Criteria-Directed Task Scheduling [*] [†]

Thomas Wagner
Computer Science Department
University of Massachusetts
Amherst, MA 01003
Email: wagner@cs.umass.edu

Alan Garvey
Division of Math & Computer Science
Truman State University
Kirksville, MO 63501
Email: agarvey@truman.edu

Victor Lesser
Computer Science Department
University of Massachusetts
Amherst, MA 01003
Email: lesser@cs.umass.edu

UMass Computer Science Technical Report 1997-59

October 16, 1997

ABSTRACT

Scheduling complex problem solving tasks, where tasks are interrelated and there are multiple different ways to go about achieving a particular task, is an imprecise science and the justification for this lies soundly in the combinatorics of the scheduling problem. Intractable problems require approximate solutions. We have developed a new domain-independent approach to task scheduling called Design-to-Criteria that controls the combinatorics via a satisficing methodology and custom designs schedules to meet a particular client's goal criteria. In Design-to-Criteria, criteria directed focusing, approximation, and heuristics, in conjunction with soft goal criteria are used to make the scheduling problem tractable. We describe the interesting facets of the Design-to-Criteria approach and give examples of its power at reducing the complexity of the scheduling task while designing custom satisficing schedules.

KEYWORDS: ***real-time, task scheduling, heuristic, approximate, satisficing***

[*] A version of this paper appears in the Specail Scheduling Issue of the *International Journal of Approximate Reasoning*, 1998.

[†] This material is based upon work supported by the National Science Foundation under Grant No. IRI-9523419, the Department of the Navy, Office of the Chief of Naval Research, under Grant No. N00014-95-1-1198, and via a subcontract from Boeing Helicopter which is being supported by DARPA under the RaDEO program (contract number 70NANB6H0074). The content of the information does not necessarily reflect the position or the policy of the Government, National Science Foundation, or Boeing Helicopter and no official endorsement should be inferred.

1. Introduction

With the advent of open computing environments adaptability in software applications is critical. Since open environments are less predictable, applications must be able to adapt their processing to the available resources and the different goal criteria set by different clients. These requirements have given rise to the subdiscipline of *flexible computation* [13, 2] that researches methodologies, algorithms, and techniques for designing adaptive applications. An interesting and difficult scheduling problem arises in adaptive systems when there are multiple different ways to achieve tasks and the tasks are interdependent, i.e., the result of one subtask affects the performance, characteristics, or outcome of another subtask in quantifiable ways. The combinatorics possible from even simple interrelated tasks of this type are significant and the problem of scheduling a sequence of actions, given complex goal criteria and limited time, is intractable. Approximate scheduling methods are required.

We have developed a new domain independent flexible computation approach to task scheduling called *Design-to-Criteria*. The most distinguishing features of Design-to-Criteria are the ability to reason about the utility attribute trade-offs of different solutions based on different goal criteria, the ability to use these utility attribute trade-offs to focus every step of the scheduling process, and the ability to do these activities from a satisficing perspective. Satisficing with respect to the scheduling process itself enables the scheduler to produce results when computational combinatorics prevent an optimal solution. Satisficing with respect to meeting the goal criteria enables the scheduler to produce a result that adheres to the spirit of the goal when the criteria cannot be satisfied perfectly due to environmental and resource constraints.

We have framed this task scheduling problem in terms of a domain-independent representation framework called TÆMS (Task Analysis, Environment Modeling, and Simulation) [4, 5] that models a wide range of computational task structures. Our research focuses on a class of computational task structures where there are typically multiple different actions for performing a particular task, each action has different statistical performance characteristics, and uncertainty about the outcomes of actions is ubiquitous. For example, in the signal processing domain [15] there are multiple different techniques that can be used to process and identify signals; an approximate signal processing algorithm such as QSTFT (quantized short-time Fourier transform) [17] is inexpensive to compute but likely to produce interpretations that have significant uncertainty and there is a high probability that the interpretations will altogether miss certain types of signal sources. In contrast, a STFT (short-time Fourier transform) [18] is expensive to compute, but has very good quality and it is highly likely that all signal sources will be represented to some degree in the interpretation. This example is deliberately simple to illustrate a point. Consider a case where there are *many* different actions for achieving a particular task and any combination of the actions can be employed and possibly in any order. Now consider a hierarchy of such tasks where the tasks themselves are interrelated and constrained by deadlines and resource limits. The TÆMS framework models such problem solving processes. In TÆMS primitive actions, called *methods*, are modeled statistically via discrete probability distributions in three dimensions, quality, cost, and duration. Probability distributions are also associated with task interactions, called *NLEs* (non-local-effects), e.g., precedence constraints or advantageous soft relationships, and the effects of the interactions are reasoned about statistically.

A highly simplified conceptual example[1] of a TÆMS task structure for gathering auto purchase information via the Web is shown in Figure 1. The oval nodes are tasks and the square nodes are methods. The top-level task is to *Gather-Purchase-Data-on-Nissan-Maxima* and it has two subtasks *Gather-Reviews* and *Find-Invoice-Price-Data*. The top-level task accumulates quality according to the *sum_all() quality accumulation function* (qaf)[2] thus both of its subtasks must be performed to satisfy the objective. The *Gather-Reviews* task has two methods, query *Edmund's-Reviews* and query *Heraud's-Test-Drives*. These methods are governed by a *sum()* qaf thus the power-set of the methods minus the empty set may be

[1] The task structures actually emitted by the information gathering planner are too complex for example purposes.

[2] Qafs define how a given task is achieved through its subtasks or methods. The *sum_all()* qaf means that all of the subtasks must be performed and that the task's quality is a sum of the qualities achieved by its subtasks.

performed to achieve the tasks, i.e., *Edmund's* may be queried, *Heraud's* may be queried, or both may be queried. The *Find-Invoice-Price-Data* task has three subtasks, two of type method and one of type task, governed by the *max()* qaf which is analogous to an OR relationship. Note the decomposition of the obtain invoice via *AutoSite* task into two methods, one that locates the URL and one that issues the query. The *enables* NLE between the URL finding method and the query method, in conjunction with the low quality associated with the URL finding method, indicate that finding the URL is necessary for task achievement but that it contributes very little to achieving the task relative to the method that actually obtains the pricing report. We discuss TÆMS in more detail in Section 2 and return to this example in Section 3.

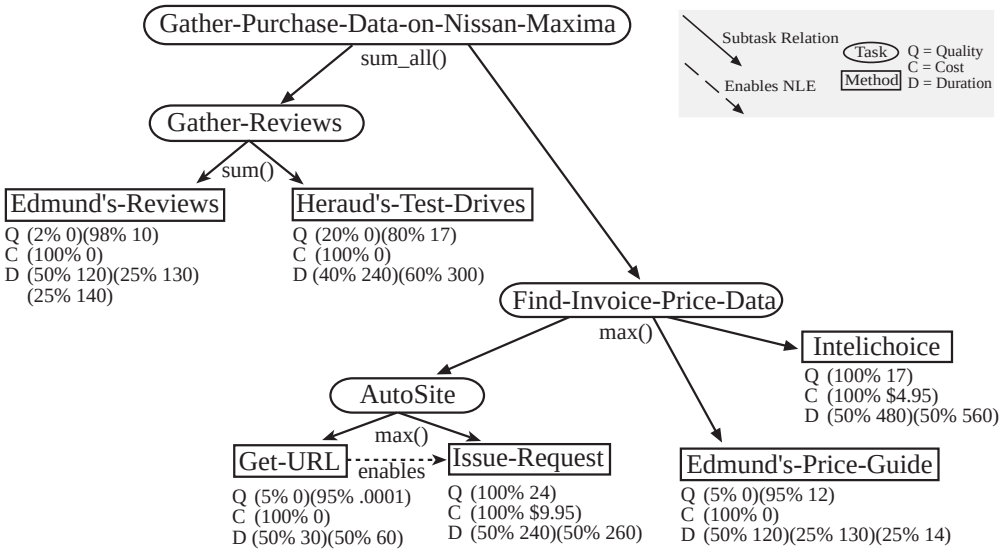


Figure 1. TÆMS Task Structure for Gathering Auto Purchase Information

Scheduling problem solving activities modeled in the TÆMS language has three major requirements: 1) to find a sequence of actions to achieve the high-level task, 2) to find the sequence of actions in soft real-time, 3) to find the sequence to meet the dynamic goal criteria, i.e., different cost, quality, duration, and certainty requirements, of different clients. TÆMS models multiple approaches for achieving tasks along with the quality, cost, and duration characteristics of the primitive actions, specifically to enable TÆMS clients to reason about the trade-offs of different courses of action. In other words, for a given TÆMS task model, there are multiple approaches for achieving the high-level task and each approach has different quality, cost, duration, and certainty characteristics. In contrast to classic scheduling problems, the scheduling objective is not to find some way to accomplish the task, but to find the approach that *best* suits a particular client's quality, cost, duration, and certainty needs. Consider the task of gathering information via the highly uncertain WWW to support a decision about the purchase of a statistical analysis software package. Certain clients may prefer a risky information gathering plan that has a potentially high pay-off in terms of information gathered, but also has a high probability of failure. Other, more risk averse clients might prefer a course of action that results in a lower pay-off in exchange for more certainty about the pay-off and a lower probability of failure. The fundamental premise of our work is that *the goodness of a particular solution is entirely dependent on a particular client's complex objectives* and that *different client's have varying objectives.* Thus the scheduling process must not only consider the attribute trade-offs of different solutions, but must also do so dynamically. Furthermore, the scheduling process must be efficient as the application domain involves agents acting in the world in real-time. Because of the inherent uncertainty in the domain, where actions may fail or have unexpected results, scheduling activities are typically interleaved with planning and execution. Thus scheduler inefficiencies are multiplied many times during a problem solving instance.

We will go into greater detail in Section 3, but the $\omega(2^n)$ and $o(n^n)$ combinatorics of our scheduling problem precludes using exhaustive search techniques for finding optimal schedules. Furthermore, the deadline and resource constraints on tasks, plus the existence of complex task interrelationships, prevent the use of focused optimal search algorithms like $A^\star$. Design-to-Criteria copes with these explosive combinatorics by satisficing with respect to the goal criteria and with respect to searching the solution space. This satisficing dualism translates into four different techniques that Design-to-Criteria uses to reduce the search space and make the scheduling problem tractable:

Criteria-Directed Focusing The client's goal criteria is not simply used to select the "best" schedule for execution, but is also leveraged to focus all processing activities on producing solutions and partial solutions that are most likely to meet the trade-offs and limits/thresholds defined by the criteria. This is achieved by creating and identifying partial solutions that seem likely to meet the criteria and concentrating further development on these classes of partial solutions, pruning or ignoring other partial solutions that are deemed least probable to lead to "good" solutions.

Approximation Schedule approximations, called *alternatives*, are used to provide an inexpensive, but coarse, overview of the schedule solution space. Alternatives contain a set of unordered actions that can be scheduled (ordered) to achieve a particular task along with estimates for the quality, cost, and duration distributions that may result from scheduling the actions. Alternatives are inexpensive to compute as the complex task interactions are only partially considered and ordering, resource, and other constraints are ignored. The alternative abstraction space is used in conjunction with criteria directed focusing to build schedules from alternatives that are most likely to lead to good schedules.

Heuristic Decision Making We have focused on the high order complexity of our scheduling problem as a whole, but the action ordering scheduling problem suffers from similar combinatorics. Given a set of n actions to perform, there are $n!$ orderings that must be considered and the $O(n!)$ expense is non-trivial.

We cope with this complexity using a group of heuristics for action ordering. The heuristics take into consideration task interactions, attempting to take advantage of positive interactions while avoiding negative interactions. They also consider resource limits, individual action deadlines, task deadlines, commitments made with other problem solving agents, and other constraints. The heuristic algorithm reduces the $O(n!)$ action ordering problem to low-order polynomial levels in the worst case.

Heuristic Error Correction The use of approximation and heuristic decision making has a price – it is possible to create schedules that do not achieve the high-level task, or, achieve the high-level task but do not live up to quality, cost, duration, or certainty expectations set by the estimates contained in the alternatives. This can be caused by an overconstrained problem, but also by complex task interactions that are glossed over by the alternative approximation and not considered by the action ordering heuristics. A secondary set of improvement [28] heuristics act as a safety net to catch the errors that are correctable. Again, this problem is potentially computationally expensive as the required fix may be achievable by any combination of the actions in the task structure and it is impossible to ascertain if a hypothetical fix will generate the desired result until it is fully scheduled. Thus this aspect of the scheduling algorithm is also heuristic and relies on abstraction and criteria directed focusing to reduce the complexity.

Design-to-Criteria thus copes with computational complexity by using the client goal criteria to focus processing, reasoning with schedule approximations rather than complete schedules, and using a heuristic, rather than exhaustive, scheduling approach. A high-level view of the Design-to-Criteria algorithm is shown in Figure 2. This methodology is effective because several aspects of the scheduling problem are soft and amenable to a satisficing approach. For example, the client goal specification mechanism, discussed in detail in Section 3.1, expresses soft client objectives or soft constraints. Solutions do not need to meet absolute requirements because clients cannot know *a priori* what types of solutions are

possible for a given task structure due to the combinatorics. Hard constraints exist in TÆMS but they originate from commitments entered into with other problem solvers and from the tasks themselves. Similarly, soft task interactions also represent soft constraints that can be relaxed, i.e., they can be leveraged or not depending on the situation. Finally, though the TÆMS scheduling problem is more complex than many traditional scheduling problems because of its representation of multiple approaches for task achievement, it is also more flexible. If we view the scheduling activity as a search process, typically there is a neighborhood of solutions that will meet the client's goal criteria and the lack of exhaustive search, i.e., search by focused processing and approximation does not necessitate scheduling failure. We provide empirical examples of this characteristic of our scheduling problem in Section 4.

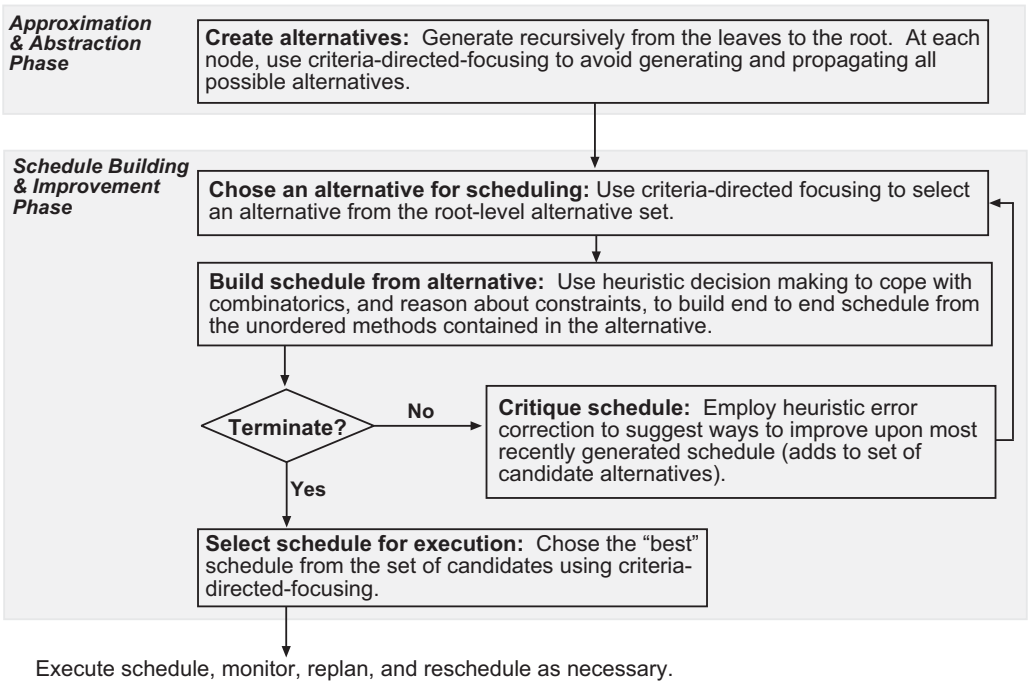


Figure 2. High-Level Control-Flow View of Design-to-Criteria

This work falls into the general area of flexible computation [13], but differs from most flexible computation approaches in its use of multiple actions for task achievement (one exception is [14]), in its first class treatment of uncertainty, and in its ability to use uncertainty information in the selection of methods for execution. Much work in flexible computation makes use of *anytime algorithms* [2, 20, 24], algorithms that always have an answer at hand and produce higher quality results as they are given more time, up to a threshold. Our multiple methods approach can model any activity, including anytime algorithms, that can be characterized statistically and we place no constraints on the statistical behavior of the activities in question. In our work, uncertainty is a first class concept that both appears in the statistical descriptions of the available methods and is propagated and related as schedules and schedule approximations are generated. Unlike most work in anytime algorithms that focuses on the propagation of uncertainty [26], we can also include uncertainty and uncertainty reduction in the goal criteria and focus work on reducing uncertainty when important to the client. This ability stems from our task model's representation of alternative ways to perform various tasks. Because multiple methods often exist to perform tasks, we can reason about the quality, cost, duration, and uncertainty trade-offs of different actions when determining which actions to perform, achieving the best possible overall results.

In Section 2 we describe the TÆMS task model in more detail and in Section 3 we present the client criteria specification metaphor and the Design-to-Criteria algorithm. Examples of Design-to-Criteria in action are provided in Section 4 and future work is discussed in Section 5.

2. TÆMS Task Models

The TÆMS task modeling framework is used to describe and reason about complex problem solving processes. TÆMS models are abstractions of problem solving processes; they represent major tasks and major decision points, interactions between tasks, and resource constraints but they do not describe the intimate details of each primitive action. Graphically the model is a tree where interior nodes, called *tasks*, denote abstract high-level problem solving activities and leaves, *methods*, represent executable actions. Alternative approaches for performing tasks are represented explicitly in TÆMS and all methods are statistically characterized in three dimensions: quality, cost and duration. Quality is a deliberately abstract domain-independent concept that describes the contribution of a particular action to the overall problem solving objective. Thus, different applications have different notions of what corresponds to model quality. Duration describes the amount of time that a method will take to execute. Cost describes the financial or opportunity cost inherent in performing the action modeled by the method. The statistical characteristics of the three dimensions are described via discrete probability distributions associated with each method. These distributions represent *a priori* expectations about the performance characteristics of actions. It is important to note that in most TÆMS applications, and with the Design-to-Criteria scheduling approach, these expectations need not be precise specifications. Scheduling is usually interleaved with execution, monitoring, and planning; the distribution mechanism serves as a vehicle for clients (planners, other humans, other systems) to express expectations that are used by the scheduling algorithm to consider trade-offs of different possible courses of action, in much the same way that a human problem solver would use expectations to make choices. When expectations differ from results, monitoring in conjunction with performance envelopes trigger replanning and rescheduling to navigate through the areas in which expectations did not hold true. Expectations may be improved over time through learning or obtained *a priori* by off-line learning. Even when expectations are perfectly in tune with prior problem solving episodes, the environments in which these systems operate are dynamic and unpredictable. The fundamental premise is that when choosing between alternative ways to solve a problem, reasoning from imprecise expectations that are at least in the "ballpark" is more effective than choosing blindly. TÆMS is a framework for modeling complex and inherently approximate problem solving activities; no TÆMS based control component relies wholly on the accuracy of the initial quality, cost, and duration distributions associated with TÆMS methods.

TÆMS models are the grounding element and medium of exchange for Design-to-Time [11, 12] and Design-to-Criteria scheduling [22, 23] and multi-agent [3] coordination research, and are being used in Cooperative-Information-Gathering [19, 7, 16], collaborative distributed design [6], distributed situation assessment [1], and surviveable systems [21] funded research projects. TÆMS diverges from traditional hierarchical representations in that different alternatives for achieving a task are expressed explicitly and reasoning about trade-offs is a first class activity. The overall objective when working in TÆMS is to achieve quality for the task structure root, *task group*, or set of task groups, all of which are synonymous with achieving the high-level task.[3] As with most hierarchical representations the high-level task is achieved by achieving some combination of its subtasks – quality achievement is a ubiquitous goal. High-level TÆMS tasks accumulate quality from their subtasks, which get quality from their subtasks recursively until the methods are reached, according to *quality accumulation functions* (qaf). Qafs are approximations that model how utilities are calculated and propagated in the problem solving process described by the model. The primary TÆMS qafs are `max()`, which is somewhat analogous to a logical OR, `min()`, comparable to logical AND, and `sum()` where any member of the power set [4] of the subtasks may be executed to achieve the task.

Hard and soft interactions between tasks, called *NLEs* (non-local effects), are also represented in

[3] The term *task group* is used to denote a set of tasks that are related hierarchically and via non-local effects. Tasks are not joined under a task group if they do not have interactions and are not related hierarchically. A TÆMS model may be composed of several task groups that are joined under a special *meta* root. In this case the overall objective is to achieve the meta root via the individual task groups, rather than to achieve a single task group via its subtasks.

[4] The number of ways to accumulate quality under the `sum()` qaf is the power-set of its subtasks minus the empty set.

TÆMS and reasoned about during scheduling and coordination. Hard task interactions delineate hard precedence constraints and model situations where the output of one computational activity is required for another computational activity to commence. Soft task interactions denote situations where the result of one computational activity can *facilitate* or *hinder* another computational activity. For example, in the trip-planning domain, if multiple agents need to ascertain the current temperature in Tampa and one agent acquires the information and communicates it to the other agents, they will benefit from the work done by the first agent and this will result in decreased duration for their problem solving activities. The effects of soft interactions are quantified via probability distributions that describe the effect of the interaction on the quality, cost, duration, quality certainty, cost certainty, or duration certainty of the tasks in question. Although soft interactions are "optional," representing soft constraints, they cannot be ignored by the scheduling process as leveraging soft interactions can greatly affect the characteristics of the schedules produced. A complete description of TÆMS is beyond the scope of this paper, however, further background information will be provided where necessary.

3. Design-to-Criteria Scheduling

Design-to-Criteria scheduling is the process of finding a satisficing course of action for a complex problem solving activity represented as a TÆMS task structure. The central objective is to cope with the combinatorial explosion of possibilities while reasoning about quality, cost, and duration criteria such as hard or soft limit/threshold requirements for each dimension and factors describing the relative importance of each dimension. Scheduler clients specify the design criteria and the scheduler *designs* a schedule to best meet the criteria, if possible given the task model. To illustrate this concept, a set of satisficing schedules produced by the Design-to-Criteria scheduler, for the Auto Purchase task structure (Figure 1), using four different sets of criteria is shown in Figure 3. Schedule A is constructed for a client interested in a fast, free, solution with any non-zero quality. Schedule B suits a client who wants a timely and free solution, but wants less uncertainty about the expected quality of the results. Schedule C is constructed for a user interested in a good quality, free, solution that can be obtained while she goes for a cup of coffee. Schedule D is generated to meet the criteria of a fourth individual who is willing to pay and wait for a high-quality response.

3.1. The Criteria Specification Metaphor

At the heart of the Design-to-Criteria paradigm is the ability to determine how well a particular schedule, or schedule abstraction (alternative), fits a set of design criteria. The process of measuring the "goodness" of schedules or alternatives and determining which items are best is called *evaluation*. Evaluation is used to focus alternative creation when the alternative sets are too large and the scheduling problem becomes intractable. It is also used to determine what alternatives to turn into schedules and to decide which completed schedule best satisfices to meet the criteria

The evaluation functions operate to determine a principled measurement of utility based on *relativity* and *proportionality*. Relativity is important because the objective is to make satisficing choices and the goodness of one option is relative to the other possible options. Proportionality is a major concern because we do not want different quality, cost, and duration scales to skew the evaluation mechanism and because the client's criteria is described in a relative/proportionalistic fashion. The evaluation functions are paired with a criteria specification metaphor, called *importance sliders*. The slider metaphor enables clients to define the relative importance of quality, cost, and duration with respect to four classes of concerns: raw goodness, thresholds and limits, certainty, and certainty thresholds.

The objective of the evaluation approach is to translate a client's needs into choosing the course of action that best meets the criteria. Clients are good at expressing and reasoning about the *relative* importance of quality, cost, and duration, but they are less good at assigning particular values that denote goodness. Thus, our evaluation approach operates on the conceptual notion of *importance sliders*

Schedule A: Fast and Free

Edmund's-Reviews	Edmund's-Price-Guide

Q (~0% 0)(5% 10)(2% 12)(93% 22)
C (100% 0)
D (25% 240)(25% 250)(31% 260)(12% 270)(6% 280)
Expected Q: 21 Q Certainty: 93%
Expected C: 0 C Certainty: 100%
Expected D: 255 seconds D Certainty: 50%

Schedule B: High Quality Certainty, Moderate Cost

Edmund's-Reviews	Intelichoice

Q (2% 17)(98% 27)
C (100% $4.95)
D (25% 600)(12% 620)(31% 680)(19% 700)
Expected Q: 26 Q Certainty: 98%
Expected C: $4.95 C Certainty: 100%
Expected D: 647 seconds D Certainty: 50%

Schedule C: Good Quality, Moderate Cost, Slow

Edmund's-Reviews	Heraud's-Test-Drives	Intelichoice

Q (~0% 17)(20% 27)(2% 34)(78% 44)
C (100% $4.95)
D (20% 840)(19% 900)(31% 920)(19% 980)(11% 1000)
Expected Q: 40 Q Certainty: 78%
Expected C: $4.95 C Certainty: 100%
Expected D: 920 seconds D Certainty: 70%

Schedule D: High Quality, High Cost, Moderate Duration

Edmund's-Reviews	Heraud's-Test-Drives	Get-AutoSite-URL	Issue-AutoSite-Request

Q (1% 0)(4% 27)(19% 34)(2% 41)(74% 51)
C (100% $9.95)
D (20% 630)(31% 690)(24% 720)(19% 740)(6% 760)
Expected Q: 46 Q Certainty: 74%
Expected C: $9.95 C Certainty: 100%
Expected D: 698 seconds D Certainty: 51%

Figure 3. Four Satisficing Schedules

that clients "set" for each dimension in the criteria set. The importance sliders, which take on percentage values from 0 to 100, describe the relative importance of each of dimension in a domain independent fashion. Using the sliders, client applications or users can express the notions like "quality is twice as important as cost and duration is half as important" or "quality and duration are equally important but cost is no issue."

While we have introduced sliders in a general sense there are actually five sets of sliders used in the criteria specification process, some of which are accompanied by absolute requirements in the form of thresholds or limits. We should note that the sliders take on percentage values, with the constraint that each bank's sliders sum to 100%, purely for semantic reasons. The entire evaluation approach will work with any range of values and without the 100% sum constraint. The slider sets, shown in Figure 4, are:

Raw Goodness This slider set contains sliders for each dimension, quality, cost, and duration. Its purpose is to describe the relative importance of each dimension. For example, setting quality to 50% and cost and duration to 25% expresses the notion that quality is twice as important as each of the other dimensions and that it should weigh twice as heavily as each when evaluating schedules or alternatives.

Threshold and Limits This slider set also contains sliders for each dimension, however, in this set each slider is paired with a value, or a gradual utility function, that denotes the minimum desired threshold for the quality dimension or the maximum limit for cost or duration. The separation of limits and thresholds from overall goodness allows clients to specify concepts like "Cost, quality and duration are equally important in general, but schedules whose cost is under my limit are particularly valuable."

Where utility functions are used instead of values to delineate changes in utility, the functions describe the range in which utility begins to increase as quality increases, or the range where utility decreases as cost or duration increase. The utility function specification lets clients express notions like "Overall quality and cost are equally important and I want a solution in five minutes, but I'll grudgingly wait up to ten minutes for a high-quality solution."

Note that the limits and thresholds describe quantities that schedules or alternatives must exceed in order to get points from this set of sliders, i.e., schedules that fail to beat thresholds and limits may still be returned for execution if they best satisfice over the criteria set as a whole. The issue

of satisficing with respect to hard constraints is beyond the scope of this paper but the solution lies in negotiation, see Section 5.

Certainty This slider set defines how important reducing uncertainty in each dimension is relative to the other dimensions. In particular applications it may be more desirable to pick a slower, more costly schedule that returns lower expected quality because the certainty about the resulting quality is very high. When reducing uncertainty is important to the client, the scheduler may schedule multiple alternatives for achieving a particular task, using a form of redundancy [23] to increase the probability that a particular result will be generated.

Certainty Thresholds This bank is analogous to the thresholds/limits bank above except that this bank focuses on the uncertainty associated with quality, cost, and duration. Schedules or alternatives whose certainty in a particular dimension meet or exceed the defined threshold are preferred. This enables clients to expression notions like "certainty in the quality dimension is not important as long as the schedule is at least 80% likely to produce the expected quality value or one better," as opposed to raw certainty objectives like "certainty in the quality dimension is important." As with the thresholds/limits sliders, the thresholds can be gradual, rather than a single value, and specified via a function or curve.

Meta This slider set relates the importance of the four previous slider sets. This separation allows clients to focus on relating quality, cost and duration with each other in each of the cases above, then to "step back" and decide how important each of the different aspects are relative to each other.

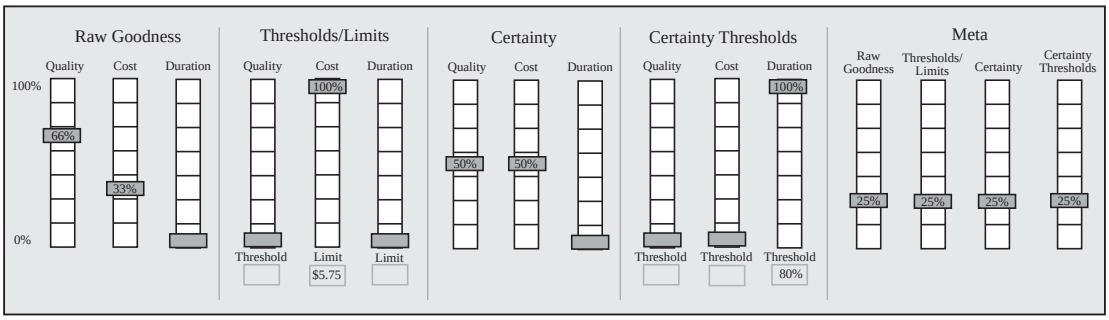


Figure 4. A Slider Set Describing Particular Criteria

In the example slider set, shown in Figure 4, quality is the most important general factor with cost being one half as important and duration being not important at all. In terms of thresholds, quality and duration have none, but schedules whose cost is below $5.75 are preferred. Schedules whose expected quality and cost values are more certain are also preferred and uncertainty about duration is not an issue as long as schedules meet or exceed the 80% duration certainty threshold. Relating the four sets of criteria together, they are all equally (25%) important and thus all contribute equally to the overall ranking. Mapping this example to the real world, this could describe the criteria of an individual performing research on the web who does not need the information in a timely fashion, has limited funds in his or her pocket, wants good quality information, but also wants to be certain of the cost and quality of the proposed solution before committing to a course of action, and is scheduling activities later in the day based on the expected search duration.

3.2. Mapping the Criteria Specification to Utility

After defining the slider sets, the problem then becomes how to use the criteria to produce results that match expectations. When determining schedule or alternative "goodness," alternatives or schedules are rated using the relative importances expressed on the sliders. We associate a rating component with each of the slider banks, excluding the meta bank, and then combine them according to the relative weights

expressed in the meta slider bank. The omnipresent themes in the rating calculations are relativity and proportionality.

In general, we calculate the rating component for a given slider bank by calculating subcomponents for each dimension: quality, cost, and duration. Each dimension's subcomponent is computed by looping over the set of items to be evaluated and normalizing each item's expected value or expected probability (in the uncertainty case) for that particular dimension, and then multiplying the result by the relative importance as expressed in the slider. It is crucial to normalize the values to a common scale so that in domains where one dimension, say quality, is exponentially larger than the others it does not dominate the ratings disproportionately. Scaling based on the observed minimum and maximum values for a given dimension is similarly important. With the exception of the threshold/limit case we are interested in relative goodness between alternatives or schedules. By using minima and maxima that are derived from the set of items being rated, we automatically scale the grain size to define relative differences in the items. For example, say Schedule A has expected quality of 4.7, Schedule B has expected quality of 4.2, and Schedule C has expected quality of 4.0. In absolute numerical terms Schedule A is "a little" better than both B and C. However, in relative terms, Schedule A is by far the best of the possible schedules. The notion of relative scaling will become more clear from the equations that follow.

We calculate the rating component for the first slider bank, that describes the raw goodness of a particular dimension, as follows:

1. Find the min and max expected values for quality, cost, and duration that occur in the set of schedules or alternatives being rated.
2. Loop over the set of alternatives or schedules to be rated and calculate the raw goodness rating for each by calculating the quality, cost, and duration subcomponents as follows in Steps 3 and 4.
3. Let $this.eq$ denote the expected quality value of the alternative or schedule under consideration. Its quality subcomponent is a function of the the percentage of quality achieved by $this.eq$ relative to the min and max, min_q and max_q, quality values of the set of items being rated, scaled by the raw goodness quality slider, RG_slider_q and the total number of points in the raw goodness bank.

$$rating_q = \frac{(this.eq - min_q)}{max_q - min_q} * \frac{RG\_slider_q}{\sum_{i=q}^{d,c} RG\_slider_i}$$

4. Duration is different than quality as greater duration is generally less preferable. Whereas with the quality related equation, achieving the best quality of all items in the set should bring the highest reward, in this case, achieving the least duration of all items in the set should bring the highest reward. Cost is like duration in that lower cost is better.

$$rating_d = \frac{(max_d - this.ed)}{max_d - min_d} * \frac{RG\_slider_d}{\sum_{i=q}^{d,c} RG\_slider_i}$$

$$rating_c = \frac{(max_c - this.ec)}{max_c - min_c} * \frac{RG\_slider_c}{\sum_{i=q}^{d,c} RG\_slider_i}$$

5. The quality, duration, and cost subcomponents are then summed to obtain the aggregate raw goodness rating component.

The threshold or limit rating component is likewise composed of three subcomponents that are simple to compute: quality above the specified threshold, and cost and duration below the specified limits, are rewarded according to the relative settings of the quality, cost, and duration sliders. Beating a threshold or a limit is rewarded the same regardless of how well a particular schedule or alternative beats the threshold or limit. Originally, the ratings were based on the distance between the expected value for a particular dimension and its limit or threshold. However, this approach leads to rewards for high

relative quality, and low relative cost and duration, from both the threshold/limit bank and from the raw goodness bank and the results were not in keeping with the semantics of the sliders. If a gradual utility function is used in lieu of a single threshold/limit value, the reward is scaled by the utility percent associated with the particular value.

The ratings for the certainty related slider banks are computed in a fashion similar to the raw-goodness and threshold/limit banks, though the focus is on certainty about values rather than the values themselves. After computing the raw goodness, threshold/limit, certainty, and certainty threshold rating components, the alternate or schedule rating is computed by weighting the rating components according to the relations specified by the meta sliders. The full details are presented in [22].

As the evaluation mechanism is used to reduce the computational work required by the scheduling process, it is important to realize that the above components are inexpensive to compute. The process of finding the minima and maxima and calculating the sub components is $O(n)$, where n is the number of items being rated. Additionally, the constants involved are very small as most of the values used in the computations are computed elsewhere and stored. Even the cost of sorting the items after they are rated, $O(nlogn)$, or selecting which m items of an unordered set of n to keep, $O(m * n)$ where $m < n$, is easily dominated by other factors in the scheduling process.

3.3. The Design-to-Criteria Approach for Managing Complexity

Design-to-Criteria scheduling requires a sophisticated heuristic approach because of the scheduling task's inherent computational complexity. To understand the complexity and get a feel for the scheduling process, consider a task structure only a single level deep, where a single task has m children that are methods and it accumulates quality according to the `sum()` qaf. In this case, there are $2^m - 1$ unordered sets of methods that can be used to achieve the parent task, and within each set of n methods, $n!$ possible orderings of methods in the schedule. In general, the upper-bound on the number of possible schedules for a TÆMS task structure containing m methods is given in Equation 1. Clearly, for any significant task structure the brute-strength approach of generating all possible schedules is infeasible.

$$\sum_{i=0}^{m} \binom{m}{i} i! \tag{1}$$

The Design-to-Criteria algorithm is composed of several discrete stages, Figure 2, each designed to assist in complexity reduction. The breakdown in stages correlates to two different sources of complexity contained in the summation above. The first source of complexity, $O(2^m)$ where m is the number of methods in the TÆMS task structure, is driven by the number of unordered method sets that can achieve the high-level objective. The second source of complexity is caused by the number of possible schedules that can be created for each unordered method set – $O(n!)$[5] where n is the number of methods in a given set.

We partly cope with the $O(n!)$ class of computational complexity by using the aforementioned schedule abstraction, the *alternative*. Alternatives contain sets of unordered methods that can be ordered to form a schedule and an estimate of the quality, cost, and duration distributions that may result from building the schedule. Alternatives are associated with all task nodes in the TÆMS task structure. Alternatives for tasks closer to the root are combinations of the alternatives associated with the subtasks; the subs' alternatives are combined according to the qaf. Alternatives are built bottom-up from the leaves to the root. A subset of the alternatives for the root of the task structure are turned into schedules to perform the high level task by achieving the lower level tasks. In a sense, alternatives of the interior nodes represent partial unordered schedules – they describe the actions that can be performed, i.e., methods, to obtain quality for the task node with which they are associated.

[5] The complexity of the action ordering task in TÆMS is actually $O((m*n)*(m!))$, where m is the number of actions to order and n is the number of nodes in the TÆMS task structure. The added factor is generated by the possibility of task interactions. When adding each method to the schedule, the entire task structure may have to be processed to propagate the non-local-effects and compute the effects of task interactions.

Alternatives contain two sets of estimates for the quality, cost, and duration distributions that are likely to result from scheduling the unordered methods. The *base-line* set of distributions is computed by ignoring all task interactions and the *potential* distribution set is computed by assuming that all positive task interactions are achieved in scheduling and all negative interactions are avoided. Note that the estimates are rough as the complex task interactions are glossed over and the other scheduling constraints, e.g., deadlines, resource limits, etc., are ignored. The estimates are currently related via an averaging technique, however, comparing the potential distribution set to the base-line set may yield useful information and is an area of future exploration.

The alternative abstraction defers the $O(n!)$ ordering component of the complexity problem until schedule time when orderings are imposed and the heuristic method ordering functions are used to reduce the $O(n!)$ complexity. However, the alternative abstraction does not address the $O(2^m)$ complexity factor which is driven by the number of possible method sets that can achieve the high-level objective. This complexity is handled dynamically during the alternative generation process by focusing processing on alternatives that best satisfice to meet the client's goal criteria. There are three different classes of combinatorial explosions that can occur during the leaf-to-root alternative generation process:

Build-Up Gradual complexity build-up occurs when a task node has a typical number of child nodes, each of which has many alternatives, and the children are joined under the `sum()` qaf.

Complexity of this type is controlled by pruning each alternative set post-hoc. In other words, the gradual build-up is eliminated by pruning the alternative sets of the child nodes after they are generated to keep the number of alternatives, at each step, within a reasonable bounds. The alternatives are pruned according to the quality, cost, duration, and certainty criteria specified by the scheduler client and the estimates associated with the alternative. The caveat is that the quality, cost, and duration distribution estimates may be inaccurate depending on the interactions between tasks. This is partly addressed by the *improvement* heuristics touched-on later.

Instant Instant complexity problems occur when a task node has a large number of child nodes, each with few alternatives, and the children are joined under the `sum()` qaf.

Complexity of this type is predictable by a simple look-a-head operation. In this case, we cannot first generate the alternative set and then prune because the actual number of alternatives that would be generated is too large. Instead, we must heuristically generate a set of alternatives that characterizes the set of possible alternatives with an eye towards the client's quality, cost, and duration criteria.[6] The details of this operation are current research. Note that if the number of child nodes is moderate, we can generate the alternative set and prune if necessary, as described above, because the exponential factor $O(2^m)$ still translates into a manageable number of alternatives.

Combination Combinations of instant and build-up as described above. This occurs when a task node has a large number of child nodes, each with many alternatives, and the children are joined under the `sum()` qaf. In this case, the solution for the build-up problem above will control the number of alternatives that reside a the child nodes before alternatives are created for the parent node, and the instant solution will control the number of alternatives generated at the parent node.

Thus the $O(2^m)$ type of complexity is controlled by focusing alternative production on alternatives that best satisfice to meet the client goal criteria. Note that the combinatorics of the alternative generation process remain unchanged, but the actual number of alternatives generated and considered during processing is kept manageable. The end result is that at the top-level task the set of alternatives that can be scheduled is much smaller than if the goal criteria is not used to guide processing. As mentioned previously, not all alternatives associated with the root task node are turned into schedules. Again, the client goal criteria is used to focusing processing on alternatives that seem most likely to lead to schedules that will meet the spirit of the goal criteria, i.e., the set of candidate alternatives compared to the goal criteria and ranked, and the most highly rated alternatives are built into schedules.

[6] This problem does not occur in the example discussed in Section 1.

Once an alternative is selected to be scheduled, a heuristic method rating process is used to determine the proper ordering of the alternative's methods to create a schedule. The method rating approach controls the $O(n!)$ complexity by not generating all possible orderings of the methods. Schedules are constructed by iterating over the set of unscheduled methods for the alternative and calculating a numerical rating for each of the methods. After each pass through the list of unscheduled methods, the highest ranked method is added to the schedule and deleted from the set of remaining candidate methods. In the event that the candidate method set is not empty, and no method can be inserted on the schedule due to various constraints, a slack-time element may be inserted or scheduling of the given alternative may halt at that point. Methods are rated using the following heuristics:

- Enforce hard NLEs, i.e., enforce precedence constraints. This heuristic examines the relationships of the method being rated and if the method must be preceded by one or more other methods, and the methods have not yet been scheduled, then the method in question is marked as unschedulable at this time. Note that the process of determining whether or not a given method is actually enabled entails determining whether or not an enabler is enabled, and so forth, potentially exploring the entire task structure.

- Enforce earliest start times and deadlines. Earliest start times specify the earliest time at which a particular method can be scheduled and deadlines specify a time at which a particular method must be completed. This heuristic determines if either of these hard constraints would be violated by scheduling, or not scheduling, the method at hand in a probabilistic fashion (as start and end times are probabilistic). If scheduling the method is likely to result in violation of a hard constraint, then it is marked as unschedulable.

- Try to take advantage of positive soft NLEs, where doing one activity before another improves overall utility. This class of heuristics examines the positive soft interactions between tasks and gives preference to methods that have a positive soft interaction on other unscheduled methods.

- Try to avoid negative soft NLEs, where doing one activity before another degrades overall utility. This class of heuristics is the converse of the class above. Methods that have negative effects on other methods if they precede the other methods in execution are deferred if possible.

- Try to satisfy external commitments and avoid violating them. This class of heuristics deals with commitments made with other problem solving agents. If scheduling the particular method will violate a commitment, its rating is downgraded, if scheduling the particular method will satisfy a commitment, its rating is upgraded.

- Try to improve overall schedule quality quickly - a greedy heuristic. In problem solving domains where uncertainty is an issue, a good heuristic is often to try to get some useful work done as soon as possible. This heuristic gives preference to methods whose execution will achieve some useful work immediately, i.e., to methods whose execution achieves some level of quality for the top level task.

While the complexity of some of the method rating heuristics is polynomial in the number of task structure nodes, overall the savings of this approach versus the $O(n!)$, $\omega(2^n)$ and $o(n^n)$ by Stirling's approximation, expense of exploring even a portion of the possible orderings is pronounced and makes the problem tractable.

After each schedule is generated (all methods are scheduled or discarded due to unsatisfiable constraints) it is critiqued by a set of improvement heuristics that ascertain if adding other methods or alternatives to the schedule will improve its overall quality, cost, and duration characteristics. Typically, the critics look for methods that are positively affected by hard or soft interactions that are missing from the current schedule. For example, if performing method `A` may improve the quality and reduce the cost of method `B`, but method `A` is omitted from a schedule that includes method `B`, it may be worthwhile to add `A`. Improvements are suggested to the system not by tweaking the schedule at hand, but by suggesting an alternative, or set of alternatives, that includes the items lacking in the current

schedule. Schedules are not directly modified by the critics because there are generally many different possible ways to achieve the improvement and it would require potentially more work to perform the trade-off and constraint analysis necessary to integrate the improvement post-hoc than to regenerate the schedule from the expanded alternative. Additionally, as the addition of new methods to the method set associated with the alternative (paired with the schedule being critiqued) will change its quality, cost, duration, and uncertainty characteristics, it must again be compared to other candidate alternatives to determine if it is even worthwhile to build a schedule from the expanded alternative. The newly created alternatives are thus added to the set of eligible alternatives and selected, or not, according to how well their potentials meet the client's criteria relative to the other alternatives.

The process of selecting alternatives and building schedules iterates until the number of schedules crosses a threshold, all the alternatives are scheduled, the time allocated to scheduling runs-out, or the remaining alternatives cannot lead to better schedules (determined by the alternatives' potentials). Schedules are then rated against the client's quality, cost, and duration criteria and the best one is returned for execution. This iterative process has an anytime [27, 25] flavor since generating the set of alternatives for the root task is relatively inexpensive and fast real-timewise, due to complexity control, compared to the process of building schedules. The scheduler can generate a small set of schedules quickly, but given more time, it can explore more schedules and increase the probability that a "better" schedule is created.[7]

As mentioned previously, the scheduling process is typically not a one-shot activity. Because of the uncertainty involved, it is entirely possible for a method execution to return results outside of the bounds of expectations thus requiring rescheduling. However, in situations where rescheduling is not appealing, the scheduler can create less efficient schedules that are fault tolerant by making conservative probabilistic assumptions about quality, cost, duration and uncertainty. This relates somewhat to previous work done by Durfee and Lesser [8] and in [9] in which schedules are made "loose" by increasing duration expectations when building schedules, effectively creating a slack-time buffer between each action. Our model is much stronger in that we change expectations based on probabilities rather than using magic numbers, and we do so in all dimensions, quality, cost, and duration.

4. Demonstrating the Efficacy of Design-to-Criteria

To illustrate the efficacy of the Design-to-Criteria approach at controlling combinatorics and custom building schedules, let us consider the problem of building schedules for the moderately complex TÆMS task structure shown in Figure 5. Note that this task structure is far too large for an exhaustive search algorithm. The 17 TÆMS methods translate into a power set with 2^{17} members, each one having $n!$ possible orderings where n is the number of methods in the subset of the powerset being scheduled. Exhaustive search would entail generating all possible $9.7 * 10^{14}$ schedules. We compare Design-to-Criteria with an exhaustive approach on a smaller task structure later in this section.

Since the hypothetical client is interested in schedules that trade-off quality and duration, and is more interested in keeping duration toward the lower end of the spectrum than achieving maximum quality, the criteria is set as follows: the raw-goodness quality slider is set to 40%, the raw-goodness duration slider is set to 60%, and the meta slider for the raw-goodness bank is set to 100%. In other words, only raw-goodness is at issue and quality is 60% as important as duration. This setting models a client with no *a priori* knowledge about expected durations or qualities, hence the emphasis on raw-goodness. Note, we are using two dimensional goal criteria, quality and duration, because visualizing two dimensions is straightforward and the graphs are intuitive. The focusing mechanism works equally well with richer multi-dimensional goal criteria.

[7] If the estimated distributions for quality, cost, and duration that are contained in the alternatives are good indicators of the schedule quality, then the algorithm will produce "good" schedules from the start and adding time only increases the certainty that "better" schedules will not be generated.

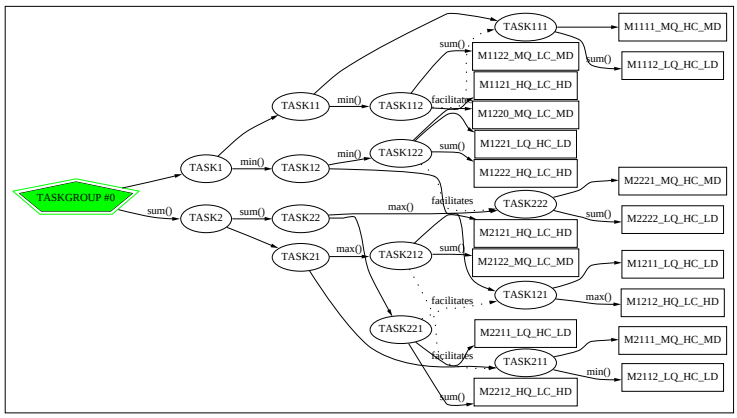
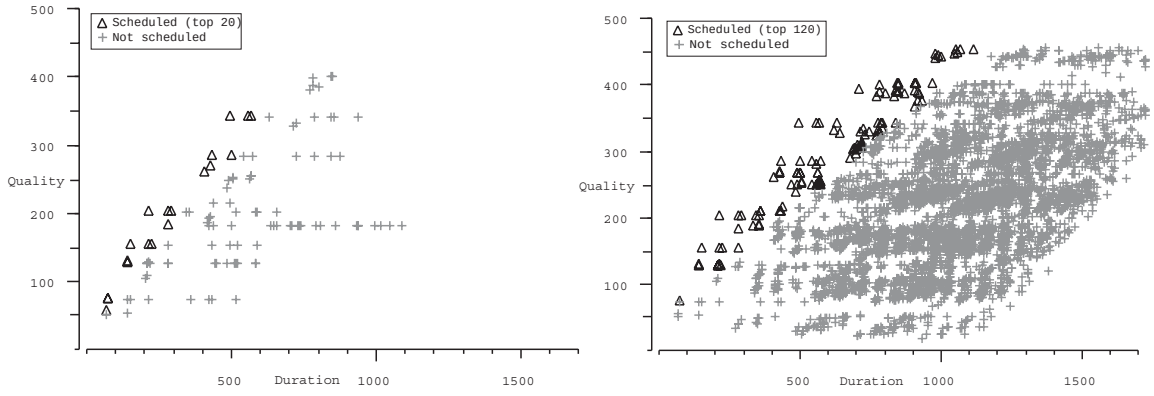


Figure 5. A Moderately Complex Sample TÆMS Task Structure

Figure 6(a) displays the top-level alternatives that are produced using the criteria-directed focusing mechanism for this task structure and the client's goal criteria. Using the goal criteria and the estimates contained in the alternatives, interior tasks' alternative sets are pruned down to 20 members or fewer and the top-level alternative set is pruned down to 40 members or fewer. For this problem instance, 656 alternatives were constructed in total and 39 were initially generated at the top-level, although the actual number of alternatives shown in the graph is slightly larger as the heuristic improvement mechanism creates new alternatives as the schedule process iterates. Note that the alternatives selected for scheduling in this case, again using the criteria, trade-off quality for duration with an emphasis on controlling duration rather than maximizing quality.

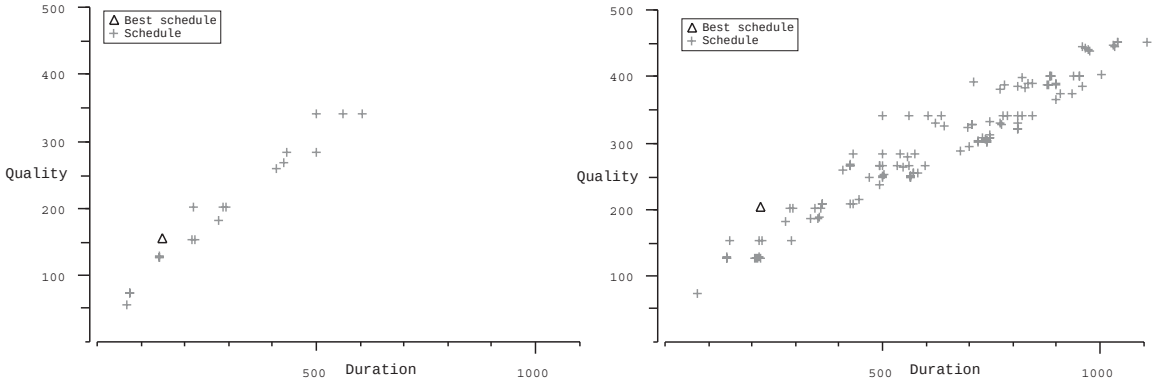
In stark contrast to the economical alternative set generated using the focusing mechanism, Figure 6(b) displays the exhaustive top-level alternative set that results when the focusing mechanism is not used. In this case, 9106 alternatives were explored during processing and 4444 alternatives were generated at the root level. Comparing the two cases, the exhaustive process produced ~ 14 times more alternatives during the intermediate stages and ~ 113 times more alternatives at the root level. Note that the top-rated alternatives in each case have similar quality and duration characteristics and exhibit similar quality/duration trade-offs – keeping duration under control while achieving good/high quality. The exhaustive alternative generation case produced a larger set of reasonable candidate alternatives, but the focused case still found a significant number of reasonable alternatives.



(a) Top-Level Alternatives Produced by Focusing

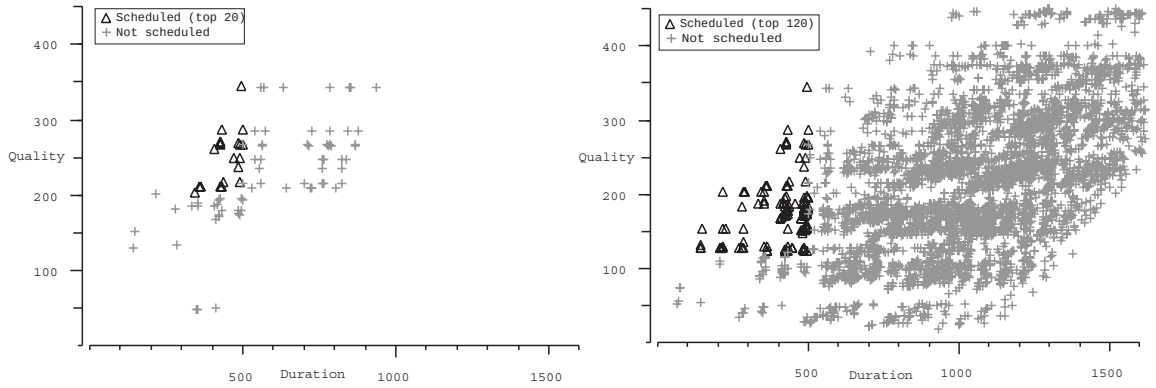
(b) All Top-Level Alternatives

Figure 6. Alternative Sets



(a) Schedules Produced from Focused Alternative Set (b) Schedules Produced from Full Alternative Set

Figure 7. Heuristically Produced Schedule Sets



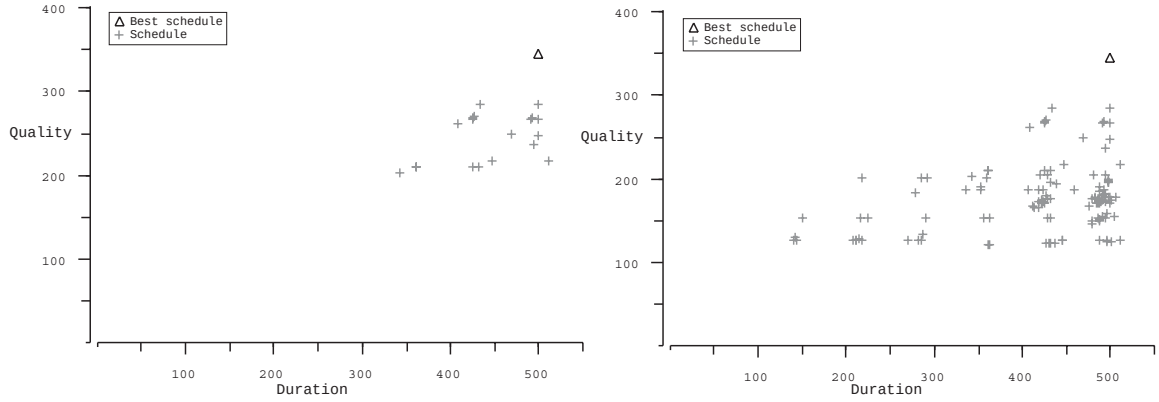
(a) Top-Level Alternatives Produced by Focusing (b) All Top-Level Alternatives

Figure 8. Alternative Sets

The schedules produced via the heuristic scheduling mechanism from the two sets are shown in Figures 7(a) and 7(b) respectively. In the focused case, 20 alternatives were selected for scheduling and in the exhaustive case, 120 alternatives were scheduled. Note that the most highly rated schedules for the two cases have similar quality and duration trade-offs and characteristics and clearly belong to the same neighborhood of possible solutions. Quantitatively, the top schedule produced from the focused alternative set achieves 75% of the quality of the top schedule produced from the exhaustive alternative set, and does this in 68% of the duration. However comparing the solutions in this fashion is not methodologically sound as the ranking and selection mechanisms are relative rather than absolute, i.e., the "best" schedules in each case are the best relative to the rest of the solution set and the sets for the two cases are different.

Let us consider another example using the same TÆMS task structure (Figure 5). In this case, the client has *a priori* knowledge about reasonable durations and is interested in good quality solutions that take 500 or fewer time units to execute. The corresponding slider settings are: raw-goodness for quality set to 100%, the duration limit set to 100% accompanied by a real value of 500, the meta slider for raw-goodness set to 50% and the meta slider for the thresholds/limits bank is also set to 50%. Thus raw-goodness in quality is equally as important as staying under the desired duration. The alternatives for the focused case are shown in Figure 8(a) and the exhaustive alternative set is shown in Figure 8(b). Once again, the focusing technique produced a reasonable set of candidate alternatives and the characteristics

of the highest-ranked alternatives are similar to those of the exhaustive set. The schedules produced heuristically from the two alternative sets are shown in Figures 9(a) and 9(b) respectively. As with the alternatives, the schedules produced using the focusing mechanism meet the spirit of the goal criteria as well as those produced from the exhaustive alternative set and the most highly ranked schedules are actually identical in this case.



(a) Schedules Produced from Focused Alternative Set (b) Schedules Produced from Full Alternative Set

Figure 9. Schedule Sets

This particular task structure provides a fertile search space – there are many possible candidate solutions. However, the focusing mechanism still performs well even when the search space is less fertile. A less fertile search space typically means that the "best" schedule produced, regardless of the approach, does not adequately satisfice to meet the goal criteria. A less fertile search space does not typically affect the ability of our approach to find reasonable schedules. Astute readers will notice that the NLEs (non-local-effects) present in the task structure are soft facilitation relationships rather than hard precedence relationships. Soft NLEs actually pose more of a challenge for the focusing mechanism and the heuristics than hard task NLEs. Because hard NLEs are hard constraints it is fairly straightforward for the heuristics to reason about them and to schedule accordingly (assuming the constraints are satisfiable).

Now let us consider an exhaustive scheduling case. Figure 10 displays a much smaller task structure that is amenable to exhaustive search. In this case the goal criteria is as follows: in the raw-goodness bank the quality slider is set to 100%; in the threshold/limit bank the quality slider is set to 50% with a threshold value of 100 quality units and the duration slider is set to 50% with a limit value of 500 time units; in the meta-bank, the raw-goodness slider is set to 25% and the limit/threshold slider is set to 75%. This criteria suits a client interested in quality over a threshold and duration under a limit, and within that range, prefers increased quality.

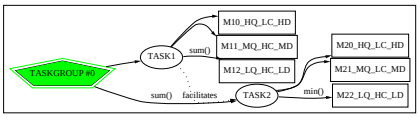
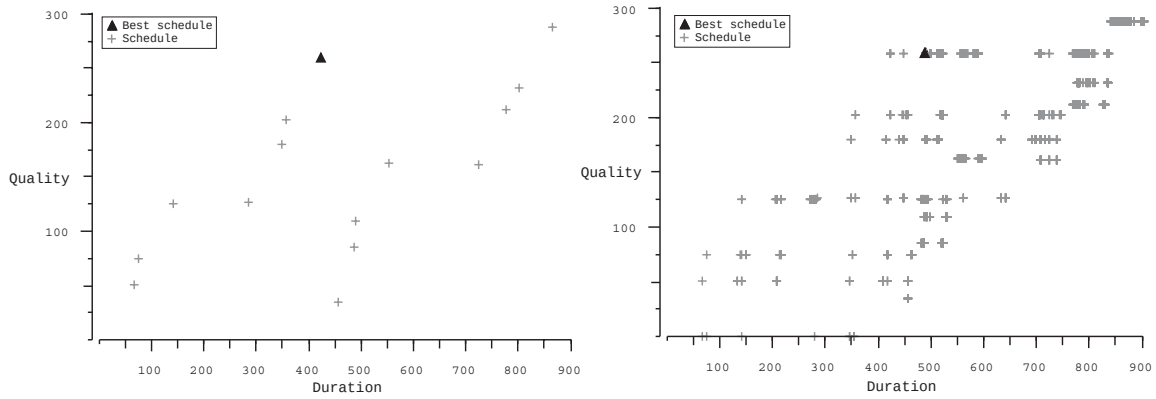


Figure 10. A Simple TÆMS Task Structure

Figure 11(a) shows the schedules produced from the heuristic focused approach and Figure 11(b) shows the schedules produced by pure exhaustive search. In this case all possible schedules of all possible length are generated and then ranked using the goal criteria. The exhaustive search case produced 1955 schedules while the focused heuristic approach produced 38 alternatives and constructed 15 schedules;



(a) Schedules Produced from Focused Alternative Set (b) Schedules Produced via Exhaustive Search

Figure 11. Schedule Sets

the exhaustive algorithm generated ~ 51 times more schedules. However, since the work involved in the creation of a single schedule in the exhaustive case is less than the work invested in a single schedule in the heuristic case, these numbers are not directly comparable. In terms of cpu time, the exhaustive case required ~ 32 times the amount of cpu time required by the heuristic case. Clearly, the focused heuristic approach compares favorably with the exhaustive algorithm both in terms of efficiency and in the quality of the results. In fact, although the "best" schedules in this case have different durations they are both equally as good relative to the goal criteria. The heuristically generated best schedule achieves the same amount of quality as the exhaustively generated best schedule, and does so in less time. However, the criteria specifies that schedules whose quality is over a threshold and whose duration is below a limit are preferred, and that quality should differentiate schedules when those constraints are satisfied, thus within these parameters schedules are not differentiated by duration. The exhaustive case actually produced several schedules with the same quality as the "best" schedule and with lower duration, however, since raw-duration is not a factor in the goal criteria the schedules in this neighborhood are all ranked equally.

5. Conclusion and Future Work

In open environments applications must adapt processing to meet available resources and the different goal criteria of different clients. Design-to-Criteria addresses these requirements with a flexible computation approach to task scheduling. Design-to-Criteria produces results in the face of high-order complexity by satisficing with respect to the client goal criteria and with respect to the scheduling activity itself. Algorithmically, the satisficing methodology takes the form of criteria-directed focusing, approximation, heuristic decision making, and heuristic error correction. Criteria-directed focusing controls the $O(2^m)$ source of complexity by limiting the number of alternatives that are created during the scheduling process and by focusing schedule building on solutions and partial solutions that are most likely to lead to a solution that meets the spirit of the goal criteria. The alternative approximation defers the $O(n!)$ ordering-based complexity while providing estimates for the quality, cost, and duration that will result from ordering the methods contained in the alternative. Heuristic decision making, in conjunction with heuristic error correction, replaces the potential $O(n!)$ ordering complexity with a one pass heuristic approach to building schedules that is low-order polynomial in the worst case. The Design-to-Criteria scheduling approach is effective and efficient.

Focused processing is central to Design-to-Criteria and one area of future work lies in automating the process of determining the degree to which the algorithm should focus. This requires meta-level

analysis of task structures and the resource constraints on a given problem instance to ascertain how much space to explore in order to find a good satisficing solution. Because of complex task interactions and scheduling constraints on individual tasks and methods, e.g., earliest start times, deadlines, etc., this meta-analysis must entail more than simply counting nodes. The analysis will have to attempt to classify the task structures using an approximation, estimate or abstraction of the task structures to get an idea of the complexity of the constraints involved.

An area of future work related to the meta-analysis focus is the refinement of the interface between the scheduler and other complex problem solving components and/or humans. Interactive negotiation [10] between the client and the scheduler could control and refine satisficing activities as they happen. With the current model of criteria specification followed by application, it is possible that none of the generated schedules satisfactorily meet the client's ideal needs (though the one that best satisfices to meet the criteria will be returned). In this case, the client may want to explore more of the search space or may prefer an alternate set of criteria rather than taking a satisficing view of the original criteria. Interactive negotiation during the alternative generation and evaluation phases could refine client expectations based on the *estimates* associated with the alternatives. This would enable the scheduler to adjust its intermediate processing to align with the client's refined criteria before any work is spent building schedules. Negotiation during the scheduling phase could help refine the criteria based on the features of schedules as they are produced. The refined criteria would then alter the selection of alternatives and retarget the scheduling activity. Negotiation during the scheduling process is clearly the next step in exploiting and leveraging the power of the Design-to-Criteria paradigm.

Uncertainty is also an area of important future work in Design-to-Criteria. Highly uncertain actions should be performed as early as possible to reduce the amount of work that is potentially wasted by an action failure. Because of complex task interactions and the complex semantics of the functions that determine how quality is accumulated by tasks via their subtasks, determining which actions are most important to the schedule and to what degree, is not a trivial or computationally inexpensive process. Contingency scheduling for methods likely to fail is also a possibility.

References

1. N. Carver and V. Lesser. The DRESUN testbed for research in FA/C distributed situation assessment: Extensions to the model of external evidence. In *Proceedings of the International Conference on Multiagent Systems*, June, 1995.

2. T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, St. Paul, Minnesota, August 1988.

3. Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms.* PhD thesis, University of Massachusetts, 1995.

4. Keith S. Decker. TÆMS: A framework for analysis and design of coordination mechanisms. In G. O'Hare and N. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, chapter 16. Wiley Inter-Science, 1995.

5. Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 2(4):215–234, December 1993. Special issue on "Mathematical and Computational Models of Organizations: Models and Characteristics of Agent Behavior".

6. Keith S. Decker and Victor R. Lesser. Coordination assistance for mixed human and computational agent systems. In *Proceedings of Concurrent Engineering 95*, pages 337–348, McLean, VA, 1995. Concurrent Technologies Corp. Also available as UMASS CS TR-95-31.

7. K.S. Decker, V.R. Lesser, M.V. Nagendra Prasad, and T. Wagner. MACRON: an architecture for multi-agent cooperative information gathering. In *Proccedings of the CIKM-95 Workshop on Intelligent Information Agents*, Baltimore, MD, 1995.

8. E. Durfee and V. Lesser. Predictability vs. responsiveness: Coordinating problem solvers in dynamic domains. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 66–71, St. Paul, Minnesota, August 1988.

9. S. Fujita and V.R. Lesser. Centralized task distribution in the presence of uncertainty and time deadlines. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, Japan, 1996.

10. Alan Garvey, Keith Decker, and Victor Lesser. A negotiation-based interface between a real-time scheduler and a decision-maker. In *AAAI Workshop on Models of Conflict Management*, Seattle, 1994. Also UMASS CS TR–94–08.

11. Alan Garvey and Victor Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1491–1502, 1993.

12. Alan Garvey and Victor Lesser. Representing and scheduling satisficing tasks. In Swaminathan Natarajan, editor, *Imprecise and Approximate Computation*, pages 23–34. Kluwer Academic Publishers, Norwell, MA, 1995.

13. Eric Horvitz, Gregory Cooper, and David Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, August 1989.

14. Eric Horvitz and Jed Lengyel. Flexible Rendering of 3D Graphics Under Varying Resources: Issues and Directions. In *Proceedings of the AAAI Symposium on Flexible Computation in Intelligent Systems*, Cambridge, Massachusetts, November 1996.

15. Frank Klassner. *Data Reprocessing in Signal Understanding Systems.* PhD thesis, University of Massachusetts at Amherst, Amherst, Massachusetts, September 1996.

16. Victor Lesser, Bryan Horling, Frank Klassner, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. Information Gathering as a Resource Bounded Interpretation Task. UMASS Department of Computer Science Technical Report TR-97-34, March, 1997.

17. S. H. Nawab and E. Dorken. Quality versus Efficiency Tradeoffs in STFT Computation. *IEEE Transactions on Signal Processing*, April 1995.

18. S. H. Nawab and T. Quatieri. Short-time Fourier transform. In S. H. Nawab and T. Quatieri, editors, *Advanced Topics in Signal Procesing.* Prentice-Hall, New Jersey, 1988.

19. T. Oates, M. V. Nagendra Prasad, and V. R. Lesser. Cooperative Information Gathering: A Distributed Problem Solving Approach. Computer Science Technical Report 94–66, University of Massachusetts, 1994. To appear in Journal of Software Engineering, Special Issue on Developing Agent Based Systems, 1997.

20. Stuart J. Russell and Shlomo Zilberstein. Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 212–217, Sydney, Australia, August 1991.

21. Regis Vincent, Bryan Horling, Thomas Wagner, and Victor Lesser. Survivability simulator for multi-agent adaptive coordination. In *Proceedings of the First International Conference on Web-Based Modeling and Simulation*, 1998. To appear.

22. Thomas Wagner, Alan Garvey, and Victor Lesser. Complex Goal Criteria and Its Application in Design-to-Criteria Scheduling. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, July 1997. Also available as UMASS Department of Computer Science Technical Report TR-1997-10.

23. Thomas Wagner, Alan Garvey, and Victor Lesser. Leveraging Uncertainty in Design-to-Criteria Scheduling. UMASS Department of Computer Science Technical Report TR-97-11, January, 1997.

24. S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 79(2), December 1995.

25. Shlomo Zilberstein. Operational rationality through compilation of anytime algorithms. Ph.D. Dissertation, Department of Computer Science, University of California at Berkeley, Berkeley, CA, 1993.

26. Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

27. Shlomo Zilberstein and Stuart J. Russell. Constructing utility-driven real-time systems using anytime algorithms. In *Proceedings of the IEEE Workshop on Imprecise and Approximate Computation*, pages 6–10, Phoenix, AZ, December 1992.

28. M. Zweben, B. Daun, E. Davis, and M. Deale. Scheduling and rescheduling with iterative repair. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*, chapter 8. Morgan Kaufmann, 1994.