An Evaluation Function to Compare Alternative Commitments During Manufacturing Planning and Scheduling

A Safavi and S F Smith
Carnegie Mellon University, USA

Abstract

Search-based approaches to scheduling require a mechanism to evaluate alternative scheduling commitments during the scheduling search process. This paper is to describe a domain independent evaluation function to compare alternative commitments during scheduling in both single agent and distributed scheduling problems based on cost/benefit analysis[1]. Our work extends existing search-based approaches by providing an evaluation function which can (1) measure the global impact of a commitment on the entire schedule, and (2) evaluate commitments in distributed scheduling environments. We present our evaluation function in the framework of software manufacturing problem domain which is a representative of a distributed scheduling problem. A program called NEGOPRO has been implemented to test the development and reactive revisement of software project schedules using the evaluation function that has been developed.

1. Introduction

Software project planning and scheduling (SPPS) is a distributed problem solving process, involving a set of agents with conflicting interests [12], in which each agent specifies a set of resource requests in order to satisfy his responsibilities. There are several reasons why SPPS is hard:

1. Even idealized formulations of the problem are NP-hard in the general case [7].
2. It involves face-to-face human negotiation between multiple agents to resolve the scheduling conflicts (i.e. unsatisfiable resource requests) that arise due to differences in goals, technical judgements, etc [5].
3. There is typically considerable uncertainty in budget (i.e. resource requirement) estimates and other project planning/scheduling constraints that must be accounted for [3, 13].
4. Software project planning and scheduling is not a static problem. Schedules must be continually revised over the course of the project as changes in planning/scheduling assumptions become known [6].

Since a project product can be developed in more than one way, alternative plans are normally generated and evaluated to choose the best one among them. Project plans are evaluated by studying alternative schedules that can implement them. A software project schedule is a specification that for each product in the plan includes the names, quantities, and absolute periods of the resources that have to be reserved for its production. Each product has a set of feature requirements (e.g. deadline requirement, reliability requirement) that can be met to varying degrees. The resource reservation data can be used to assess how well the project can meet the resource requirements of its plan if that plan is chosen to be implemented. In this paper, we consider both the selection of a project plan (production plan) and the scheduling of that plan, and frequently refer to it as *scheduling*.

Evaluation functions for preference analysis in manufacturing scheduling have been under investigation by operations research and more recently by artificial intelligence independently. Although the focus of these approaches has been on the shop and factory scheduling, the techniques that they have developed can be applied to SPPS as well. Generally speaking, an evaluation function trades off the cost and benefit of making a scheduling commitment. The cost of making a scheduling commitment refers to the cost of such items as material, machine, and labor which are used in that commitment, while benefit of making a scheduling commitment refers to the effect of the commitment on such items as revenue. Cost and benefit of a scheduling alternative can be quantified in terms of dollars or utility functions[2] [10, 24] depending on the model which is assumed.

Cost/benefit analysis has been widely used by operations research for making scheduling decisions and optimizing shop parameters (e.g. present value of cash flow 1-1) [1, 9, 15, 25]. The main problem with cost/benefit

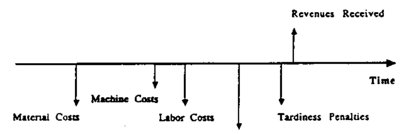

Figure 1-1: Cash flows due to scheduling a job

analysis in operations research literature is that their investigation has been limited to the case where all cost and benefit data are provided by a single agent. In SPPS there is a possibility of discrepancy in the perceived costs and benefits of product requirements since they are specified by different decisions-makers with conflicting interests. [21, 23] have proposed distributed frameworks to combine the utilities of different agents. The problem with these frameworks is that they assume the preferences of agents are on the same scale and therefore can be compared and used directly to prioritize them [18]. We have extended cost/benefit analysis to distributed decision-making environments, and have provided a framework for reconciling the conflicting cost/benefit data by considering the authority structure of the project organization. This framework is domain independent and can be used in other distributed manufacturing scheduling domains as well.

A second problem with cost/benefit analysis in operations research and economics literature is that they evaluate the cost and benefit of a scheduling commitment *k* locally relative to the conflict (a resource request that can not be met) that *k* is trying to solve. The problem with this approach is that if the conflict which is being resolved is not significantly important to the project as a whole, then a local measurement of importance of the commitment would depict an inaccurate picture of how important it is to make the commitment. We show how to assess the global impact of a commitment on different project feature requirements. Furthermore, we measure the importance (benefit) of a commitment locally when local preferences provide a good estimator for global preferences; otherwise, the global importance of the commitment is measured.

Artificial intelligence approaches to manufacturing scheduling [4, 16, 21] have relied on utility functions as a means of preference analysis. In these approaches, utility function has been used to specify the preference to reserve one resource (or start time) over another in an activity or to prioritize orders, and also to represent the utility after accounting for both cost and benefit. We have identified two problems with this approach: (1) it is difficult to develop a utility function that accounts for both cost and benefit when cost and benefit have a dynamic nature (i.e. their value is subject to specific situation) (2) it is not extendible to a distributed framework because the utilities of different agents relative to the same commitment can not be compared accurately [19]. More recent work (e.g. [17]) has used constraint propagation and analysis to measure the global impact of a commitment, but this impact is measured only relative to the project deadline (while a project has other feature requirements such as reliability and portability that might be affected by propagation as well).

The organization of the remaining of this paper is as follows. First we provide a formal definition of SPPS. We then describe an evaluation functions for comparing alternative scheduling commitments, or evaluating part (or all) of a schedule. Finally, we present some experimental results obtained with the NEGOPRO program. NEGOPRO exploits our approach to schedule evaluation within a heuristic search framework to refine and revise software project schedules. of a set of search operators, the evaluation function that has been developed, and

Further details of the overall heuristic search model can be found in [19].

[1]This research has been sponsored in part by DARPA under contract #F30602-88-C-0001 and in part by the Robotics Institute at C.M.U.

[2]A utility function is normally a 1-1 continuous function *utility*:$A \rightarrow I$ where A is the set of alternative commitments and $I = [0\ 1]$ represents the normalized preference for each alternative.

2. Formal Definition of the Problem

The notation used is described in two parts: symbols and constructs. The symbols will be formally defined as they appear in the formal definition of the problem. The symbols uniquely identify a planning/scheduling concept while constructs are used to manipulate them. Let r, p, p^*, and p^{**} be resources, F be the set of possible product feature requirements of all products, Φ denote a set of product feature requirements of a product and Φ_p^i denote the i-th feature requirement of p, Γ denote a process plan and Γ^l_p denote the l-th process plan of a product p, Λ be a schedule and Λ_Γ^i denote the i-th schedule of process plan Γ, Λ^* and Γ^* denote the set of all schedules and all process plans of a product respectively, $[a_i]$ denote an ordered list of objects a_i, $Card(A)$ denote the cardinality of a set or an ordered list A, f,g,h be functions, membership test for ordered lists be defined and be referred to by the symbol $\in$, $[a\ b]$ *s.t.* $a,b \in N$ denote the inclusive set of numbers between a and b, and $(a_1\ a_2\ ...)$ denote a multi-dimensional domain.

We begin by considering the resources that must be allocated to support a given software project and the specification of the temporal constraints surrounding resource usage.

Definition 1: Resource

Let R be the set of all resources that can be used and produced in an organization G. Then $R=R_{pro} \cup R_{pri}$ where $R_{pro} = \{r_j\}$ *s.t.* $\exists \Pi_i\ r_j$ is the product of Π_i and R_{pri} is the complement of R_{pro} in R.

R_{pri} denotes the set of *primitive resources* in G and includes those resources that are not produced inhouse. In contrast, R_{pro} denotes the set of *products* in G and includes those resources that are produced inhouse[3]. A file server, an office, any type of software, hardware, documentation, staff, or time are typical examples of primitive resources. Furthermore, all products in a software project are either software or documentation and are considered to be "infinite capacity" resources (therefore they need to be produced only once).

There are three types of temporal resource constraints: availability constraints, reservations (resource allocations), and resource requests. All temporal resource constraints share a common representation. We use the symbol ζ to refer to a prototypical temporal resource constraint. When multiple temporal resource constraints are needed, we use subscripts to distinguish between different temporal resource constraints. Furthermore, for a product p, we use r_ζ to refer to the resource requested by ζ, $\zeta(p)$ to refer to the set of all resource requests of p under a fixed process plan, and $r_\zeta(p)$ to refer to the set of all resource names of a fixed process plan of p[4].

Definition 2: Primitive Resource

Let R_{pri} be the set of all primitive resources that can be used in an organization G, R_{Upri} denote the set of unshared resources, R_{Spri} denote the set of shared resources, and *includes* be a predicate such that *includes*$(\zeta,[t_0\ t_1])$ is true iff the temporal/capacity constraint ζ includes the period $[t_0\ t_1]$.

```
Then $R_{pri}=R_{Upri} \cup R_{Spri}$ s.t.
$\forall p_1,p_2 \in R_{pro}$  $\forall \zeta_1$ (reservation of $p_1$), $\forall \zeta_2$ (reservation of $p_2$)
  if $(r_{\zeta_1}=r_{\zeta_2} \wedge r_{\zeta_1} \in R_{Upri} \wedge \exists [t_0 t_1]$ includes$(\zeta_1,[t_0\ t_1]) \wedge$
    includes$(\zeta_2,[t_0\ t_1]))$
  then $p_1=p_2$
  else $r_{\zeta_1} \in P_{Spri}$
  endif
```

A file server, an office, and any type of software or documentation are typical examples of shared resources while a system analyst, a coder or a workstation are typical examples of unshared resources. A system designer is an unshared resource despite the fact that it can be allocated to different projects at the same time. This is because if the size of time window is chosen small enough (e.g. manhour) a system designer can be working only on one project. *Time* is considered to be a shared resource since many activities (if their temporal ordering allows) can execute in parallel thus share the same unit of time. However, time has a special status that no other resource has, namely it is not treated as a separate resource and instead is implicit in the specification of ζ that involves other resources.

Definition 3: if ζ is a temporal/capacity constraint for a resource r by a product p, then ζ is of the form:

```
if $r \in R_{Upri}$ then $[(t_{2i}\ t_{2(i+1)}\ q_i)]$  $\forall i \in 0..n$
else if $r \in R_{Spri}$ then $[(t_{2i}\ t_{2(i+1)})]$  $\forall i \in 0..n$
else if $r \in R_{pro}$ then $t$
```

such that t is a negative offset from the date that p is expected to be completed (initially mapped to zero) and denotes how early r has to become available in order to complete p on schedule, q_i is the quantity of r that is requested over $(t_{2i}\ t_{2(i+1)})$, and $(t_{2i}\ t_{2(i+1)})$ is the period during which r needs to be reserved where t_i and t_{i+1} are also offsets from the date that p is expected to be completed and $\forall i \in 0..n\ \ t_{2i} < t_{2(i+1)}$.

Example 1: Consider the resource requirement specification *[(-5 -3 2) (-1 0 1)]* for a senior programmer. The specification requires 2 senior programmers for the first three months, no senior programmer for the fourth month (therefore the specification of this month is left out) and 1 senior programmer for the last month of development.

Example 2: Consider that to develop a debugger (a product) we need a simulator (also a product). Furthermore, suppose that the simulator has to be available at least three months before the debugger can be completed. This resource requirement can be specified as *-3*. The specification of the upper bound of the interval is redundant because the required resources which are products will remain available once they are produced for the first time.

Given the above formulation of resources and temporal resource constraints, we are now in a position to formalize the overall software project planning and scheduling problem. A software development organization is an organization that can carry out orders from several clients concurrently:

Definition 4: Software Development Organization

$G = \langle \{\Pi_i\}, \rho \rangle$

This definition implies that all projects within G compete for resources that are globally shared; ρ is of the form $\{\zeta_j\}$.

For every order, a project is created to undertake that order. However, our definition of a software project is recursive in that every inhouse order generated to meet a client order can be considered a project in its own rights. In the following definitions, let Π be a project in G and let p be the product of Π.

Definition 5: Software Project

Then $\Pi = \langle p,\Phi,\alpha,\beta,\theta \rangle$ s.t. p is the product of Π, Φ is the ordered set of feature requirements that p has to meet, α is the ordered set of elements that themselves are ordered sets of levels that each feature requirements of p can be met at (i.e. $\alpha=[x_i]$ where $x_i=\{y_j\}$ is the set of levels at which the i-th feature requirement of Φ can be met[5]), β is the set of desired levels of meeting each feature requirement (i.e. $\exists f$ 1-1 onto Φ) $f{:}\alpha \rightarrow \beta$ *s.t.* $\forall a \in \alpha\ \ f(a) \in a$, and θ is a set of heuristic evaluation functions (definition 6).

The above definition implies that each feature requirement Φ_i of a product can be satisfied at multiple levels l $(1 \le l \le \alpha_i)$ from which one, β_i, constitutes the desired level. Of course a schedule can be constructed to satisfy a feature requirement above its desired level but this might be undesirable or incur additional cost without satisfying any new objectives. Modeling the satisfaction of feature requirements at multiple levels allows us to study the consequences of relaxing the desired level of meeting a feature requirement or alternatively the consequences of reserving additional resources to assure that a feature requirement will be met at a level which is closer to what is desired.

[3]In software projects, these resources will remain available indefinitely once they are produced.

[4]Since our notation allows r_ζ to be used both as a function and as an object, we use the context of usage to determine whether it is a function or an object. More specifically, if it is followed by an argument which is enclosed in a parenthesis, then it is a function (the function value is a set). Otherwise, it is an object.

[5]y_j is a discrete variable that can assume only natural numbers. Furthermore, the elements of x_i are assumed to be organized in the increasing order.

Definition 6: Heuristic Evaluation Functions

Let θ be the set of heuristic evaluation functions of Π. Then $\theta=\theta_{uh}\cup\theta_{mh}$ (where θ_{uh} denotes the project dependent heuristics and θ_{mh} denotes the project independent heuristics[6]) and $\forall\theta_i\in\theta \quad \theta_i{:}A\rightarrow n \quad n\in N$, s.t. A is a new commitment at the commitment point Λ[7] and n is the rating of A.

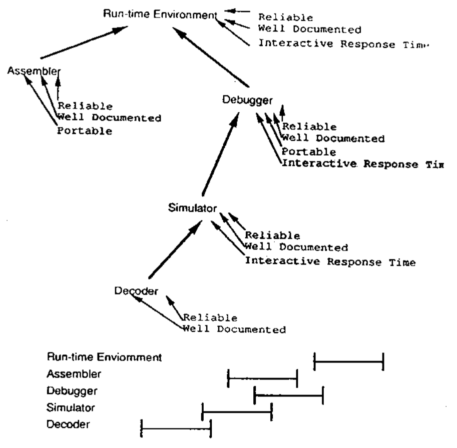

Figure 2-1: Production Dependency Graph of the TMS3020 Runtime Environment

Consider a software project to develop a runtime environment for a TMS3020 chip. Furthermore, consider that the environment has to provide interactive response time and integrate the debugging and assembly functions. Once the project is awarded, the developing organization produces a *production dependency graph* of the final and intermediate products that need to be developed along with the feature requirements of each product (figure 2-1). In the production dependency graph of figure 2-1, products are represented by nodes while activities (productions) are represented by directed arcs. For instance the activity of producing the debugger is denoted by the directed arc that is incident from the simulator on the debugger.

According to definition 5, the project to develop a run-time environment in figure 2-1 only includes the activity to produce the run-time environment once the assembler and the debugger are completed. In general, the production of every single product (e.g. a simulator in figure 2-1) is formalized as a project. A *complete project* of producing a run-time environment is constructed by recursively replacing each intermediate product by its complete project.

To illustrate the use of our specification language to specify the temporal/capacity knowledge of a problem consider again the example of developing a runtime environment for a TMS3020 chip (figure 2-1). Figure 2-2 depicts the basic topology of the resource requests of each product that has been specified in our specification language under a fixed process plan. Hardware description *(HD)* is the only shared primitive resource that is included in the resource request graph while the unshared resources consist of junior programmer *(JP)*, senior programmer *(SP)*, graphic generator *(GC)*, and lexical analyzer *(LA)*. The labels of the arcs that connect two nodes reflect the resource requirements (temporal/capacity constraints) of a product end of the arc for the resource end of the arc. For instance, *HD* is required one month into the production of assembler while to develop the run-time environment 2 senior programmers will be required for the first two months and only one senior programmer for the third month.

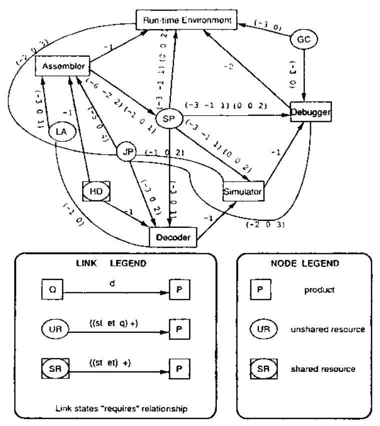

Figure 2-2: Resource Request Graph of the TMS3020 Runtime Environment

Each project can typically be implemented through many process plans. A process plan specifies how a product can be produced from its required resources but stops short of allocating resources and temporally instantiating that plan. Furthermore, the process plan of a product does not specify how its required resources are acquired or produced. A *complete* process plan of a product p can be constructed by recursively merging p with the *complete process plans* of all products that are required by p.

Among the set of resources required to produce a product under a process plan, some resources can be replaced by others. For instance, it might be possible to implement a process plan of a product by either 2 senior programmers and 3 junior programmers or 1 senior programmers and 6 junior programmers. The set of resources required under a process plan can be broken down to disjoint partitions υ such that each resource is in the same partition as all other resources that it is replacable with. For each partition, the user can specify multiple mixes of resource allocation requirements (e.g. two senior programmers and 3 junior programmers vs. one senior programmers and 6 junior programmers). We denote the set of these mixes for all partitions by τ.

Definition 7: Process Plan

Let Γ be a process plan of project Π which produces p. Then $\Gamma = <R_p,\upsilon,\tau>$ s.t. R_{p} is the names of resources required to implement Π, $p\notin R_p$, $\upsilon\subsetneq 2^{R_p}$ is a set of disjoint partitions of R_{p} denoting substitutable resources, $\tau\subsetneq[[\zeta_i]_j]$ (ζ is a resource request) is the range of a 1-1 onto $f{:}\upsilon\rightarrow\tau$ where $\forall a\in\upsilon \quad f(a)$ denotes the alternative resource allocation requirements of a and $(Card(a)=Card(\zeta) \quad \forall\zeta\in f(a))$.

Each process plan can be implemented through many schedules. A schedule is characterized by

1. determining which feature requirements (if any) have to be compromised (to avoid violating more important requirements such as missing a delivery deadline). We use ψ to denote the actual level at which each feature requirement of a product is met in a schedule.
2. committing to a set of resource requirements among all alternative sets of resource requirements for each set of substitutable resources. We let χ be the set of indices of the selected resource requirements.
3. committing to a set of resource allocations denoted by π to budget (allocate resources to) the process plan which the schedule implements; π is of the form $\{\zeta_j\}$.

The duration and start time of a schedule can be derived from the resource requirement specification of that schedule [19]. Therefore these parameters need not be specified as independent variables.

Definition 8: Schedule

[6]"Meeting the deadline is the most important feature requirement of a product under a rush mode" is a typical project independent heuristic that prefers the meeting of one feature requirement over the others if all can not be met together. In contrast, θ_{uh} includes the heuristics that are specific to the project being planned/scheduled and is specified with the problem.

[7]A *commitment point* refers to the plan/schedule that exists before it is revised by making a new commitment.

Let Λ be a schedule of Γ and p be the product of the project for which Γ is a process plan. Then $\Lambda = <\pi,\chi,\psi>$ s.t. π denotes the resource reservations of R and is the range of 1-1 onto h s.t. $h{:}R \rightarrow \pi$, χ is an ordered list of natural numbers where $\exists g$ 1-1 onto $g{:}\tau \rightarrow \chi$ s.t. $\forall a \in \tau \quad 0<g(a)<\mathrm{Card}(a)$ (a position within g(a)), and ψ is an ordered list of natural numbers that denotes the actual level of meeting each feature requirement (i.e. $\exists f$ 1-1 onto $f{:}\alpha \rightarrow \psi$ where $\forall a \in \alpha \quad f(a) \in a$.

Normally we are interested not only in the schedule that describes how p is produced from its required resources but also the schedule of all intermediate products that need to be produced in order to develop the required resources of p. The complete schedule of p, Λ_{p*}, can be defined as the union of the schedule of p and the *complete schedules* of all required products of p. This can be formally written as follows:

Definition 9: Complete Schedule

Let Λ_r denote the present schedule of producing a product r.
Then $\Lambda_{p*} = \Lambda_p \cup \Lambda_{q*}$ s.t. $\forall q \quad p \in ptc(q)$

Λ_{p*} can be constructed by starting from p working back recursively and including the schedules of all resource requirements of p that are of the type product.

Definition 10: Product-Transitive-Closure (ptc)

Let R and R_{pro} be as defined in definition 1 and ζ be a resource request. Then $\forall x \in R_{pro} \quad \forall y \in R \quad x \in ptc(y) \quad iff \quad y \in r_\zeta(x) \quad \vee$
$\exists w \in R_{pro} \quad (y \in r_\zeta(w) \quad \wedge \quad x \in ptc(w))$

The *scheduling problem* for a project consists of developing a schedule which is *consistent*. A project schedule Λ is consistent if and only if it meets the resource requirements that are necessary to satisfy the feature requirements of the product of that project (e.g. p)[8]. Since meeting the resource requirements of p is tied to meeting the resource requirements of the resources that p requires, then *consistency* needs to be measured across the *complete schedule* of p. A formal definition of the schedule of p is provided below:

Definition 11: Schedule Consistency

Let Λ be a schedule for project Π, p be the product of Π, and $n\text{-}th(x,i)$ return the i-th element of list x. Then Λ is *consistent* iff
$\forall q \quad p \in ptc(q) \; \forall r \in r_\zeta(q)$
for $x \in \tau$ s.t. $\exists \zeta \in x$ where $r_\zeta = r$
$n\text{-}th(x \; , \; g(x)) \subseteq h(r)$) (states that the allocation for r is at least equal to the request for r) s.t. τ is as in definition 7 and h,g are as in definition 8.

Although the goal of scheduling is to develop a consistent schedule, incremental scheduling in our framework can be characterized as a process that continuously refines a given schedule with the goal of achieving consistency. To guide this process, we need to measure the distance between the present schedule and the goal schedule during each problem solving iteration. The value that the evaluation function returns for a schedule can be interpreted as the *consistency-distance (cd)* of that schedule.

3. Cost/Benefit Model

This section describes an evaluation function for comparing alternative scheduling commitments, or evaluating part or all of a schedule. According to cost/benefit analysis, two alternatives can be compared on the basis of the *cost* to the project of budgeting each and also their *benefit* (degree to which each meets the requirements of the project). This implies that an alternative is preferred over another if it balances the satisfaction of a more important subset of resource requests with incurring a smaller cost overrun. The reason for using cost/benefit analysis to compare alternative scheduling commitments is that

1. labor and raw material cost of budgeting different alternatives are not the same.
2. client requirements that can be satisfied under different alternatives are not the same.

Intuitively, the cost of a resource request to a project is defined as the expenses that is incurred on the project in an effort to satisfy that request. The cost of satisfying a resource request is a non-negative quantity and is expressed in dollars. Cost of a resource request depends on both the time window over which the request is made and the capacity requested. The input for estimating the cost of an alternative consists of the marginal cost of acquiring only such primitive resources as manpower, off-the-shelf hardware, and off-the-shelf software.

Definition 12: Marginal cost of an unshared primitive resource

Let the marginal cost of acquiring q_r quantity of resource $p \in R_{Upri}$ over a period I when q_a of p is already allocated to all other requests be denoted by $MC(p,I,q_r,q_a)$.

In the case of the marginal cost of a shared resource, q_a is irrelevant and therefore should be dropped from the formula because allocation does not reduce the availability.

Benefit of a resource request to a project is a measure of the importance of that request to the project. This importance is measured in terms of the feature requirements of the project product p which will be met only if the resource request is satisfied. Intuitively, the input for estimating the benefit of a resource request includes

1. the relative importance of meeting feature requirements of p.
2. for every feature requirement that can be compromised, the resources that will be saved if that feature requirement is compromised.

The focus of discussion in the remaining of this section will be on the four central questions of cost/benefit analysis: how to measure the cost, the benefit, cost and benefit, and how to use the conclusions from tradeoff analysis to make alternative selection and feature requirement relaxation.

3.1. Cost Function Definitions

This section is to describe how to measure the cost of a resource request, a schedule, a process plan, and a software project from primitive cost data.

Intuitively, cost of a project schedule is measured by the sum of the cost of all resource requirements that the project schedule does not meet. First the set of all resource requests that are not satisfied by the schedule is constructed. For each resource that is requested then all requests for that resource are correlated and then unified (correlation and unification are described in separate sections below). Ultimately, the cost of correlated and unified resource requests are summed to calculate the schedule cost.

Definition 13: cost of satisfying a resource request ζ

```
if $r_\zeta \in R_{pro}$ then
    $C(r_\zeta,t)$
else if $r_\zeta \in R_{Spri}$ then
    $\sum^{(t_b \; t_e) \in \zeta} C(r_\zeta,[t_b \; t_e])$
else if $r_\zeta \in R_{Upri}$ then
    $\sum^{(t_b \; t_e \; q) \in \zeta} C(r_\zeta,[t_b \; t_e],q)$
```

Definition 14: cost of any number of shared primitive resource p over $[t_b \; t_e]$

$C(p,[t_b \; t_e],q) = MC(p,[t_b \; t_e])$

Definition 15: cost of q_r unshared primitive resource p over $[t_b \; t_e]$

let Ξ be the set of largest adjacent subintervals over $[t_b \; t_e]$ s.t. the aggregate quantity of p allocated over each subinterval is constant. Then

$C(p,[t_b \; t_e],q_r) = \sum^{I \in \Xi} MC(p,I,q_r,q_a)$ s.t. I, q_r, q_a denote duration, quantity requested, and quantity of p allocated so far respectively.

Let p be the product of a project Π, Γ^l be the l-th process plan of Π, $\Lambda^i_{\Gamma^l}$ be the i-th schedule of the l-th process plan of Π. Then

Definition 16: cost of producing a product p

Let $\Gamma(p)$ denote the set of all process plans of a product p. Then $C(p) = min[C(\Gamma^l) \quad \forall \Gamma^l \in \Gamma(p)]$

The cost of implementing a project is the same as the cost of developing the product of that project.

[8] In the scheduling literature, *consistency* refers to the fact that all reservations have to obey the temporal and capacity restrictions ρ of available resources.

Definition 17: cost of the *l*-th process plan of *p*

Let $\Lambda(\Gamma)$ denote the set of all schedules of Γ and Λ_{Γ}^{i} be the *i*-th schedule of a process plan Γ of a product.
Then $C(\Gamma^{l}) = min[C(\Lambda_{\Gamma^{l}})\ \forall\Lambda_{\Gamma^{l}} \in \Lambda(\Gamma^{l})]$.

Definition 18: cost of the *i*-th schedule Λ of *p*

Let Λ be a schedule for project Π, and *n-th(x,i)* return the *i*-th element of ordered list x. Then

$C(p,\Lambda) = \sum C(\zeta)\ \ \forall\zeta \in \{n\text{-}th(x\ ,\ g(x)) - h(r)$
$\forall r \in R,\ for\ x \in \tau\ s.t.\ \exists\zeta \in x$ where $r_{\zeta}=r\}$ s.t.
τ is as in definition 7 and *h,g* are as in definition 8.

The above definition shows that the cost of a schedule of a product is measured recursively in terms of its required resources. Moreover, since the above summation does not remove the duplicate resource requests, cost of a product or a shared primitive resource could be counted more than once. We will discuss how this problem can be fixed in the following subsection.

All cost functions exploit the *correlation* [19] between a set of unsatisfied resource requests to insure that the *cost of the same shared resource* will not be counted more than once. Resource requests are also transformed through *unification*. Unification involves translation of the requests for the same resource to utilize its marginal cost. The translation will result in reduction in the number of requests and also in minimization of the cost to satisfy them. The analysis of this algorithm can be found in [19].

3.2. Benefit Measurement

Benefit can be measured both locally and globally. Local measurement of benefit refers to ruling on the importance of making a scheduling commitment that benefits a product *p* by the agent who is responsible for *p*. In contrast, global measurement of benefit refers to ruling on the importance of making a scheduling commitment by an agent who is not responsible for *p*.

In general, the local authority to rule on benefit is provided when there is a fair idea about the usefulness of a scheduling commitment to the entire schedule, and also when there is not enough time to measure the global usefulness of a scheduling commitment. In contrast, benefit is measured globally when there is a great deal of uncertainty about the global usefulness of a local scheduling commitment and also when there is enough time to measure the global benefit.

3.2.1. Local Measurement of Benefit

The information needed to rule on the importance of a scheduling commitment can be broken to two parts: (1) the difference in the feature requirements of *p* that can be met before and after the commitment has been made, and (2) the penalty of not meeting each feature requirement of *p*.

Definition 19: benefit of a project

Let Π be a project and *p* be the product of that project. Then $B(\Pi)=B(p)=1$.

Definition 20: benefit of the *l*-th process plan of *p*

$B(p,\Gamma^{l}_{p}) = max[B(p,\Lambda)\ \ \forall\Lambda \in \Lambda(\Gamma^{l}_{p})]$

Definition 21: benefit of the *l*-th schedule of a process plan of *p*

Let Π be a project, *p* be the product of Π, Γ be a process plan of π, and Λ be a schedule of Γ. Furthermore, let

- Ω be the weight (penalty) of not meeting each Φ_i s.t. $0 \le \Omega_p^{i} < 1\ \ \forall i$, and
- $\sigma:\Phi \rightarrow X\ \ X \subseteq \{0,1\}$ be such that $f(\phi)=0$ iff Λ does not meet ϕ.

Then $B(p,\Lambda) = 1 - \sum^{\phi \in \Phi} \Omega_{\phi} \times \sigma(\phi)$.

The complexity of measuring the local benefit is $Card(\Phi) \times n$ where *n* is the number of distinct ways that the members of Φ can be satisfied under a fixed set of resource allocations.

Example 3: Consider the specification of penalties and the ability of two alternative process plans for developing the simulator in figure 2-1 to meet its feature requirements as depicted in table 3-1. The column headings of table 3-1 denote the feature requirements of the simulator and the row headings denote alternative process plans as well the penalty of not meeting each feature requirement.

Simulator	meet-the deadline	reliable	well documented	portable
production plan 1	X	X	X	
production plan 2		X	X	X
penalty	.3	1	.1	.1

Table 3-1: Input for the Measurement of Local Benefit

To be able to fill out the rows that correspond to process plans 1 and 2, the human expert needs to identify the resources and feature requirements of those resources (if they are products) that are needed to be present in order to meet each feature requirement of the simulator and then decide whether those resources and resource features are provided under the process plans that are being considered. Although human experts go through this process in order to decide which process plan to choose, they consider only the features and resources that they believe are significant. In accordance with this, we propose that σ be specified only if Φ is critical and there are multiple Γ.

If each row heading (a fixed allocation of resources) can produce multiple sets of feature requirement satisfaction alternatives, then definition 21 has to be extended to find the alternative that yields the highest benefit. For instance, if the expert attempts to specify that process plan 1 can meet either the delivery deadline or the reliability feature but not both, then two propagation paths need to be spawned at the simulator: one assuming that only reliability will be met and the other assuming that only delivery deadline will be met.

Definition 22: benefit of a resource request

Let Π be a project, *p* be the product of Π, Γ be a process plan of Π, Λ_1 be the present schedule of Γ, and ζ be a new resource allocation. Then $B(p,\zeta) = B(p,\Lambda_1) - B(p,\Lambda_2)$ s.t. Λ_2 is the revised schedule of Γ (the one resulted from adding ζ to Λ_1).

3.2.2. Global Measurement of Benefit

The main difference between global measurement of the benefit and the local measurement of the benefit is that to measure the global benefit the data about the satisfaction of each product feature has to be propagated until it reaches an agent who has the authority to make rulings on its importance. Intuitively, this process is as follows:

1. start from *p* and propagate the product features that are satisfied recursively until a stoppage point (e.g. product *q*) is reached.
2. list all possible alternative sets of product features that can be met.
3. choose one set by taking into consideration the preferences of the agent who has authority over *p*.

The complexity of measuring the global benefit is discussed in [19].

3.3. Tradeoff Between Cost and Benefit

Definition 23: Consistency-Distance

Let Λ be a schedule of Γ (a process plan of Π which produces *p*). Then $cd(p,\Lambda) = \frac{C(\Lambda)}{B(p,\Lambda)} \times w$ s.t. $0<w<1$ is a weight.

The specification of *w* has to be left to the agent who has the authority to make preference rulings over *p* since different agents use different scales in specifying the penalty of not meeting a product feature requirement.

4. Experiment Results

Our goal in conducting experiments was to study the feasibility of our cost/benefit approach. We compared the overall improvement in the schedule using our approach with the schedule that we started with in each case, and also analyzed the computational cost of evaluating a scheduling commitment (using our approach) in relation to other system components in a typical search-based scheduler.

Two groups of data were used to conduct these experiments. The data in each group sketched a multi-project organization that is engaged in the concurrent execution of three projects. The second group of data contained a greater degree of assembly and had a considerably larger search space. The data within each group was not randomly generated.

First, we conducted a total of 46 experiments and in each case we measured a number of parameters including the improvement in the quality of the final schedule in comparison to the quality of the seed schedule. This improvement was measured by comparing the *cd* of the seed schedule with that of the final schedule. In designing a seed schedule, our goal was to minimize the duration of each project at the cost of other resources. This implies that we considered "time" to be the most expensive resource. Moreover, in most experiments we assumed that the feature requirements are roughly equally important. As a result of this assumption, the choice of "which compromise to make next" were not transparent any longer during the scheduling.

Once we conducted all 46 experiments, we analyzed the results by measuring the correlation between the input and output parameters. In each case one or two input parameters were allowed to vary while all others were fixed. Among the output parameters, those that we believed were interesting were recorded and then the results were tabulated into a number of tables [19] such that each cell of a table averaged all experiments that satisfied its rows and column attributes. Below, we have included one of these tables. The budget overrun of individual resources were not accounted for since "cost" of the final schedule measured the combined effect of all of them. The use of an effective global measure in this case simplified the analysis.

Table 4-1 compares the effect of the two different groups of input on the output. The headings of the three middle columns *conflict analyzer, action manager,* and *progress analyzer* refer to the three components of NEGOPRO used for conducting incremental heuristic search. Progress analyzer has implemented a simplified version of the evaluation function which has been described and is used to measure the consistency of a revised schedule (intermediate solution) at the end of each incremental search step. Table 4-1 shows that by increasing the overall size and the assembly nature of the problem, the time that is spent in the conflict analyzer, action manager, and the progress analyzer all increase. However, the rate by which the computational complexity of action manager and progress analyzer increase is smaller than the rate by which the computational complexity of conflict analyzer increases. This suggests that the cost/benefit analysis is not a bottleneck and therefore may be coupled with other scheduling strategies as well. A thorough analysis of our data and our experimental results can be found in [19].

5. Conclusion

In this paper, we have presented an evaluation function for search-based approaches to scheduling that lends itself to cost/benefit analysis to compare alternative scheduling commitments during the search process. The function was described in the framework of software manufacturing problem domain. Our work extended existing search-based approaches by providing a domain independent evaluation function which can (1) measure the global impact of a commitment on the entire schedule, and (2) evaluate commitments in distributed scheduling environments. A major advantage of our evaluation function is that by following the methodology of defining cost and benefit functions, it can be used to evaluate scheduling commitments in a variety of manufacturing domains.

Group No	AVG # of Operators	Time in Conflict Analyzer	Time in Action Manager	Time in Progress Analyzer	Quality Improvement
Group1	6	1.2	0.7	0.1	490%
Group2	19	3.1	1.1	0.2	260%

Table 4-1: The effect of the size of the problem on the speed and quality of the final schedule (time in mins).

We implemented a program called NEGOPRO that uses our model to support the development and reactive revision of software plans/schedules. Our experimental results with NEGOPRO program indicate that our approach to schedule evaluation can be efficiently implemented, and does not constitute the bottleneck (from a computational standpoint) in comparison to other parts of a typical search-based scheduler.

6. Acknowledgements

We are greatful to Mark Fox from the center for integrated manufacturing decision systems at CMU, Charles Marshal at Digital Equipment, and Hamid Nabavi at Xerox for their valuable comments.

References

1. R. Banker, S. Datar, S. Kekre. "Relevant Costs, Congestion and Stochasticity in Production Environments". *Journal of Accounting and Economics* (1987).

2. Kent Bimson and Linda Burris. "Assisting Managers in Project Definitions". *IEEE Expert*, 2 (1989).

3. Barry Boehm. *Software Risk Management Tutorial.* TRW, 1988.

4. M. Fox and S. F. Smith. "ISIS - A Knowledge Based System for Factory Scheduling". *Expert Systems 1*, 1 (July 1984).

5. Les Gasser. "The Integration of Computing and Routine Work". *ACM Transactions on Office Information Systems* (July 1986), 205-225.

6. Abdel-Hamid Tarek. *Project Management Modeling.* Ph.D. Th., MIT, School of Business Management, 1984.

7. M. Garey and D. Johnson. *Computers and Interactibility.* Freeman, New York, 1979.

8. Richard Jullig, Wolfgang Polak, Peter Ladkin and Li-Mei Gilham. KBSA Project Management Assistant. Tech. Rept. RADC-TR-87-78, Kestrel Institute, 1987.

9. U. Karmakar, S. Kekre, S. Freeman. "Lot Sizing in Multi-machine Job Shops". *I.I.E. Transactions* (1985).

10. Keeney, R. L., and Raiffa H.. *Decisions with Multiple Objectives.* John Wiley and Sons, 1975.

11. H. Kerzner. *Project Managment: A systems Approach to Planning, Scheduling and Executing.* Van Nostrand Reinhold Company, 1984.

12. Rob Kling and Walt Scacchi. "Computing as Social Action: The social dynamics of computing in complex organizations". *Advancements in Computers*, 19 (July 1980), 249-327.

13. K. McKay, J. Buzacott, F. Safayeni. The Scheduler's Knowledge of Uncertainty: The Missing Link. Proceedings of IFIP Conference on Knowledge Based Production Management Systems, August, 1988.

14. Moder, Philips, and Davis. *Project Management with CPM, PERT & Precedence Diagramming.* Van Nostrand Reinhold Company, New York, 1983.

15. T. Morton, S. Lawrence, S. Rajagopolan, K. Kekre. "SCHED-STAR: A Price based Shop Scheduling Module". *Journal of Manufactiring and Operations Management*, 1 (1988), 131-181.

16. Peng Si Ow, Stephen Smith, and Alfred Thiriez. Reactive Plan Revision. Proceedings of American Association of Artificial Intelligence, Saint Paul, 1988, pp. 77-82.

17. N. Sadeh, M. Fox. Focus of Attention in Activity-Based Scheduler. Proceedings of the NASA Conference on Space Telerobotics, 1989.

18. A. Safavi and L. Gasser. Supporting the Negotiation During Planning/Scheduling, working paper. , 1990.

19. A. Safavi. Scheduling and Planning of Large-Scale Software Manufacturing Projects. Robotics Institute, C.M.U., in preparation, 1990.

20. A. Sathi, T. Morton, and S. Roth. "Callisto: An Intelligent Project Management System". *AI Magazine*, Winter (1986).

21. Arvind Sathi. *Cooperation Through Constraint Directed Negotiation: Study of Resource Reallocation Problems.* Ph.D. Th., Carnegie Mellon University, 1988.

22. Walt Scacchi. "Software Engineering: A Social Analysis Study". *IEEE Transacations on Software Engineering* (Jan. 1984), 45-60.

23. E. P. Sycara. *Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytical Method.* Ph.D. Th., Georgia Institute of Technology, 1987.

24. Whitmore, G.A., and Cavadias, G.S. "Experimental Determination of Community Preferences for Water Quality Alternatives". *Decisions Sciences* (1974), 614-631.

25. J. Zimmerman and M. Sovereign. *Quantitative Models for Production Management.* Englewood Cliffs, NJ, Prentice-Hall, 1974.