An Optimization-Based Approach for Design Project Scheduling

Ming Ni, Peter B. Luh, *Fellow, IEEE*, and Bryan Moser

***Abstract*—Concurrent engineering has been widely used in managing design projects to speed up the design process by concurrently performing multiple tasks. Since the progress of a design task often depends on the knowledge about other tasks and requires effective communication, tasks and communication activities need to be properly coordinated to avoid delays caused by waiting for information or the need for rework. This paper presents a novel formulation for design project scheduling with explicit modeling of task dependencies and the associated communication activities. General dependencies are modeled as combinations of three basic types representing sequential, concurrent, and independent processes. Communication activities are also modeled as tasks, and their interactions with design tasks are described by sets of intertask constraints. The objective is to achieve timely project completion with limited resources. To improve algorithm convergence and schedule quality, penalties on the violation of constraints coupling design tasks are added to the objective function. A solution methodology that combines Lagrangian relaxation, dynamic programming, and heuristic is developed to schedule design and communication tasks, and a surrogate optimization framework is used to overcome the "inseperability" caused by nonadditive penalties. A heuristic procedure is then developed to obtain scheduling policies from optimization results and to dynamically construct schedules. Numerical results show that the approach is effective to handle various task dependencies and the associated communication activities to provide high-quality schedules.**

***Note to Practitioners*—A short product design cycle is critical for manufacturers to succeed in the era of time-based competition. To speed up the design process, concurrent engineering has been widely used where multiple design tasks are concurrently performed. As a design task can be performed with preliminary or partial information on other tasks and the information may be updated, effective communication is required. If design and communication activities are not properly coordinated, the project may be significantly delayed by waiting for information or by the need for rework. This paper presents a novel formulation for design project scheduling by appropriately modeling task dependencies and the associated communication activities. A solution methodology based on decomposition and coordination is developed to schedule both design and communication tasks to achieve timely project completion. Numerical results show that the approach is effective to handle various task dependencies and the associated communication activities to provide high-quality schedules.**

Manuscript received May 16, 2006; revised January 7, 2007. This paper was recommended for publication by Associate Editor Y. Narahari and Editor N. Viswanadham upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under Grant DMI-9813176 and Grant DMI-0223443, and by a grant from Global Project Design.

M. Ni and P. B. Luh are with the Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT 06269-2157 USA (e-mail: mingni@engr.uconn.edu; Peter.Luh@uconn.edu).

B. Moser is with Global Project Design, Edgewood, KY 41017 USA (e-mail: bryan@gpdesign.com).

Digital Object Identifier 10.1109/TASE.2008.916928

***Index Terms*—Concurrent engineering, design project scheduling, Lagrangian relaxation, surrogate optimization, task dependency and communication activities.**

I. Introduction

A. Motivation

A SHORT product design cycle is critical for manufacturers to succeed in the era of time-based competition. To speed up the design process, concurrent engineering has been widely used where design tasks within a project are concurrently performed. These tasks are assigned to design teams based on team capabilities, and the progress of a task often depends on the knowledge about other tasks such as design concepts, component configurations, ranges of major parameters, and other key technical data. Unlike traditional sequential design where a task is started by a team with full knowledge on upstream tasks, a task under the concurrent engineering paradigm may be started with preliminary or partial information. In view that preliminary designs are often modified, new information is frequently generated by upstream teams, and changes have to be incorporated in downstream tasks [12]. These updates may demand intensive communication, such as phone calls, teleconferences, and face-to-face meetings, etc. [18]. A study showed that communication time may constitute 75% of total project time according to Christian [9].

In view of the above, if design tasks and communication activities are not appropriately scheduled, a downstream team might start before enough information has been generated by upstream teams, or the information has not been transferred to downstream teams in a timely manner. As a result, tasks may be significantly delayed since design teams may have to wait for information in order to proceed further, and the delay of a single task may have a domino effect on subsequent tasks linked through dependency relationships or through sharing of common resources. Also, performing a task with preliminary or partial information is inherently risky for a downstream team. If a downstream task starts too early or proceeds too fast with insufficient information, time-consuming rework may have to be carried out to incorporate changes in upstream designs, causing significant delays. To achieve timely project completion without unnecessary delays, an appropriate approach is imperative to model task dependencies to indicate how early a task can start and how fast it can proceed, and to effectively schedule both design and communication activities. Additionally, design and communication times could be uncertain in view of the creative nature of design, and the approach should be able to appropriately handle uncertainties and to dynamically construct schedules.

1545-5955/$25.00 © 2008 IEEE

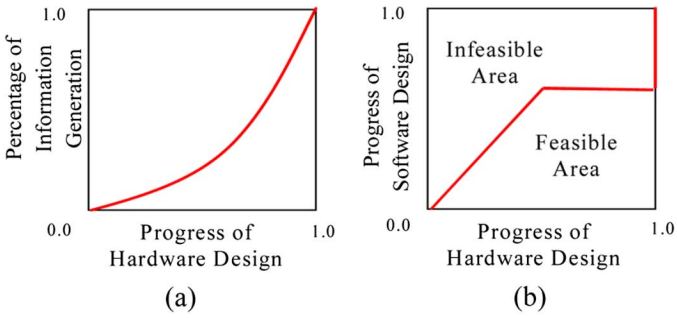

Fig. 1. Task dependency. (a) Information Generation. (b) Task Progress Relationship.[2]

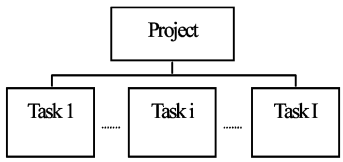

Fig. 2. An example of project structure.

B. Literature Review

In the following, existing models on task dependencies are first introduced. Studies of the associated communication activities are then reviewed. Finally, existing project scheduling methods are discussed.

Task Dependency: To analyze the dependency relationships among tasks in a project, it is important to delineate for each task which other tasks information is needed from and which other tasks information will be provided to as described by the Dependency Structure Matrix (DSM) [29]. This matrix has been used to identify necessary design iterations, and to find appropriate task sequences to reduce possible rework [8], [11], [19]. Since the DSM only captures zero-one task dependency relationships, generalized precedence relations were developed to include start–start, start–finish, finish–start, and finish–finish relationships [7], [27]. For example, the relationship between two concurrently performed tasks can be described by start–start and finish–finish relationships, i.e., assuming that information is generated throughout the upstream task, the downstream task can start after the upstream task is started, and finish after the upstream task is finished. This method has been used in project scheduling [27]. However, generalized precedence relations cannot describe "how fast a task can be performed," since concurrency within the design process cannot be captured by task start and finish times.

To clearly model task dependency relationships under the concurrent engineering paradigm, the information generation process was modeled and task dependencies were specified in terms of task progresses (percentage of completion) instead of start and finish times in [9]. Consider for example the design of a computer data acquisition system, as shown in Fig. 1, where the software design task requires the information from hardware design. Fig. 1(a) describes the percentage of information generated in the process of hardware design. Fig. 1(b) represents the maximum progress of software design for a given progress of hardware design, and indicates how fast the software design can be performed.[1] This curve specifies the feasible area for software design. By keeping software design in the feasible area (including the curve itself), the chance of rework is assumed small since sufficient information on hardware design can be obtained. Such a progress curve is assumed to be monotonically nondecreasing. In [9], communication activities are modeled as part of the design work, and their required person-hours are given. The models were used to study the effects of task person-hours variations, to determine design team sizes, and to predict project duration by using simulations. Relationships among design and communication activities, however, have not been analyzed, and when communication activities should be performed has not been studied. In addition, the mathematical formulation of task dependencies for the schedule optimization purpose was beyond the scope of [9].

Communication: In most results on task dependencies, communication has not been explicitly modeled. A method to predict communication flows was presented by Morelli *et al.* [23], where Dependency Structure Matrices were used to identify task dependencies that may lead to communication requirements. Statistical models were used to study how factors such as the degrees of task dependencies affect communication by Sosa *et al.* [28]. No communication models have been presented in the literature for the schedule optimization purpose to the best of our knowledge.

Project Scheduling: With respect to project scheduling, studies established that it is a generalization of job-shop scheduling problems and as such is NP-hard [5]. To solve the problems, optimization-based methods such as branch-and-bound were developed by Demeulemeester and Herroelen [10], Nazareth *et al.* [25], Zamani [34], and Heilmann [15]. For most results, a total of less than 50 tasks were scheduled on about five types of resources due to problem complexities. Other optimization methods including genetic algorithms were developed by Hartmann [13], Ozdamar [26], Hartmann [14], and Valls and Ballestin [30]. Problems of larger sizes with more than 100 activities were solved, however, solution qualities were hard to evaluate. In addition to optimization, heuristic procedures were developed by Ahn and Erenguc [2], Narahari *et al.* [24], Yan and Wu [33], and Merkle *et al.* [22]. These procedures are computationally efficient, however, results are often of questionable quality, and there is no good way to systematically improve the quality. Techniques to generate benchmark instances were developed by Kolisch *et al.* [17].

The above approaches are based on deterministic models. A review of stochastic methods with uncertain activity durations was provided by Brucker *et al.* [6], covering methods such as stochastic dynamic programming and heuristic procedures. An optimization method combining Lagrangian relaxation and stochastic dynamic programming was developed to handle uncertain process times [20], and uncertain number of design iterations [21]. In addition, product development projects were modeled as stochastic processing networks [1], and factors that affect product development time were analyzed.

[1]Comparing with [9, Fig. 2.3], x–y axes are swapped in this paper to be consistent with the figure of information generation process.

[2]The name of this type of figures is "Completion" in [9].

C. Scope of This Paper

In this paper, a novel optimization formulation and the corresponding solution methodology for design project scheduling are presented. In Section II, communication activities are explicitly modeled as tasks to send and receive information, and the method by Christian [9] is extended to describe dependency relationships among various tasks for the schedule optimization purpose. General task dependencies are modeled as combinations of three basic types representing sequential, concurrent, and independent processes, and are described by linear inequality constraints. A design project scheduling problem is then formulated based on the scheduling model by Zhang *et al.* [36]. The objective is to achieve timely project completion subject to task dependencies and resource capacity constraints. To improve algorithm convergence and schedule quality, penalty terms for the violation of coupling communication dependency constraints and resource capacity constraints are added to the objective function. As our focus is on modeling task dependencies, uncertainty is not introduced and the formulation is deterministic.

In the formulation, constraints, and the original objective function are additive in terms of design tasks, and only the penalty terms are nonadditive, resulting in a "pseudoseparable" structure. A solution methodology based on decomposition and coordination using Lagrangian relaxation is developed in Section III, and a surrogate optimization framework is used to overcome the inseparability. After coupling constraints are relaxed by using Lagrangian multipliers, all terms associated with a particular design task and its associated communication tasks are pulled out from the Lagrangian to form a task subproblem. By keeping decision variables belonging to other subproblems at their latest available values, subproblems are solved by using dynamic programming to obtain solutions satisfying the "surrogate optimization condition" (to be described later) for proper multiplier updating directions. Subproblem solutions are then coordinated through iterative updating of multipliers by using the surrogate subgradient method by Zhao *et al.* [37]. A heuristic procedure is also developed to construct feasible schedules based on subproblem solutions. To handle possible uncertainties in task processing times, solutions obtained based on the deterministic model of Section II are used to establish scheduling policies to dynamically construct schedules based on the realizations of random events.

Numerical results presented in Section IV demonstrate that the approach is effective to handle task dependencies to provide high-quality schedules. The importance of appropriately scheduling communication activities is demonstrated by the significantly better performance of our approach as compared with the case where communication activities are not initially considered, but are performed during the execution phase when teams need information to proceed further. Our approach is also shown to be effective to handle uncertain task processing times.

II. Problem Formulation

In this section, the structure of a design project is first presented, and communication activities are explicitly modeled as tasks to send and receive information. The graphic method in [9] is extended to describe dependency relationships among various tasks. For simplicity, it is assumed that there is no rework by keeping progresses of tasks within their feasible areas, e.g., the one in Fig. 1(b). General task dependencies are represented by combining three basic types, i.e., precedence, pace, and independency, respectively, representing sequential, concurrent, and independent processes, and are described by linear inequality constraints. The project scheduling problem is then formulated based on the scheduling model of Zhang *et al.* [36]. Since our focus is on the modeling and treatment of task dependencies, the formulation is deterministic. Nevertheless, when task processing times are uncertain, solutions obtained based on the deterministic model can be used to establish scheduling policies to dynamically construct schedules based on the realizations of random events.

A. Design Project Structure

Suppose that the scheduling horizon has K discrete time units, with index k ranging from 0 to K-1. There are H teams working on the project. For simplicity, it is assumed that each team has a finite number of designers with identical capabilities, and is modeled as a single "resource type" from the scheduling viewpoint. The capacity (or the number of designers) of resource type h $(1 \leq h \leq H)$ at time k is assumed given and is denoted as M_{kh}. The project under consideration has I design tasks based on the overall product structure, e.g., one task for designing one module of the product, as shown in Fig. 2. The ith design task may be assigned to a team $h_i (1 \leq h_i \leq H)$ with a given resource-hour R_{ih}, which is assumed to be deterministic.

To describe task dependencies, suppose that design task i is dependent on the knowledge of design task j. Task j is an "upstream design task" for providing the information, and i is a "downstream design task" for requiring the information. The information generated in the progress of j needs to be sent by h_j to h_i. For simplicity, the activities of h_j to organize and send information are aggregated as a single "sending task" belonging to j and denoted by $s_i(j)$ and require certain resource-hours to perform. Similarly, the activities of h_i to receive and digest the information are aggregated as a single "receiving task" belonging to i and denoted by $r_j\,(i)$. The relationships among the four tasks are shown in Fig. 3. In addition to obtaining information from h_j, team h_i may need to provide feedback information to h_j. In this case, tasks i and j are interdependent. Accordingly, there is a sending task for h_i to send the feedback and a receiving task for h_j to receive the feedback.

B. Dependencies Among Various Tasks

With the introduction of sending and receiving tasks, relationships among design and communication tasks need to be established as part of the overall task dependencies for the scheduling purpose. These relationships include dependencies between upstream design and sending, sending and receiving, and receiving and downstream design tasks; and each is represented by a graphical method similar to that of Christian [9], as illustrated in Fig. 1. In this way, how early and how fast tasks can be performed are governed by task dependencies. In the following, characteristics of the graphical method are first described. Task dependency relationships where a downstream design task depends on a single upstream design task are then presented in Section II-C. The case where a downstream

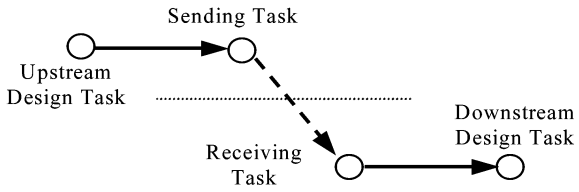

Fig. 3. Design and communication tasks in a task dependency relationship.

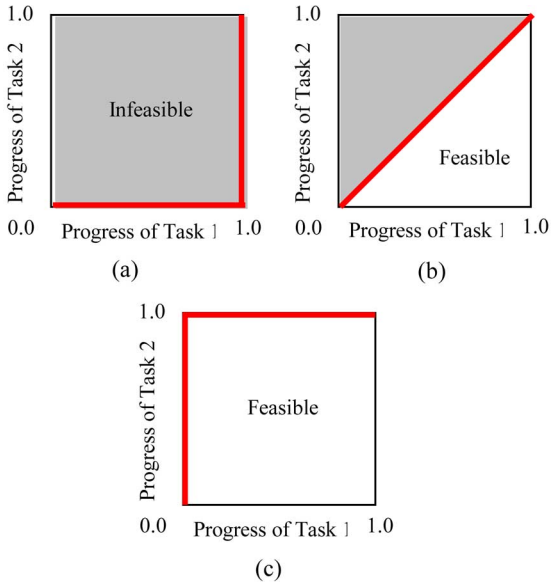

Fig. 4. Basic dependency types. (a) Precedence. (b) Pace. (c) Independence.

design task depends on multiple upstream design tasks will be presented in Section II-D

For simplicity, information generation and task progress curves, e.g., Fig. 1, are assumed to be piecewise linear and monotonically nondecreasing. As a result, a task dependency relationship can be divided into stages, each corresponding to a linear segment. The relationships within a stage can be classified into three basic types, as illustrated in Fig. 4: precedence, pace, and independence, representing, respectively, sequential, concurrent, and independent processes. In the figures, there are two general tasks: Tasks 1 and 2, each with a single stage, and Task 2 is dependent on Task 1. Fig. 4(a) describes the precedence relationship, where Task 2 can only start after Task 1 is completed. Equivalently, the progress of Task 2 remains zero until that of Task 1 reaches one, as shown by the thick lines representing the feasible area. Fig. 4(b) describes the pace relationship where two tasks can be concurrently performed, but the maximum progress of Task 2 cannot exceed the progress of Task 1. The feasible area is the unshadowed triangle below the diagonal line (including the diagonal line). Fig. 4(c) describes the independence relationship where two tasks can be independently performed, and the entire area is feasible. These three basic types will be described by linear inequality constraints in the next subsection, and a general dependency relationship will be modeled as a combination of them.

With the three basic dependency types, the dependency relationships among the tasks of Fig. 3 can now be illustrated by Fig. 5, where the downstream design task depends on a single

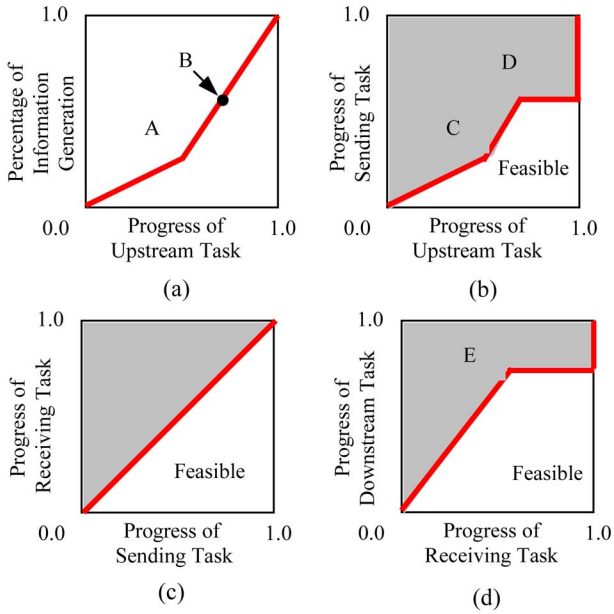

Fig. 5. Analysis of task dependencies. (a) Information generation. (b) Upstream design-sending relationship. (c) Sending-receiving relationship. (d) Receiving-downstream design relationship.

upstream design task. Assume that the percentage of information generated in the progress of the upstream design task is described by Fig. 5(a), where most information is generated after point A. In addition, it is assumed that the downstream design task can be performed with preliminary or partial information obtained in the early progress of the upstream design task [before B which corresponds to D in Fig. 5(b)], but has to wait for the complete information to finish the downstream design. The dependency relationships among the tasks are then illustrated in Fig. 5(b)–(d), as described next.

Upstream Design-Sending: Based on the dependency relationship between a pair of upstream and downstream design tasks, a stage of the sending task may have to be performed after the upstream information of that stage is completely available (precedence relationship). Or it can be performed concurrently with the upstream design task to send preliminary information, and its maximum progress[3] is constrained by how much information has been generated and the quality of the information, with a slower maximum progress for less information or lower-quality information (pace relationship). In Fig. 5(b), three stages are assumed: two paces and one precedence. For the two pace stages (ending, respectively, at C correspond to A, and D correspond to B), the sending task can be performed concurrently with the upstream design task, and the maximum progress is assumed for simplicity equal to the percentage of information generated. In the precedence stage (starting at D), the sending task has to be performed after the complete information of that stage is available.

Sending-Receiving: The methods of communication determine how sending and receiving tasks are performed. For example, in e-mail communication, the receiving task receives and digests information after the sending task (organizing and sending information) is completed. In a face-to-face meeting,

[3]The minimum progress can also be specified for synchronization between tasks and can be handled in a way similar to that of the maximum progress. For simplicity, the minimum progress is not considered.

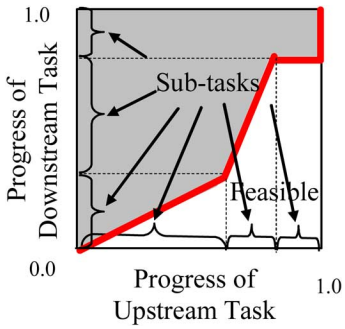

Fig. 6. Upstream–downstream design relationship.

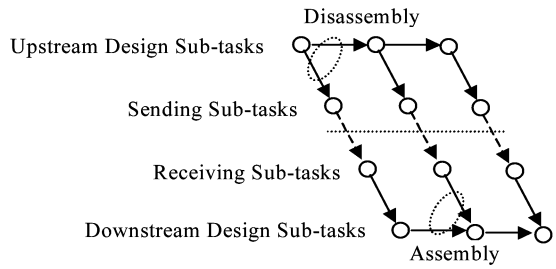

Fig. 7. Subtasks in a task dependency relationship.

the receiving task is generally performed concurrently with the sending task, with its maximum progress constrained by how much information has been communicated by the sending task. Fig. 5(c) depicts a face-to-face meeting, where the maximum progress of the receiving task is assumed for simplicity equal to the progress of the sending task.

Receiving-Downstream Design: With the information received, the maximum progress of a downstream design task is determined by the degree it requires the information, an intrinsic property of the design process. In Fig. 5(d), two stages are considered. In the first pace stage (ending at E corresponding to B and D), the downstream design task can be performed concurrently with the receiving task, assuming that it may proceed with preliminary or partial information. In the second precedence stage (starting at E), the downstream design task is performed after the receiving task is completed, assuming that complete information is required for the design.

Different from [9] where only dependency relationships among design tasks are modeled, in this paper, dependency relationships among various tasks (including communication tasks) are modeled, and will be used to determine when to perform the tasks for timely project completion. As a dependency relationship between two tasks may have multiple stages, each described by one of the three basic types in Fig. 4, tasks are divided accordingly into subtasks, as described below using the example of Fig. 5.

The dependency relationship between the upstream and downstream design tasks is first derived by mapping coordinates of "stage starting points" in Fig. 5 (b)–(d) to the progresses of the upstream and downstream design tasks, as shown in Fig. 6. These two tasks are then divided into three subtasks, one for each stage, and these subtasks are assumed to be performed sequentially.[4] The sending and receiving tasks generally do not introduce additional stages, and are divided accordingly into the same number of subtasks. For simplicity, it is assumed that there is no constraint among subtasks of a sending task, or among subtasks of a receiving task.[5] With the above introduction of subtasks and carrying over the dependency relationships of Fig. 3 to the subtask level, the relationship among a design subtask and its sending subtask and the succeeding design subtask is depicted in Fig. 7, resembling a disassembly operation; and the relationship among a design subtask and its receiving subtask and the preceding design subtask resembles an assembly operation. These dependency relationships among subtasks will be mathematically formulated next.

[4]If subtasks can be concurrently performed, then they should be redefined as tasks so that their relationships can be described by dependency models, as in Fig. 5.

[5]If there are constraints among subtasks of a communication task, they can be relaxed by using Lagrangian multipliers and handled by using the method presented in [36].

C. Subtasks and Dependency Relationships

In this section, design and communication tasks are considered as general tasks without differentiation unless explicitly stated otherwise. With the division of tasks into subtasks, subtasks are basic scheduling units, and are performed by teams in a nonpreemptive way. The required resource-hours for subtasks are estimated based on previous project experiences, and are assumed given. In the following, resource allocation and processing time requirements for subtasks are first presented. The relationships among subtasks are then formulated as linear inequality constraints.

Resource Allocation and Processing Time Requirements for Subtasks: Suppose that task i is dependent on task j, and both are divided into N subtasks, with the nth $(1 \leq n \leq N)$ subtask of i denoted as (i,n). Task i is assigned to team h_i, and the processing time of (i,n), denoted as t_{in}, is governed by the required resource-hour R_{inh} and the amount of resource allocated by team h_i, with a large amount leading to a short processing time. For simplicity, it is assumed that resource allocation within the processing period is fixed, and t_{in} is inversely proportional to the level of resource allocated and takes a few discrete values. From a different perspective, the allocation of h_i to subtask (i,n) at time k, denoted as δ_{inkh}, is inversely proportional to t_{in} within the processing period of (i,n) and zero outside [36], i.e.,

$$\delta_{inkh} = \begin{cases} \frac{R_{inh}}{t_{\text{in}}}, & \text{if} \quad h = h_i \text{ and } b_{\text{in}} \leq k \leq c_{\text{in}} \\ 0, & \text{otherwise;} \end{cases} \qquad \forall i, n, k, \text{ and } h. \quad (1)$$

In the above, b_{in} and c_{in} are the beginning and completion times of (i,n), respectively. With resource allocated, subtask (i,n) needs to be performed by team h_i for a period of time t_{in}, satisfying the following processing time requirement:

$$c_{\text{in}} = b_{\text{in}} + t_{\text{in}} - 1, \quad \forall (i,n). \quad (2)$$

The progress of (i,n) at time k, denoted as p_{ink}, is represented by the percentage of completed resource-hours [9, pp. 10–11] as follows:

$$p_{\text{ink}} = \begin{cases} 0, & k \le b_{\text{in}} \\ \frac{k-b_{\text{in}}}{t_{\text{in}}}, & b_{\text{in}} < k \le c_{\text{in}} \\ 1, & c_{\text{in}} < k \end{cases} \quad \forall i, n, \text{ and } k. \tag{3}$$

Relationships of Successive Subtasks Within a Design Task: As mentioned earlier, if i is a design task, then subtasks within i are to be performed sequentially, i.e., the $(n+1)$th subtask can only start after the nth subtask has been completed

$$c_{\text{in}} + 1 \le b_{i(n+1)}, \quad \forall\, i \text{ and } n. \tag{4}$$

Relationships of Subtasks Belonging to Two Dependent Tasks: For two general tasks i and j, where i is dependent on j, the dependency relationship between their subtasks (i,n) and (j,n) is described by one of the three basic types. For *precedence*, (i,n) can only start after (j,n) has been completed, i.e.,

$$c_{jn} + 1 \le b_{\text{in}}, \quad \forall\, n. \tag{5}$$

A *pace* relationship requires the maximum progress of (i,n) to be less than or equal to that of (j,n), implying a constraint at every k. To avoid too many constraints, an equivalent view based on constant resource allocation (1) and progress computation (3) is that (i,n) cannot be started before (j,n) is started, and cannot be completed before (j,n) is completed, i.e.,

$$b_{jn} \le b_{\text{in}} \tag{6}$$

and

$$c_{jn} \le c_{\text{in}}, \quad \forall\, n. \tag{7}$$

For *independency*, there is no constraint between the subtasks.

D. Dependencies With Multiple Upstream Design Tasks

In the above two subsections, a downstream design task is dependent on a single upstream design task. Sometimes a design task may need information from multiple design tasks. Suppose that design task i is dependent on a set of N_i upstream design tasks denoted as D_i. The dependency relationships between design task i and a design task in D_i are described by the graphical method of Fig. 5, and there are N_i sets of such relationships. Tasks are divided into subtasks based on all stage starting point of these relationships, and each set of relationships is described by constraints among relevant subtasks as formulated in Section II-C. An example is a single-chip computer data acquisition system, as shown in Fig. 8, where the software design task depends on the knowledge about both computer circuit design and data acquisition circuit design. Based on the task dependencies, each task (either a design task or a communication task) is divided into three subtasks, and a software design subtask must satisfy two sets of dependency constraints described by (5) (precedence) or (6) and (7) (pace), one for the computer circuit design and the other for the data acquisition circuit design.

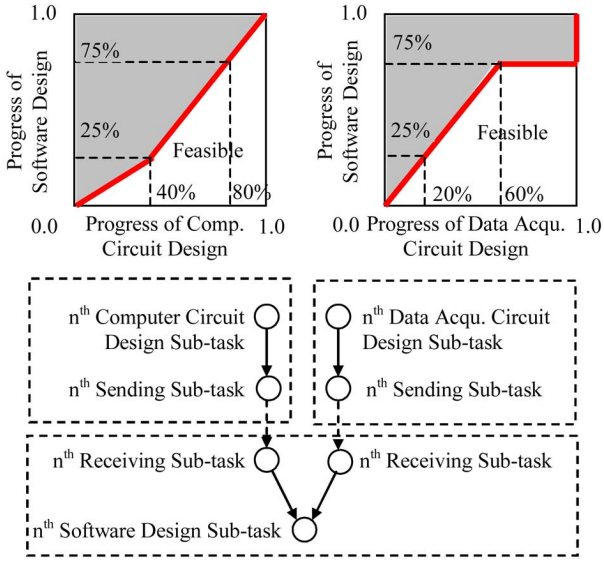

Fig. 8. Task dependency relationship with multiple upstream design tasks.

E. Resource Capacity Constraints

In addition to task dependency relationships, resource capacity constraints require that at any time, the total allocation of type h resource to tasks should be less than or equal to the resource capacity M_{kh}. Considering design and communication tasks as general tasks, the constraints are

$$\sum_{g,n} \delta_{gnkh} \le M_{kh}, \forall kh, \text{ where } g \text{ is a general task.} \tag{8}$$

F. Objective Function

As described in Section II-A, tasks are assigned to teams. The teams could be internal groups within an organization or outside contractors, and may be distributed at different locations. To coordinate them, each design task is assigned a due date [31]. These due dates could be determined through negotiation, backward planning, or other methods [16]. Penalties are imposed on missing due dates, and the objective function J' to be minimized is set to be the weighted sum of such penalties on design task tardiness, i.e.,

$$J' = \sum_i w_i T_i^2, \text{where } i \text{ is a design task.} \tag{9}$$

In the above, T_i is the tardiness of design task i, i.e., $\max(0, c_i - d_i)$ with d_i being the due date, and w_i is the weight. The weight is generally high for the last design task of the project.

Based on the above, the overall problem is to decide beginning and processing times of design, sending, and receiving subtasks to minimize J' in (9) subject to constraints (2)–(7) associated with task dependencies and constraints (8) associated with resource capacity.[6] As the objective function (9) and constraints (2)–(7) are additive in terms of design tasks, the problem formulation is separable.

[6]By minimizing the task tardiness function (9) while assuming given resource-hour requirements, the cost over run issue, an important subject for design project scheduling, should be mostly contained.

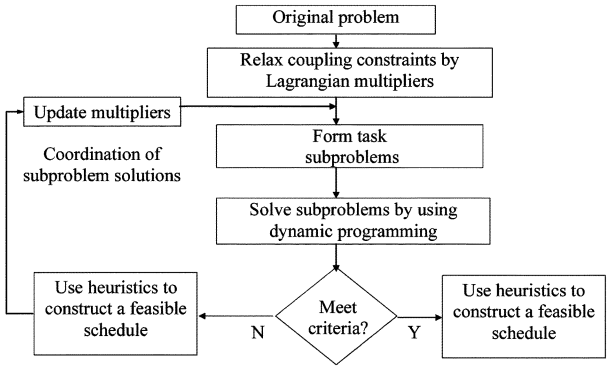

Fig. 9. Schematic of the solution process.

III. Solution Methodology

A. Overview

In view that the problem presented above has a separable structure, the Lagrangian Relaxation (LR) technique is used to decompose the problem into easier task subproblems after relaxing the coupling sending-receiving dependency constraints (5)–(7) and resource capacity constraints (8) by using Lagrangian multipliers. Motivated by a method developed for power system optimization [35], penalty terms for the violation of coupling constraints are added to the objective function (9) to improve algorithm convergence and schedule quality through the reduction of constraint violation. As the penalty terms are not additive, a surrogate optimization framework [37] is used to overcome the limitations of standard Lagrangian relaxation approaches requiring separable formulations [4]. In this framework, all terms associated with a particular design task and its associated sending and receiving tasks are pulled out from the Lagrangian to form a subproblem in Section III-B. By keeping decision variables belonging to other tasks at their latest available values, subproblems are solved by using dynamic programming to obtain solutions satisfying the "surrogate optimization condition" in Section III-C. Subproblem solutions are then coordinated through the iterative updating of Lagrangian multipliers to maximize the dual function in Section III-D. At the termination of such updating iterations, subproblem solutions, when put together, may not constitute a feasible schedule since coupling constraints have been relaxed. A heuristic procedure is then developed to construct feasible schedules based on subproblem solutions in Section III-E. The schematic is presented in Fig. 9.

The above method is based on the deterministic model of Section II. As stated earlier, task resource-hours (or processing times) could be uncertain. In this case, our method can still be applied by using expected values of probabilistic resource-hours. Results from dynamic programming then establish scheduling policies to dynamically construct schedules based on the realizations of random events also in Section III-E.

B. Problem Reformulation and Decomposition

To improve algorithm convergence and solution quality of Lagrangian relaxation, penalty terms on the violation of coupling sending-receiving dependency constraints (5)–(7) and resource capacity constraints (8) are added to the objective function as follows [35]:

$$\begin{aligned} J = \sum_i w_i T_i^2 &+ c \sum_{k,h} \max\left[0, \left(\sum_{g,n} \delta_{gnkh} - M_{kh}\right)\right] \\ &+ c \sum_{i,j,n} \max\left[0, c_{s_i(j,n)} + 1 - b_{r_j(i,n)}\right] \\ &+ c \sum_{i,j,n} \max\left[0, b_{s_i(j,n)} - b_{r_j(i,n)}\right] \\ &+ c \sum_{i,j,n} \max\left[0, c_{s_i(j,n)} - c_{r_j(i,n)}\right]. \end{aligned} \tag{10}$$

In the above, design task i is dependent on design task j; $s_i(j,n)$ and $r_j(i,n)$ denote, respectively, the nth sending and receiving subtasks between them; g represents a general task; and weight c reflects the importance of satisfying the coupling constraints. Since the original problem in Section II is separable in terms of design tasks but penalty terms with "max" operators in (10) are not separable, the problem to minimize J in (10) has a pseudoseparable structure. Our goal is to relax the coupling constraints and then decompose the relaxed problem into task subproblems, one for each design task together with its sending and receiving tasks. Therefore, coupling sending-receiving dependency constraints (5)–(7) are relaxed by using Lagrangian multipliers λ_{ijn}, μ_{ijn}, and η_{ijn}, respectively, and resource capacity constraints (8) are relaxed by using multipliers π_{kh}. The relaxed problem is to minimize L below

$$\begin{aligned} L \equiv J &+ \sum_{k,h} \left[\pi_{kh}\left(\sum_{g,n} \delta_{gnkh} - M_{kh}\right)\right] \\ &+ \sum_{i,j,n} \left\{\lambda_{ijn}\left[c_{s_i(j,n)} + 1 - b_{r_j(i,n)}\right]\right\} \\ &+ \sum_{i,j,n} \left\{\mu_{ijn}\left[b_{s_i(j,n)} - b_{r_j(i,n)}\right]\right\} \\ &+ \sum_{i,j,n} \left\{\eta_{ijn}\left[c_{s_i(j,n)} - c_{r_j(i,n)}\right]\right\}. \end{aligned} \tag{11}$$

The above minimization is subject to processing time requirements (2), design subtask precedence constraints (4), and task dependency constraints (5)–(7) between a design subtask and its sending or receiving subtasks. Decision variables are the beginning and processing times of subtasks.

In view that penalty terms with "max" operators in J are not additive, the relaxed problem cannot be directly decomposed into task subproblems. To overcome this difficulty, a subproblem for design task i is formed by pulling out all terms associated with i and its sending and receiving tasks in (11), while

keeping decision variables belonging to other tasks at their latest available values, i.e.,

$$\begin{aligned}\min \tilde{L}_i \text{ with } \tilde{L}_i \equiv\ & w_i T_i^2 + \sum_{k,h,n} (\pi_{kh}\delta_{inkh}) \\ & + c\sum_{k,h} \max\left[0, \left(\sum_{g,n} \delta_{gnkh} - M_{kh}\right)\right] \\ & + \sum_{n,j \text{ with } i \in D_j} \tilde{L}_{s_j(i,n)} + \sum_{n,j \in D_i,} \tilde{L}_{r_j(i,n)}. \end{aligned} \tag{12}$$

In the above, D_j is the set of design tasks providing information to design task j, and $\tilde{L}_{s_j(i,n)}$ are the costs associated with a sending subtask $s_j(i,n)$ that sends information from j to j

$$\begin{aligned}\tilde{L}_{s_j(i,n)} \equiv & \sum_{k,h,n} \left[\pi_{kh}\delta_{s_j(i,n)kh}\right] + \lambda_{jin} c_{s_j(i,n)} \\ & + \mu_{jin} b_{s_j(i,n)} + \eta_{jin} c_{s_j(i,n)} \\ & + c\max\left[0, c_{s_j(i,n)} + 1 - b_{r_i(j,n)}\right] \\ & + c\max\left[0, b_{s_j(i,n)} - b_{r_i(j,n)}\right] \\ & + c\max\left[0, c_{s_j(i,n)} - c_{r_i(j,n)}\right]. \end{aligned} \tag{13}$$

Similarly, $\tilde{L}_{r_j(i,n)}$ are the costs associated with a receiving subtask $r_j(i,n)$ that receives information for design task i from design task j

$$\begin{aligned}\tilde{L}_{r_j(i,n)} \equiv & \sum_{k,h,n} \left[\pi_{kh}\delta_{r_j(i,n)kh}\right] - \lambda_{ijn} b_{r_j(i,n)} \\ & - \mu_{ijn} b_{r_j(i,n)} - \eta_{ijn} c_{r_j(i,n)} \\ & + c\max\left[0, c_{s_i(j,n)} + 1 - b_{r_j(i,n)}\right] \\ & + c\max\left[0, b_{s_i(j,n)} - b_{r_j(i,n)}\right] \\ & + c\max\left[0, c_{s_i(j,n)} - c_{r_j(i,n)}\right]. \end{aligned} \tag{14}$$

The minimization in (12) is subject to processing time requirements (2), design subtask precedence constraints (4), and task dependency constraints (5)–(7) between a design subtask and its sending or receiving subtasks. Decision variables are the beginning and processing times of design subtasks within task i and the beginning and processing times of the associated sending and receiving subtasks.

C. Solving Task Subproblems With BFDP

To solve a task subproblem (12), dynamic programming (DP) is used with a stage corresponding to a subtask, and a state corresponding to the earliest possible subtask beginning time with a particular level of resource allocation. The decision for a state is to choose the beginning time for the corresponding subtask. Possible transitions between states at successive stages are delineated by the dependency relationship between the two subtasks. In the DP process, $\tilde{L}_i$ in (12) needs to be allocated to individual subtasks as stagewise costs. However, since the penalty terms for the violation of resource capacity constraints involve the max operator and are not additive in terms of subtasks, $\tilde{L}_i$ cannot be directly allocated to individual subtasks. This difficulty is overcome by approximation under the surrogate optimization framework. All terms associated with a particular subtask are pulled out from $\tilde{L}_i$ to form its stagewise cost, and decision variables not belonging to this subtask are kept at their latest available values. With the this, subproblem (12) is approximately solved. The handling of difficulties caused by complicated dependency relationships among subtasks is presented next.

As presented earlier, the relationship among a design subtask and its sending subtask(s) and the succeeding design subtask resembles a disassembly operation, while the relationship among a design subtask and its receiving subtask(s) and the preceding design subtask resembles an assembly operation. Backward dynamic programming (BDP) can solve a subproblem with disassemblies but not assemblies, and this is illustrated by the examples in Fig. 10 [36]. For a subproblem with disassembly as represented by the *tree* of Fig. 10(a), BDP starts from leaf nodes subtasks 3 and 4 to decide their optimal beginning time for each state. The algorithm then moves backward from right to left to decide the beginning time for each state of subtask 2 to minimize its cost-to-go, which is the sum of its stagewise cost plus the minimal terminal costs for subtasks 3 and 4. The beginning time for each state of the subtask 1 is then decided. The optimal trajectory is then traced forward from the initial state to determine subtask optimal beginning and processing times. If assembly exists, as shown in Fig. 10(b) and BDP is used, the above process moves backward from subtasks 3 and 4 to subtask 2, and then to both subtasks 1 and 5. When tracing forward to determine the optimal trajectory, there are two decisions on the optimal beginning and processing times for subtask 2, one from 1 and the other from 5. These two decisions, however, may be different, indicating that BDP cannot solve a subproblem with assemblies. Similarly, forward dynamic programming (FDP) can solve a subproblem with assemblies but not disassemblies. In addition, forward dynamic programming cannot handle uncertainties since the information about "where-to-go" may not be exist for certain states of a subtask.

To overcome the above difficulties, the backward/forward dynamic programming (BFDP) of Zhang *et al.* [36] is used. For the example of Fig. 10 (b), a new tree is formed by flipping subtask 5 from the left-hand side of subtask 2 to its right-hand side, as shown in Fig. 10(c). Since this new tree has a structure similar to that of Fig. 10(a), the BDP process can be carried out from right to left as for Fig. 10(a), i.e., from subtasks 3–5 to subtask 2, and then to subtask 1. Nevertheless, for each state of subtask 2, since subtask 5 in fact precedes subtask 2, the transition cost is evaluated from every state of 5 satisfying the dependency relationship between 5 and 2, and the minimal cost is the minimal "cost-to-arrive" in the forward dynamic programming. Accordingly, the cumulative cost for each state of subtask 2 is the sum of its stagewise cost plus the minimal costs-to-go for subtasks 3 and 4 plus the minimal cost-to-arrive from subtask 5. Similar to Fig. 10(a), the optimal trajectory is traced from the initial state of subtask 1 to determine subtask optimal beginning and pro-

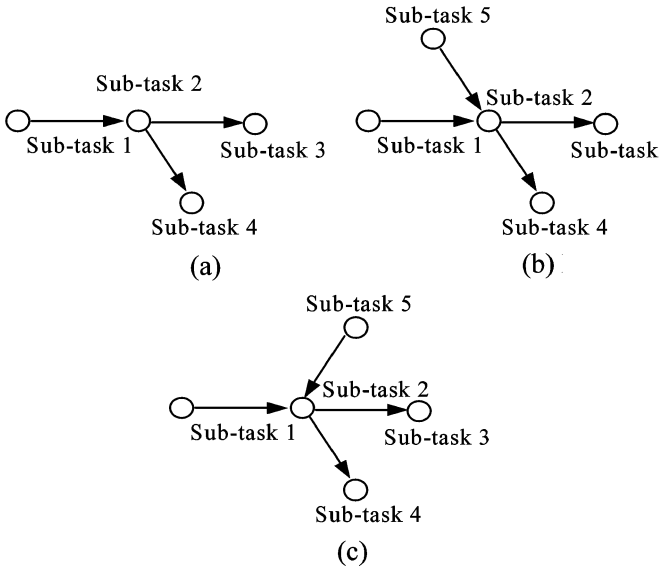

Fig. 10. Subproblems with disassembly and assembly. (a) Disassembly. (b) Disassembly and assembly. (c) New tree.

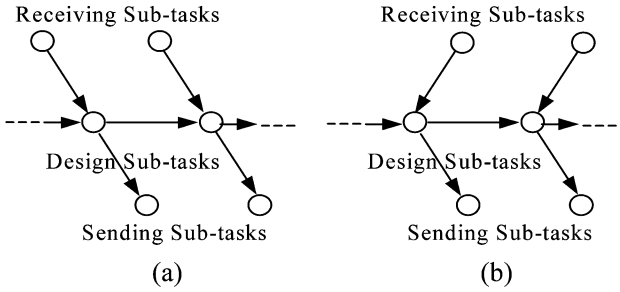

Fig. 11. Original and new trees for a task subproblem. (a) Original tree. (b) New tree.

cessing times. Since the optimal beginning and processing times of subtask 2 are decided only from subtask 1 and those for subtask 5 are decided from subtask 2, possible decision conflicts are avoided. This method combines backward DP and forward DP, and is thus called backward/forward DP. A side comment is that since this method has the forward DP component, it cannot be directly used to handle uncertainties [36].

For a subproblem (12) with design, sending, and receiving subtasks, the original and the new trees are shown in Fig. 11(a) and (b), respectively. The backward/forward DP starts from computing costs for leaf nodes, i.e., the receiving and sending subtasks in Fig. 11(b). The algorithm then moves from the last design subtask to the first design subtask to recursively minimize the cumulative cost, which is the sum of the stagewise cost plus the minimum costs-to-go from the succeeding design and sending subtasks plus the minimum costs-to-arrive from the receiving subtasks. After the minimum cumulative cost is computed for the first design subtask, the optimal trajectory is traced from the initial state to determine subtask optimal beginning and processing times.

D. Updating Lagrangian Multipliers

With task subproblems (12) solved, the high-level dual problem is to find the optimal multipliers to maximize the dual function subject to the non-negativity of multipliers, i.e.,

$$\max_{\lambda \geq 0} q(\lambda), \text{ with } q(\lambda) \equiv L^*$$

and

$$\lambda \equiv (\pi_{kh}, \lambda_{ijn}, \mu_{ijn}, \eta_{ijn})^T \tag{15}$$

where L^* is L in (11) evaluated at subproblem optimal solutions. In view that (12) are approximately solved as stated earlier, the dual value obtained is a "surrogate" dual, denoted as $\tilde{q}(\lambda, b_{gn}, t_{gn})$. The surrogate subgradient method of Zhao *et al.* [37] is then used to update the multipliers. The key idea is that a proper direction to update the multipliers can be obtained if the following "surrogate initialization condition" (16) and "surrogate optimization conditions" (17) are satisfied without optimally solving all subproblems:

$$\tilde{q}^0 < q^* \text{at the first iteration, and} \tag{16}$$

$$\tilde{q}^m \equiv \tilde{q}\left(\lambda^m, b_{gn}^m, t_{gn}^m\right) < \tilde{q}\left(\lambda^m, b_{gn}^{m-1}, t_{gn}^{m-1}\right) \text{ at the } m\text{th iteration.} \tag{17}$$

Then, it can be proved that the surrogate dual is always less than the optimal dual value q^*, i.e.,

$$\tilde{q}^m < q^* \tag{18}$$

and the corresponding surrogate subgradient forms an acute angle with the direction toward the optimal multipliers, therefore is a proper direction to update multipliers. Components of surrogate subgradient $\tilde{g}^m$ at iteration m are computed as follows:

$$\tilde{g}_{\pi_{kh}}^m = \sum_{g,n} \delta_{gnkh} - M_{kh}, \tilde{g}_{\lambda_{ijn}}^m = c_{s_j(i,n)} + 1 - b_{r_i(j,n)}$$

$$\tilde{g}_{\mu_{ijn}}^m = b_{s_j(i,n)} - b_{r_i(j,n)}, \tilde{g}_{\eta_{ijn}}^m = c_{s_j(i,n)} - c_{r_i(j,n)}. \tag{19}$$

The multipliers are then updated along the surrogate subgradient direction, i.e.,

$$\lambda^{m+1} = \lambda^m + s^m \tilde{g}^m$$

with

$$\tilde{g}^m \equiv \left(\tilde{g}_{\pi_{kh}}^m, \tilde{g}_{\lambda_{ijn}}^m, \tilde{g}_{\mu_{ijn}}^m, \tilde{g}_{\eta_{ijn}}^m\right)^T. \tag{20}$$

In the above, s^m is the step size at iteration m, and should satisfy the following "surrogate stepsize condition":

$$0 < s^m < (q^* - \tilde{q}^m)/\|\tilde{g}^m\|^2. \tag{21}$$

In view that q^* is generally unknown, it needs to be estimated. For this purpose, a feasible solution and its corresponding cost J^m are obtained at iteration m by using a heuristic procedure to be presented in Section III-E. The estimated optimal dual value

$\tilde{q}^*$ is then obtained as the average of the best feasible cost and the highest surrogate dual value obtained thus far. Step size s^m is then calculated as

$$s^m = \alpha^m(\tilde{q}^* - \tilde{q}^m)/\|\tilde{g}^m\|^2,\ 0 < \alpha^m > 1. \tag{22}$$

To evaluate solution quality, a lower bound to the optimal feasible cost is required. If all the three surrogate conditions (16, 17, and 20) are satisfied, $\tilde{q}^m$ is a lower bound according to (18). However, since $\tilde{q}^*$ is an estimate of q^*, s^m of (22) may not satisfy condition (21). Consequently, $\tilde{q}^m$ may not be a lower bound to the optimal feasible cost. To obtain a lower bound, the original objective function (9) without penalty terms is used at the end of the surrogate subgradient iterations when the relative surrogate duality gap $(J^m - \tilde{q}^m)/\tilde{q}^m$ is less than a given threshold, or when the maximal number of iterations or the maximal CPU time has been reached. Since the problem with the original objective function (9) is separable, a few iterations of the traditional LR method starting with the latest available multipliers would generate a reasonable dual cost, which serves as a lower bound to the optimal feasible cost to provide a measure of solution quality. The original objective function (9) is also used at the first iteration to satisfy surrogate initialization condition (16).

E. Constructing Feasible Schedules

Since coupling sending-receiving dependency constraints (5)–(7) and resource capacity constraints (8) have been relaxed, subproblem solutions, when put together, generally do not constitute a feasible schedule. A list scheduling heuristic based on Zhang *et al.* [36] is thus developed to generate feasible schedules using resource allocation levels determined in subproblems. In the procedure, a list of "assignable" subtasks with relevant dependency relationships satisfied is created at time zero and updated at subsequent time units. Subtasks in the set are sorted in the ascending order of their subproblem beginning times. Subtasks are scheduled according to the list if the required resources are available. If there is not enough resource for a subtask at a particular time, the subtask is delayed by one time unit. When multiple subtasks compete for a limited amount of a resource, the incremental changes in cumulative DP costs are compared for competing subtasks. Subtasks with higher incremental changes are then scheduled and other subtasks are delayed by one time unit. After a subtask is scheduled, it is removed from the list, and its succeeding subtasks are added to the list if their relevant dependency relationships are satisfied. Note that if a subtask and its succeeding subtask have a pace relationship, they may start at the same time. The above process continues until all subtasks are scheduled. In addition to the above procedure, other heuristic methods such as the constraint propagation technique of Wim and Pape [32] may also be applied to construct a feasible schedule.

The above method is based on the deterministic model of Section II. As stated earlier, the resource-hours (or processing times) required by subtasks could be uncertain. In this case, our method can still be applied by using the expected values of probabilistic subtask resource-hours. Results from the backward/forward DP are then used as scheduling policies describing what to do under which circumstances. As stated in Section III-C, backward/forward DP has both backward and forward DP components. For a backward DP component, the beginning time of a subtask at a particular state is determined by tracing the optimal policy forward in stages based on the realization of random events. For a forward DP component, however, such a policy may not exist since the information on "where-to-go" may not exist. For example, if an uncertain receiving subtask is performed with a processing time varied from its expected value, then the optimal beginning time of the subsequent design subtask may not be obtainable because of the lack of "where-to-go" information. In this case, the beginning time of the subsequent design subtask is set to be the solution obtained in DP if the dependency relationship is satisfied, or the beginning time is postponed to satisfy the dependency relationship. Another way to handle the forward DP component is to restart the DP process from the current time based on the realizations of random events. With subtask beginning times determined based on the above policies, the list scheduling heuristic presented above can be used to dynamically construct schedules.

IV. Numerical Results

The above method has been implemented by using the object-oriented language C++ and tested on a Pentium IV 1.8 GHz personal computer running the Windows XP operating system. In the following, three examples are presented. In Example 1, scheduling communication and design activities together is compared with the case where communication activities are not considered initially, but are performed when design teams need information to proceed further. In Example 2, our method is compared with the LR method without the additional penalty terms and with a heuristic method based on a modified *shortest processing time/critical ratio* (SPT/CR) rule. The effects of different penalty weights are also examined. In Example 3, schedules are dynamically constructed according to DP policies for a project with uncertain resource-hours. Simulations are then performed to compare the performance with that of the modified SPT/CR rule.

Example 1: In this example, two cases are examined where communication activities are scheduled in different ways. In the first case, communication and design subtasks are scheduled together by using our method. In the second case, communication tasks are not considered initially, but are performed when design teams need information to proceed further. As a result, the corresponding downstream design tasks and their subsequent tasks may be delayed because of waiting for information. Three problem instances with different project sizes are examined. In the first instance, a simple project is considered so that results can be illustrated by using Gantt charts. Two more instances are then examined to compare the results as project size increases.

For the first problem instance, a project with two design tasks are considered with data summarized in Table I. Design task 2 is dependent on the knowledge of design task 1, and there are a sending task and a receiving task between them. Among these tasks, there are three dependency relationships, each having a single stage and described by the pace relationship. Thus each task has one subtask. The resource-hours are given, with communication constituting a significant portion of the project time as stated earlier. Subtasks are to be assigned to two teams, each with two designers. In the objective function, design task 2 has

TABLE I
DATA FOR INSTANCE 1 IN EXAMPLE 1

Task	Resource –hours	Team	Due Date	Tardiness Weight
Design Task 1 (Sub-Task (1, 1))	8	Team 1	4	1.0
Sending Task (Sub-Task $S_2(1, 1)$)	6	Team 1	/	/
Receiving Task (Sub-Task $R_1(2, 1)$)	6	Team 2	/	/
Design Task 2 (Sub-Task (2, 1))	4	Team 2	6	10.0

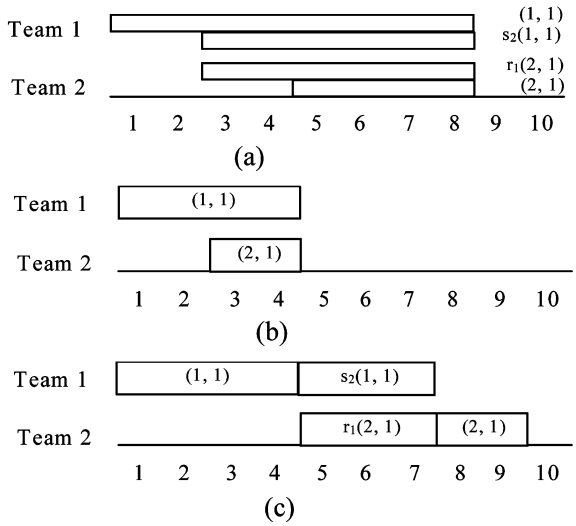

Fig. 12. Gantt charts for Problem Instance 1 (a) Case 1: Schedule with both communication and design activities. (b) Case 2: Schedule without communication activities. (c) Case 2: Realized schedule with communication activities added.

a higher tardiness penalty weight since it is the last design task of the project.

For Case 1, the schedule obtained by using our method is illustrated by the Gantt chart of Fig. 12 (a), where each team performs its design and communication subtasks in parallel by allocating half of its resource to each subtask. The feasible cost is 56 with a zero duality gap. For Case 2, only design subtasks are scheduled initially, as shown in Fig. 12(b), and the feasible cost is zero. However, this result is misleading since communication activities are not considered. When design subtask (2, 1) needs information from (1, 1), sending subtask $s_2(1,1)$ and receiving subtask $r_1(2,1)$ have to be added, and design subtask (2, 1) has to wait until the required information is received. The realized schedule is shown in Fig. 12(c) with a feasible cost of 90. This cost is much higher than that of Case 1, with an increase of 60.7%. The results show that proper scheduling of design and communication activities together can avoid the significant delay caused by waiting for information.

For the second problem instance, the project has 30 design tasks and 16 communication tasks. There are 24 dependency relationships among them, and each task is divided into 1 to 5 subtasks. These subtasks are to be assigned to six teams, each with one to three designers. Results for this problem instance are summarized in Table II. Similar to the previous problem instance, design and communication subtasks are scheduled together for Case 1. The feasible cost is 5366 with a duality gap of 4.4%. For Case 2, only design subtasks are scheduled initially, and the feasible cost is 2844. When communication subtasks are added for the design teams to proceed further, the feasible cost for the realized schedule is 13 211, which is much higher than that of Case 1 with an increase of 146.2%.

TABLE II
PERFORMANCE COMPARISONS BETWEEN CASES 1 AND 2

Problem Instance		CASE 1	Case 2 (a)	Case 2 (b)	Increase Percentage
Inst. 2	Feasible Cost	5366	2844	13211	146.2%
	Duality Gap	4.4%	3.2%	/	/
	CPU Time (s)	112	68	/	/
Inst. 3	Feasible Cost	12263	6514	33932	176.7%
	Duality Gap	4.9%	4.2%	/	/
	CPU Time (s)	209	121	/	/

For the third problem instance, the project has 60 design tasks and 32 communication tasks. There are 48 dependency relationships among them, and each task is divided into 1 to 5 subtasks. These subtasks are to be assigned to ten teams, each with one to three designers. Results are also summarized in Table II. For Case 1, design and communication subtasks are scheduled together, and the feasible cost is 12 263 with a duality gap of 4.9%. For Case 2, only design subtasks are scheduled initially, and the feasible cost is 6514. When communication subtasks are added for the design teams to proceed further, the feasible cost for the realized schedule is 33 932, which is much higher than that of Case 1 with an increase of 176.7%. Therefore, the conclusion presented before that proper scheduling of design and communication activities together can avoid the significant delay caused by waiting for information still holds when the project size increases.

Example 2: A project with a total of 100 design tasks and 100 communication tasks is to be scheduled. There are 150 dependency relationships among tasks, and each task is divided into 1 to 5 subtasks. These subtasks are to be assigned to 20 teams, each with one to four designers. Our method is compared with the LR method without penalty terms on the violations of coupling constraints. It is also compared with a heuristic method based on a modified *shortest processing time/critical ratio* (SPT/CR) rule. The original SPT/CR rule has been proved to be effective for job shop scheduling [38]. It gives a subtask high priority if the subtask has a short processing time, and low priority if the subtask has slack time to meet its due date. To reflect different levels of importance of tasks, the SPT/CR index for subtask (i, n) at time k is modified to include the tardiness penalty weight

$$\text{modified SPT/CR} \equiv \frac{w_i}{t_{\text{in}} \max[1, (d_i - k)/r_i]}. \tag{22}$$

TABLE III
PERFORMANCE COMPARISONS OF DIFFERENT METHODS

Compared Items	LR with penalty terms	LR without penalty terms	Heuristics based on SPT/CR
Feasible Cost	16086	19418	29054
Constrain Violation	1346	2860	/
Duality Gap	7.9%	12.2%	/
CPU Time (s)	376	745	7.2

TABLE IV
PERFORMANCES OF THE LR METHOD WITH DIFFERENT PENALTY WEIGHTS

Compared Items	c = 1	c = 3	c = 5	c = 15
Feasible Cost	18903	16086	17823	32314
Duality Gap	11.3%	7.9%	10.2%	/
CPU Time (s)	475	376	429	600

In the above, t_{in} is the processing time of (i, n), and r_i is the remaining processing time for task i.

Results of different methods are summarized in Table III, where "constraint violation" is computed as the sum of the violations of coupling constraints by subproblem solutions, including the sending-receiving dependency constraints (5)–(7) and resource capacity constraints (8).

Comparing to the LR method without penalty terms, our method (LR with penalty terms) has a lower feasible cost, a lower duality gap, and requires less computation time. This is because our method improves the feasibility of subproblem solutions, thus requiring less iteration to converge and fewer adjustments of subproblem solutions to construct a feasible solution. Results also show that both LR methods generate better schedules than that of the modified SPT/CR rule.

To study the effects of penalty weight c associated with the violations of coupling constraints on solutions, four values of c are examined. Results are summarized in Table IV.

In the table, the lowest feasible cost 16 086 and the shortest computation time 376 s are obtained when c equals three. When c equals 15, the algorithm cannot converge because this c leads to an ill-conditioned dual problem. The above results demonstrate that appropriately setting penalty weight c is important and can lead to low cost and fast convergence. The appropriate value of c can be selected by experimentation.

Example 3: In this example, schedules are dynamically constructed for a small project with two design tasks, four communication tasks, and uncertain resource-hours. There are six dependency relationships among the tasks, and each task is accordingly divided into three subtasks. These subtasks are to be assigned to two teams, each with two designers. Data are summarized in Table V. Three cases with different levels of uncertainties are considered, where the distributions of uncertain resource-hours are assumed Gaussian with standard deviations of 10% (low), 20% (medium), and 40% (high) of their expected values.

Our method is compared with the modified SPT/CR rule. For our method, the deterministic problem is first solved by using the expected values of resource-hours to obtain scheduling policies. The method presented in Section III-E is then used to dynamically construct schedules based on the realizations of random events. Fifty Monte Carlo runs are performed for each uncertainty level, and results are summarized in Table VI. The optimal comparison technique is then used to compare the two methods, where the significance of comparison is computed based on hypothesis testing [3].

TABLE V
PROJECT DATA FOR EXAMPLE 3

Task	Expected Resource –hours	Team	Due Date	Tardiness Weight
Design Task 1	40	Team 1	40	1.0
Sending Task 1	30	Team 1	/	/
Receiving Task 1	20	Team 1	/	/
Task 2	30	Team 2	60	10.0
Sending Task 2	20	Team 2	/	/
Receiving Task 2	30	Team 2	/	/

TABLE VI
SIMULATION RESULT FOR EXAMPLE 3

Standard Deviation of Resource-Hours	Mean Feasible Cost, DP Policies	Standard Deviation, DP Policies	Mean Feasible Cost, SPT/CR	Standard Deviation, SPT/CR
Low 10%	518	62	1084	84
Medium 20%	860	186	1242	238
High 40%	1632	524	1806	606

Results in Table VI demonstrate that the mean feasible costs obtained by using our method are lower than those obtained by using the modified SPT/CR rule. For the low uncertainty (10%) case, our method is better than the heuristic method with 99.98% confidence. For the medium uncertainty (20%) case, our method is better than the heuristic method with 99.22% confidence. For the high uncertainty (40%) case, our method is still better than the heuristic method, however, the confidence level drops to 81.06%. This is because our scheduling policies are obtained based on the deterministic model, therefore become less effective as standard deviations increase.

V. CONCLUSION

In this paper, a novel optimization model has been established to schedule design projects. Task dependencies and the associated communication activities are explicitly modeled to represent sequential, concurrent, and independent processes. The problem is solved by using the Lagrangian relaxation technique within the surrogate optimization framework. Numerical results demonstrate that high-quality schedules are generated in a computationally efficient way, and penalties on the violations of coupling constraints effectively improve schedule quality and algorithm convergence. Furthermore, scheduling design and communication subtasks together avoids the high costs associated with the delay caused by waiting for information. In case that task resource-hours are uncertain, results based on the deterministic model provide good policies that can be used to dynamically construct schedules.

The above method provides a new direction to solve inseparable problems. Also, although the scheduling of a single project is presented, the method can be extended to handle multiple

projects that may share common resources through relaxation. In addition, the method can be extended to formally handle uncertain process times by following Luh *et al.* [20], and uncertain number of design iterations by following Luh *et al.* [21] by combining Lagrangian relaxation with stochastic dynamic programming instead of deterministic dynamic programming.

REFERENCES

[1] P. S. Adler, A. Mandelbaum, V. Nguyen, and E. Schwerer, "From project to process management: An empirically-based framework for analyzing product development time," *Manage. Sci.*, vol. 41, no. 3, pp. 458–484, 1995.

[2] T. Ahn and S. S. Erenguc, "The resource constrained project scheduling problem with multiple crashable modes: A heuristics procedure," *Eur. J.Oper. Res.*, vol. 107, pp. 250–259, 1998.

[3] Y. Bar-Shalom and X. R. Li, *Estimation and Tracking: Principles, Techniques and Software*. Norwood, MA: Artech, 1993.

[4] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA: Athena Scientific, 1999.

[5] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Scheduling projects to resource constraints: Classification and complexity," *Discrete Appl. Math.*, vol. 5, pp. 11–24, 1983.

[6] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *Eur. J. Oper. Res.*, vol. 112, no. 1, pp. 3–41, 1999.

[7] K. Brinkmann and K. Neumann, "Heuristic procedures for resource-constrained project scheduling with minimum and maximal time lags: The minimum project-duration and resource-leveling problem," *J. Decision Syst.*, vol. 5, pp. 129–156, 1996.

[8] S. Chen and L. Lin, "Decomposition of interdependent task group for concurrent engineering," *Comput. Ind. Eng.*, vol. 44, pp. 435–459, 2003.

[9] A. D. Christian, "Simulation of Information Flow in Design," Ph.D. dissertation, Dept. Mech. Eng., Massachusetts Inst. Technol., Cambridge, MA, 1995.

[10] E. Demeulemeester and W. Herroelen, "An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 90, pp. 1803–1818, 1996.

[11] S. D. Eppinger, D. E. Whitney, R. P. Smith, and D. A. Gebala, "A model-based method for organizing tasks in product development," *Res. Eng. Des.*, vol. 6, no. 1, pp. 1–13, 1994.

[12] R. M. Eastman, "Engineering information release prior to final design freeze," *IEEE Trans. Eng. Manage.*, vol. EM-27, pp. 37–41, May 1980.

[13] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Naval Research Logistics*, vol. 45, no. 7, pp. 733–750, 1998.

[14] S. Hartmann, "Project scheduling with multiple modes: A genetic algorithm," *Ann. Oper. Res.*, vol. 102, pp. 111–135, 2001.

[15] R. Heilmann, "A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags," *Eur. J. Oper. Res.*, vol. 144, no. 2, pp. 348–365, 2003.

[16] D. S. Kelly, "Team design problem in technology," *J. Ind. Technol.*, vol. 21, no. 1, pp. 2–8, 2005.

[17] R. Kolisch, A. Sprecher, and A. Drexl, "Characterization and generation of a general class of resource-constrained project scheduling problems," *Manage. Sci.*, vol. 41, no. 10, pp. 1693–1703, 1995.

[18] V. Krishnan, "Managing the simultaneous execution of coupled phases in concurrent product development," *IEEE Trans. Eng. Manage.*, vol. 43, no. 2, pp. 210–217, 1996.

[19] A. Kusiak, J. Wang, D. W. He, and C. Feng, "A structured approach for analysis of design processes," *IEEE Trans. Components, Packaging, Manuf. Technol.—Part A*, vol. 18, no. 3, pp. 664–673, 1995.

[20] P. B. Luh, D. Chen, and L. S. Thakur, "An effective approach for job-shop scheduling with uncertain processing requirements," *IEEE Trans. Robot. Autom.*, vol. 15, no. 2, pp. 328–339, 1999.

[21] P. B. Luh, F. Liu, and B. Moser, "Scheduling of design projects with uncertain number of iterations," *Eur. J. Oper. Res.*, vol. 113, pp. 575–592, 1999.

[22] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 333–346, 2002.

[23] M. D. Morelli, S. D. Eppinger, and R. K. Gulati, "Predicting technical communication in product development organizations," *IEEE Trans. Eng. Manage.*, vol. 42, no. 3, pp. 215–222, 1995.

[24] Y. Narahari, N. Viswanadham, and V. K. Kumar, "Lead time modeling and acceleration of product design and development," *IEEE Trans. Robot. Autom.*, vol. 15, no. 5, pp. 882–896, 1999.

[25] T. Nazareth, S. Verma, S. Bhattacharya, and A. Bagchi, "The multiple resource constrained project scheduling problem: A breadth-first approach," *European Journal of Operational Research*, vol. 112, no. 2, pp. 347–366, 1999.

[26] L. Ozdamar, "A genetic algorithm approach to a general category project scheduling problem," *IEEE Trans. Syst., Man, Cybern. Part C*, vol. 29, no. 1, pp. 44–59, 1999.

[27] B. D. Reyck and W. Herroelen, "The multi-mode resource-constrained project scheduling problem with generalized precedence relations," *Eur. J. Oper. Res.*, vol. 119, pp. 538–556, 1999.

[28] M. E. Sosa, S. D. Eppinger, M. Pich, D. G. McKendrick, and S. K. Stout, "Factors that influence technical communication in distributed product development: An empirical study in the telecommunications industry," *IEEE Trans. Eng. Manage.*, vol. 49, no. 1, pp. 45–58, 2002.

[29] D. V. Steward, *Systems Analysis and Management: Structure, Strategy, and Design*. Princeton, NJ: Petrocelli Books, 1981.

[30] V. Valls and F. Ballestin, "A population-based approach to the resource-constrained project scheduling problem," *Ann. Oper. Res.*, vol. 131, pp. 305–324, 2004.

[31] M. Vanhoucke, E. Demeulemeester, and W. Herroelen, "An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem," *Ann. Oper. Res.*, vol. 102, pp. 179–196, 2001.

[32] N. Wim and C. L. Pape, "Constraint-based job shop scheduling with ILOG scheduler," *J. Heuristics*, vol. 3, no. 1, pp. 271–286, 1998.

[33] J. Yan and C. Wu, "A scheduling approach for design activities in concurrent engineering," *IEEE Trans. Syst., Man, Cybern., Part C*, vol. 31, no. 3, pp. 361–365, 2001.

[34] M. R. Zamani, "A high-performance exact method for the resource-constrained project scheduling problem," *Comput. Oper. Res.*, vol. 28, no. 14, pp. 1387–1401, 2001.

[35] Q. Z. Zhai, X. H. Guan, and J. Cui, "Unit commitment with identical units: Successive subproblem solving method based on lagrangian relaxation," *IEEE Trans. Power Syst.*, vol. 17, no. 4, pp. 1250–1257, 2002.

[36] Y. Zhang, P. B. Luh, K. Narimatsu, T. Moriya, T. Shimada, and L. Fang, "A macro-level scheduling method using lagrangian relaxation," *IEEE Trans. Robot. Autom.*, vol. 17, no. 1, pp. 70–79, 2001.

[37] X. Zhao, P. B. Luh, and J. Wang, "The surrogate gradient algorithm for lagrangian relaxation method," *J. Optim. Theory, Appl.*, vol. 100, no. 3, pp. 699–712, 1999.

[38] R. Shafaei and P. Bruno, "Workshop scheduling using practical data: Part 1: The performance of heuristic scheduling rules in a dynamic job shop environment using a rolling time horizon approach,," *Int. J. Prod. Res.*, vol. 37, no. 17, pp. 3913–3925, 1999.

Ming Ni received the Ph.D. degree in electrical engineering from the University of Connecticut, Storrs, in 2006.

He is currently with ISO New England, Holyoke, MA. His research interests include optimization theory and algorithms, and their applications to manufacturing systems and wholesale power market.

Peter B. Luh (F'95) received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1973, the M.S. degree in aeronautics and astronautics engineering from the Massachusetts Institute of Technology (MIT), Cambridge, in 1977, and the Ph.D. degree in applied mathematics from Harvard University, Cambridge, MA, in 1980.

Since then he has been with the University of Connecticut, and currently is the SNET Professor of Communications and Information Technologies and the Head of the Department of Electrical and Computer Engineering. He is also a Visiting Professor at the Center for Intelligent and Networked Systems, Department of Automation, Tsinghua University, Beijing, China. He is interested in planning, scheduling, and coordination of design, manufacturing, and supply chain activities; configuration and operation of elevators and HVAC systems for normal and emergency conditions; schedule, auction, portfolio optimization, and load/price forecasting for power systems; and decision-making under uncertain or distributed environments.

Prof. Luh is the founding Editor-in-Chief of the new IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, an Associate Editor of *IIE Transactions on Design and Manufacturing*, an Associate Editor of *Discrete Event Dynamic Systems*, and was the Editor-in-Chief of the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION (1999–2003).

Bryan Moser, photograph and biography not available at the time of publication.