, *28*, 299–330.

Epperly, T. G. W., Ierapetritou, M. G., & Pistikopoulos, E. N. (1997). On the global and efficient solution of stochastic batch plant design problems. *Computers and Chemical Engineering*, *21*, 1411–1431.

Flaherty, T. (1986). *Applichem Case Study*. Harvard Business School.

Garey, M.R., Johnson, O.S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness, New York: Freeman.

Greenberg, H. (1998). An annotated bibliography for post-solution analysis in mixed integer programming and combinatorial optimization. In D. L. Woodruff, *Advances in computational and stochastic optimization, logic programming, and heuristic search* (pp. 97–148). Boston, MA: Kluwer Academic.

Grossmann, I. E., & Westerberg, A. W. (2000). Research challenges in process systems engineering. *AIChE Journal*, *46*(9), 1700–1703.

Honkomp, S. J. (1998). PhD thesis dissertation. Purdue University, West Lafayette, IN.

Kalagnanam, J. R., & Diwekar, U. M. (1997). An efficient sampling technique for off-line quality control. *Technometrics*, *39*(3), 308–319.

Kondili, E., Pantelides, C. C., & Sargent, W. H. (1993). A general algorithm for short-term scheduling of batch operations-I. MILP Formulation. *Computers and Chemical Engineering*, *17*(2), 211–227.

Kudva, G. K., & Pekny, J. F. (1993). A distributed exact algorithm for the multiple resource constrained sequencing problem. *Annals of Operations Research*, *42*, 25–54.

Magnusson, P. (1997). *The Internet Revolution History and Significance*, http://www.sics.se/ ~ psm/ar97/, Swedish Institute of Computer Science.

Miller, D. L., & Pekny, J. F. (1991). Exact solution of large asymmetric traveling salesman problems. *Science*, *251*, 754–761.

Miller, D. L., & Pekny, J. F. (1995). A staged primal-dual algorithm for perfect b-matching with edge capacities. *ORSA Journal on Computing*, *7*(3), 298–320.

Moscato, P. (2002). *TSPBIB Home Page*, http://www.densis.fee.unicamp.br/ ~ moscato/TSPBIB_home.html.

Padberg, M. W., & Rao, M. R. (1982). Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, *7*(1), 67–80.

Pantelides, C. C. (1993). Unified frameworks for optimal process planning and scheduling. In Rippin, D. W. T., Hale, J. (Eds.), *Proceedings 2nd Conference on Foundations of Computer-Aided Operations*, CACHE Publications.

Parker, R. G., & Rardin, R. L. (1988). *Discrete optimization*. New York: Academic Press.

Pekny, J. F., Reklaitis, G. V. (1998). Towards the convergence of theory and practice: a technology guide for scheduling/planning methodology. *Foundations of Computer Aided Process Operations Conference Proceedings* (91–111).

Pekny, J. F., & Miller, D. L. (1991). Exact solution of the no-wait flowshop scheduling problem with a comparison to heuristic methods. *Computers and Chemical Engineering*, *15*(11), 741–748.

Pekny, J. F., Zentner, M. (1994). Learning to solve process scheduling problems: the role of rigorous knowledge acquisition frameworks. *Foundations of Computer Aided Process Operations Conference Proceedings* (275–310).

Phelps, K. (2002). *You Think This is a Revolution You ain't Seen Nothing Yet*, http://www.xanadu.com.au/archive/revolution.html.

Qin, S., Badgwell, T.A., (2002). An overview of industrial model predictive control technology. http://www.che.utexas.edu/ ~ qin/cpcv/cpcv14.html.

Reklaitis, G. V., Pekny, J., & Joglekar, G. S. (1997). *Scheduling and simulation of batch processes, handbook of batch process design* (pp. 24–60). London: Blackie Academic & Professional, an imprint of Chapman & Hill.

Rohrer, M. (2000). *Simulation Success*. Warehousing Management, August.

Shobrys, D. E., White, D. C. (2002). Planning, scheduling, and control systems: why can't they work together. *Computers and Chemical Engineering*, 26(2), in press.

Subramanian, V. (2000). MS thesis dissertation. Purdue University, West Lafayette, IN.

Subramanian, V. (2001). Expansion of the harvard applichem case study to account for engineering detail. *CIPAC Report*, *School of Chemical Engineering*. Purdue University, West Lafayette, IN.

Subrahmanyam, S., Bassett, M. H., Pekny, J. F., & Reklaitis, G. V. (1995). Issues in solving large scale planning, design and scheduling problems in batch chemical plants. *Computers and Chemical Engineering*, *19*, s577–s582 (Suppl.).

Subrahmanyam, S., Pekny, J. F., & Reklaitis, G. V. (1996). Decomposition approaches to batch plant design and planning. *Industrial and Engineering Chemistry Research*, *35*(6), 1866–1876.

Subramanian, D., Pekny, J. F., Reklaitis, G.V. (2001). A simulation-optimization framework for research and development pipeline management. AICHE Journal, 47(10), 2226–2242.

Thomas, L. R., & Shobrys, D. E. (1988). Planning and scheduling lube blending and packaging. *Hydrocarbon Processing*, 111.

Trick, M. A. (2002). *A Tutorial on Integer Programming*, http://mat.gsia.cmu.edu/orclass/integer/node1.html.

Tsay, A. (1999). The quantity flexibility contract and supplier–customer incentives. *Management Science*, *45*(10), 1339–1358.

Zentner, M. G., Pekny, J. F., Reklaitis, G. V., & Gupta, J. N. D. (1994). Practical considerations in using model based optimization for the scheduling and planning of batch/semicontinuous processes. *Journal of Process Control*, *4*(4), 259–280.

Zentner, M., Elkamel, A., Pekny, J., & Reklaitis, G. V. (1998). A language for describing process scheduling problems. *Computers and Chemical Engineering*, *22*(1–2), 125–145.