9 HARDWARE/SOFTWARE PARTITIONING USING THE LYCOS SYSTEM

Jan Madsen, Jesper Grode, and Peter V. Knudsen

Department of Information Technology
Technical University of Denmark
Lyngby, Denmark

9.1 INTRODUCTION

Hardware/software partitioning is often viewed as the synthesis of a target architecture consisting of a single processor and a single dedicated hardware component (full custom, FPGA, etc.) from an initial system specification, e.g. as in [289]. Even though the **single processor**, single **dedicated hardware** architecture is a special and limited example of a distributed system, the architecture is relevant in many areas such as **DSP** design, construction of embedded systems, software execution acceleration and hardware emulation and prototyping [290], and it is the most commonly used target architecture for automatic hardware/software partitioning.

J. Staunstrup and W. Wolf (eds.), Hardware/Software Co-Design: Principles and Practice, 283-305.

In this chapter we present the LYCOS (LYngby CO-Synthesis) system which is an experimental hardware/software co-synthesis system. In its current version, LYCOS may be used for hardware/software partitioning using a target architecture as described above. In this chapter we will focus on how LYCOS is used to do design space exploration in codesign. Details about the algorithms used within LYCOS can be found elsewhere [291, 292, 293, 294].

The chapter is organized as follows. Section 9.2 presents the problem of hardware/software partitioning and its relation to design space exploration. In Section 9.3 we give an overview of the LYCOSsystem. Section 9.4 gives a step by step walk-through of a partitioning session in LYCOS. Section 9.5 describes how LYCOSmay be used for design space exploration. Finally, Section 9.6 provides an evaluation of the system and some conclusions on the current version of LYCOS.

9.2 PARTITIONING AND DESIGN SPACE EXPLORATION

Consider the hardware/software **partitioning** and a set of requirements to be fulfilled by the partitioned system. In order to obtain a feasible partition, that is, a partition which fulfills the requirements, we need to know the target architecture, i.e. the processor on which to run the software, the technology of the dedicated hardware, and the interface used for communication between hardware and software. However, the choice of target architecture will greatly influence the outcome of the partition as each target architecture will have different "best" partitions. For instance, selecting a *fast* but expensive processor may lead to little (or no) hardware needed, while selecting a *slow* but cheap processor may require a large amount of dedicated hardware. Thus, in order to solve the problem efficiently we need a way to explore the design space.

Typically we will have an idea about possible suitable target architectures. These may be selected based on the designer's experience, the desire to reuse predesigned components or the use of third party components. One way to explore the design space is to find the best partition for each possible target architecture and select the best among these.

To find the best partition for a given target architecture, we need a model to represent computation and ways to estimate metrics of software, hardware, and communication.

It is important that the model of computation is independent of any particular implementation strategy, that being software or hardware. This will allow for an unbiased design space exploration as well as for translations from various specification languages. One model targeted towards partitioning is based on extracting chunks of computation, called basic scheduling blocks or just blocks. A partition in this model is an enumeration of each block indicating whether

Ideal Partition of Resources for Metareasoning*

Eric J. Horvitz
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
Stanford, California 94305

John S. Breese
Rockwell International Science Center
Palo Alto Laboratory
444 High Street
Palo Alto, CA 94301

January 1990

Technical Report KSL-90-26, Knowledge Systems Laboratory, Stanford University.

Abstract

We can achieve significant gains in the value of computation by metareasoning about the nature or extent of base-level problem solving before executing a solution. However, resources that are irrevocably committed to metareasoning are not available for executing a solution. Thus, it is important to determine the portion of resources we wish to apply to metareasoning and control versus to the execution of a solution plan. Recent research on rational agency has highlighted the importance of limiting the consumption of resources by metareasoning machinery. We shall introduce the *metareasoning-partition* problem—the problem of ideally apportioning costly reasoning resources to planning a solution versus applying resource to executing a solution to a problem. We exercise prototypical metareasoning-partition models to probe the relationships between time allocated to metareasoning and to execution for different problem classes. Finally, we examine the value of metareasoning in the context of our functional analyses.

*This work was supported by a NASA Fellowship under Grant NCC-220-51, by the National Science Foundation under Grant IRI-8703710, and by the U.S. Army Research Office under Grant P-25514-EL. Computing facilities were provided by the SUMEX-AIM Resource under NLM Grant LM05208.

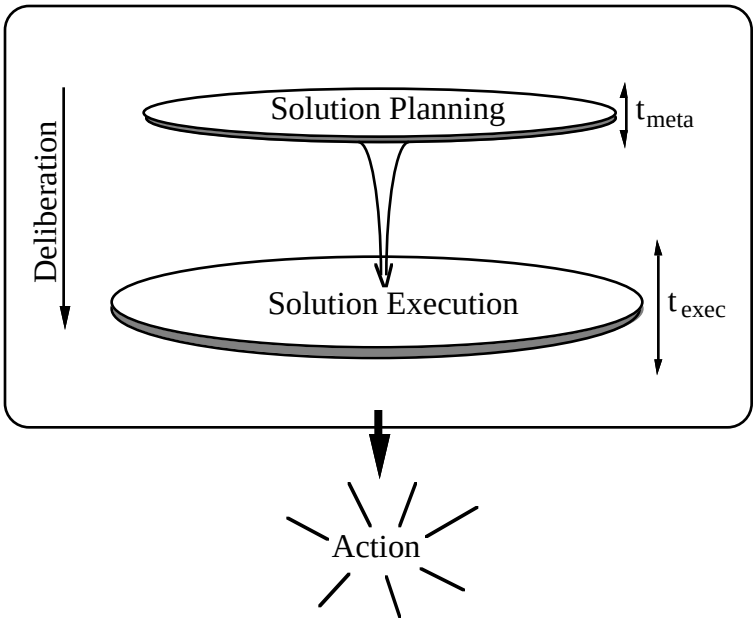


Figure 1: We are interested in determining the ideal quantity of resources that should be applied to metareasoning, given knowledge about the ability of metareasoning to reduce the difficulty of solving the base problem.

1 Introduction

Recent research, spanning AI and the decision sciences, has highlighted the usefulness of metareasoning for directing and limiting base-level problem solving [1, 9, 11]. Metareasoning refers to several classes of explicit consideration and control of problem-solving processes; these include planning a solution, limiting the extent of base-level problem solving, choosing the next best problem-solving step, and searching for a better formulation of a problem instance. Metareasoning can increase the efficiency or increase the utility of base-level inference. However metalevel deliberation can consume a significant portion of valuable computational resources.

Recent investigation of metareasoning and rational action has highlighted the importance of managing the consumption of resources by metareasoning processes. Although some of our earlier work relied on tractable solutions to metalevel decision problems [9], we have stressed the need to generalize analyses and implementations to the case where a metareasoner may dynamically choose to consume greater portions of the total quantity of resources used in problem solving [7]. Indeed, given a problem, a context, and the details of a computational architecture, it may be best to expend a significant proportion of the total consumed resources, at the metalevel. At other times, however, it may be best to expend little or no resources on metareasoning. We shall discuss the problem of ideally apportioning costly reasoning resources to metalevel inference versus applying resource to executing a solution to a problem. We call this the *metareasoning-partition* problem. This metareasoning-partition problem is an important link in the pursuit of rational behavior under resource constraints.

We shall develop insights about the ideal partition of resources for metareasoning by constructing and solving expressive mathematical models of the metareasoning-partition problem. These models capture prototypical functional relationships between reasoning and metareasoning for a large class of problems. We exercise the models to identify the ideal configuration

of a small number of variables that define an agent's metareasoning policy. In a related paper, we describe the results of empirical analyses to instantiate the constant parameters of the metareasoning-partition models for the case of belief-network reformulation [4]. Our analyses shall center on solution-planning problems. In solution-planning problems, effort is expended on the meta-analysis of the problem instance before the solution is executed. In related work on the metareasoning-partition problem, Shekhar and Dutta have examined the tradeoff between solution planning and execution for sample search algorithms; these investigators applied their results to enhance the speed of database queries [12]. Research on more general forms of metareasoning, that are interleaved with base-level reasoning, is described in [9, 11]. We shall touch on uncertainty issues, but will remain focused largely on the analyses of deterministic cases; a discussion of resource partition under uncertainty is found in [3].

2 Metareasoning Partition Problem

The total time t for generating a solution includes the time used by solution planning and for the execution of the solution. We use t_m to refer to the time used for metareasoning about the problem before executing the solution, and use t_e to refer to the computation time used by base-level execution. The solution of the metareasoning-partition problem is a meta–metanalysis that requires a quantity of reasoning resource itself; the cost of a dynamic analysis of the ideal resource partition is the time required to solve an optimization problem or to apply a previously generated solution. As we shall see, it is feasible to determine the ideal metareasoning resource partition with an inexpensive, constant-cost analysis. We include a constant term t_{mm} in our metareasoning optimization equations to address, in a reflective manner, the cost of solving those equations. Thus, the total time used by the base-level execution, metareasoning, and optimization of the metareasoning partition is

$$t \;=\; t_e + t_m + t_{mm}$$

Solution-planning methods are precursory metalevel analyses that increase the efficiency of the execution of a naive base-level reasoning strategy. We can model the relationship between solution planning and execution with functions that capture the efficacy of solution planning for enhancing base-level inference. We shall examine the ideal metareasoning partition for utility-directed and goal-directed problem solving.

2.1 Utility-Directed Computation

Utility-directed computation centers on the optimization of the value of computation, as dictated by possible actions and an explicit or implicit model of preference. This class of problem solving is receiving increasing attention among AI investigators [7, 9, 2, 11]. With utility-directed computation, we measure and represent the expected value of computation with a numerical measure of preference, termed *utility*. Utility is defined by the axioms of utility theory enumerated by von Neumann and Morgenstern over four decades ago [13].[1]

[1] A detailed review of past, and more recent, efforts to apply probability and decision theory for solving challenging artificial-intelligence problems is found in [8].

We use *comprehensive value* u_c to refer to the *expected utility* associated with the application of a computational strategy. The comprehensive value can often be decomposed into two components: the *object-related* value and *inference-related* value. The *object-related* value u_o is the expected utility associated with the best action or result available to an agent, given a state of the world. The *inference-related* cost u_i is the expected cost associated with delay of computation. If we can decompose u_c into u_o and u_i, and these components of utility are related through addition, the expected value of computation (EVC) is just the difference between the increase in object-level utility and the cost of the additional computation.

A metareasoning-partition analysis of utility-directed computation considers the ability of solution planning to change the rate at which utility is delivered with execution time. Metareasoning partition analyses for utility-directed computation are based on the maximization of the utility of computation with respect to a model solution planning and execution

$$\max_{t_e,t_m} u_c(t_m,t_e) \quad = \quad \max_{t_e,t_m}[u_o(t_m,t_e) - u_i(t_m+t_e+t_{mm})] \tag{1}$$

We shall introduce and analyze expressive models for the ideal partition of resources for utility-directed computation in Section 3.

2.2 Goal-Directed Computation

Goal-directed computation refers to a more traditional approach to problem solving, centering on the computation of discrete, predefined results from problem instances. A goal-directed reasoning system seeks to minimize the total time required to compute important goals. Given the functional relationship between t_e and t_m, our goal is to identify a t_m^* that minimizes t; thus, our metareasoning partition analyses for goal-directed computation focus on the minimization of the total time to solution

$$\min_{t_e,t_m}(t_e + t_m + t_{mm}) \tag{2}$$

We shall examine and solve models of ideal metareasoning partition for goal-directed systems in Section 4.

3 Partition for Utility-Directed Problems

In this section, we will introduce models that can be used to represent and solve the problem of ideally apportioning resource to solution planning for utility-directed problems. We will focus our attention on utility optimization with flexible reasoning strategies, described in Sections 3. We shall first examine the problem of determining ideal problem-solving cessation, in Subsection 3.2. In Subsection 3.3, we will examine the case of ideal partitioning of resources, making use of tools from the ideal cessation problem.

3.1 Flexible Computation

A important aspect of developing utility-directed reasoners that are resilient to uncertain challenges and resources is the development of *flexible* or *anytime* computation strategies [5, 6, 2].

We formally analyzed properties of flexible reasoning that are desirable for reasoning under bounded resources [7]. First, we wish our solution strategies to deliver some object-level return for any quantity of allocated resource. Thus, we desire our strategies to provide us with partial results that have value that increases monotonically with allocated resource. We also desire our strategies also to exhibit graceful degradation, or to be relatively insensitive to small reductions in the allocated resource. That is, we wish to limit the magnitude of discontinuous jumps in the value of a partial result as we diminish the quantity of resources devoted to a problem, preferring instead a degree of incrementality or *continuity* in the refinement of partial results with the application of resources. Finally, we wish our results to demonstrate *convergence* on an ideal answer with sufficient resources.

Flexible computation is especially useful for reasoning under uncertain challenges and deadlines; flexible problem-solving generates immediate object-level returns on small quantities of invested computation, and minimizes the risk of dramatic losses in situations of uncertain resource availability. We highlighted these benefits in an analysis of several classes of varying and uncertain cost and deadline [7].

Flexible algorithms for inference have been constructed. These include the bounded-conditioning approach [10] that produce, and iteratively tighten, upper and lower bounds on probabilities of interest for probabilistic inference and decision making under uncertainty.

3.2 Ideal Reflection with Flexible Computation

We shall first consider a special case of solution planning that addresses the problem of *ideal reflection*. We wish to determine the ideal time for dwelling on a problem execution before acting. We wish to compute the ideal fraction of the problem to solve, given a flexible reasoning strategy that works to refine an answer as increasing amounts of resource is applied.

Let us consider an example of a medical-diagnosis problem under time constraints, borrowed from [6]. Assume we have a flexible strategy for the generation of a result needed for making a time-pressured medical therapy decision. Figure 2 displays the object-level utility, u_o, delivered by a flexible strategy as a result is refined with additional time t. To reason about optimal allocation of resources to this strategy, we must also consider the costs of delay. Assume that we find, through preference assessment, that a patient incurs a linear cost when a clinician delays an action in a particular context. Figure 2 shows a specific linear inference-related cost function $u_i(t) = ct$. Assuming that the inference-related cost and the object-level utility are decomposable, and are related by addition, we can determine the comprehensive utility, u_c, by adding the object-level utility and the inference-related cost. As portrayed in Figure 2, u_c rises to a maximum, u_c^*, at an ideal stopping time t^*. After reaching u_c^*, it is not worthwhile to expend additional resource: for each tick of the clock past t^*, we lose more in the cost of delay than we gain through additional object-level refinement.

We can summarize the relationships among the components of the comprehensive utility, and supply a reasoner with the ability to perform optimizations quickly. For example, if the object-level utility and inference-related costs are related by addition, we know that maxima for U_c will occur when the derivatives

$$\frac{\partial u_o}{\partial t} = -\frac{\partial u_i}{\partial t} \tag{3}$$

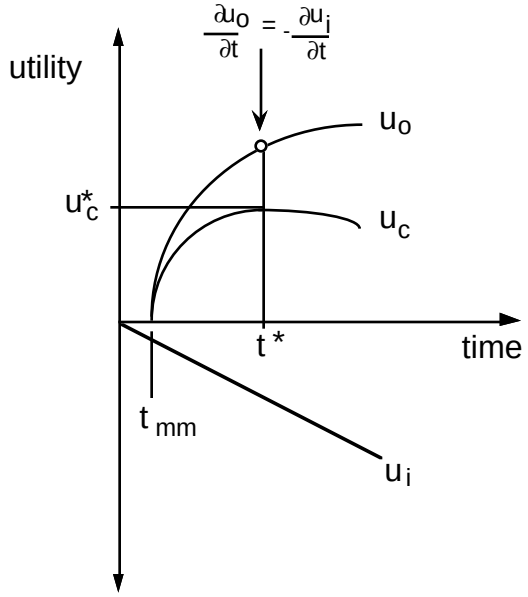


Figure 2: The economics of flexible computation. The graph highlights the fundamental relationships among the object-level value (u_o), the cost of reasoning (u_i), and the comprehensive value of computation (u_c). As indicated by the graph, the ideal halting time (t^*) for this utility model is the time where the magnitude of the derivatives of the object-level value and inference-related cost are equal.

If the second derivative of the object-level utility function is everywhere negative and the second derivative of the cost of computation is everywhere nonnegative, the single maximum will indicate the optimal u_c^*. Otherwise, we may have to compare several allocations of resource to distinguish local from global maxima.

3.2.1 Exponential Model

We now shall develop and solve functions that can model problem solving with flexible computation. Consider the example where we can describe the way the object-level value approaches the value of a complete solution with a negative exponential process

$$u_o(t_e) \quad = \quad 1 - e^{-kt_e} \tag{4}$$

Such a computational process produces results that are refined incrementally and monotonically, and that converge with some quantity of resource on an ideal object-level answer. Let us assume that the cost of delay is separable from the object-level utility and that the object-level value and cost are related by addition, and that we have a linear cost with

$$u_i(t_e) \quad = \quad -c(t_e + t_{mm}) \tag{5}$$

where t_{mm} is the constant cost of the optimization process needed to determine the optimal halting point. We know that the comprehensive utility is just the sum of Equations 4 and 5

$$u_c(t_e) \quad = \quad 1 - e^{-k(t_e)} - c(t_e + t_{mm}) \tag{6}$$

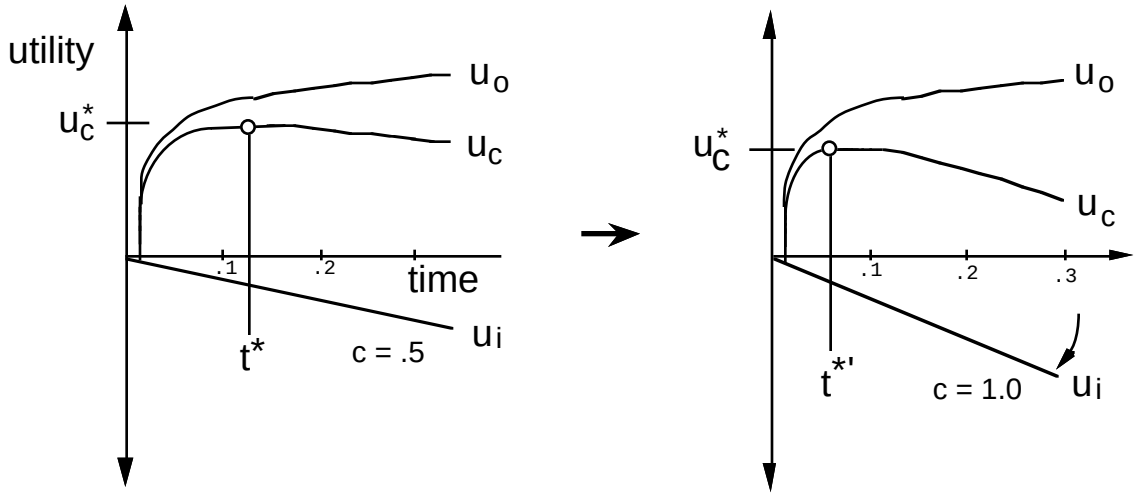


Figure 3: Analysis of the inverse-power model. The graph displays the change in ideal value u_c^* and halting time (t^*), given changes in the cost of reasoning (c). Here, we change the criticality of the situation by doubling the cost of delay.

We can solve for an optimal halting time by differentiating Equation 6 in terms of the time expended on the problem. For our example, the ideal amount of execution time before halting is

$$t_e^* = -\frac{ln(\frac{c}{k})}{k}$$

and the total time before computation cessation is

$$t^* = -\frac{ln(\frac{c}{k})}{k} + t_{mm}$$

Through substitution of the ideal execution time, and cost of our analysis, into the comprehensive utility function, we can calculate the ideal u_c, u_c^*. Note that we could substitute the derivative of an arbitrary monotonic cost function, $\mathcal{C}'(T)$, for c in the above analysis, yielding formulae for more complex cost functions.

3.2.2 Inverse-Power Model

Flexible computation computation can also be modeled by an inverse-power refinement trajectory, as in the following form

$$u_o(t_e) = 1 - \frac{1}{kt_e^a} \tag{7}$$

Assuming a linear cost c with time, the ideal amount of execution time before halting for this model is

$$t_e^* = \left(\frac{a}{kc}\right)^{\frac{1}{a+1}}$$

and $u_c^*(t_e)$, at the ideal halting point, t^*, is

$$u_c^*(t_e) = 1 - k^{-1}\left(\frac{a}{kc}\right)^{-\frac{a}{a+1}} - c(t_e^* + t_{mm}) \tag{8}$$

With a minimal expenditure of resources, we can quickly determine the ideal computation time and utility, as a function of a small number of problem-solving parameters.

The graph at the left of Figure 3 demonstrates how we can use these models to efficiently determine how the optimal halting time and ideal comprehensive utility will change, given changes in the cost of reasoning (dictated by c or, more generally, by a function $\mathcal{C}$), in the rate of refinement (dictated by k), and the cost of determining the optimal halting time (t_{mm}). In the figure, we change the criticality of the situation by increasing the cost of delay. The new optimization shows that we should reflect for less time, and can expect to receive less value from reasoning.

We have studied the behavior of several prototypical equations. As an example, consider the case of problem-solving described by Equation 6, where $k = .1$, and the metareasoning time required for the optimization of execution dwell, t_{mm}, is .01 seconds. In this situation, when the urgency, represented by the cost c, is .04, the ideal execution time, t_e^*, is 9.16 seconds, and the optimal comprehensive utility of computation, u_c^*, is .23. If we double the cost of delay, so that $c = .08$, we find that t_e^* is reduced to 2.23 seconds and u_c^* drops to .02.

3.3 Ideal Metareasoning Partition

In the previous analysis, we assumed the absence of a capability for solution planning to make execution more efficient. We studied a solution planner that could only make decisions about when to halt base-level computation. Now, let us enrich the analyses described in Section 3.2 to represent the ability of a metareasoner to enhance the efficacy of object-level problem solving. We can model metareasoning processes with functions $\mathcal{K}(t_m)$ that modify the rates of problem solving in terms of the resources applied to meta-analysis. In particular, let us model the efficacy of solution planning with functions that dictate the solution-refinement constants k in our object-level utility models as monotonic functions of the amount of time expended for metareasoning. That is, we make k a variable that is a function of the planning time where

$$\mathcal{K}(t_m) \quad = \quad k$$

In cases where we are uncertain about the efficacy of solution planning, our analysis can be extended to consider a probability distribution over k, $p(k|t_m)$. For example, an enriched model for the exponential form, defined in Equation 6, is

$$u_o(t_m, t_e) \quad = \quad 1 - e^{-\mathcal{K}(t_m)t_e} \tag{9}$$

Now, the rate of refinement depends on the amount of solution planning. Similarly, for the inverse-power form, we add a $\mathcal{K}(t_m)$ that is sensitive to solution planning,

$$u_o(t_m, t_e) \quad = \quad 1 - \frac{1}{[\mathcal{K}(t_m)]^b t_e^a} \tag{10}$$

Let us delve more deeply into the inverse-power model of solution planning. Our goal now is to maximize the following model of comprehensive utility with respect to solution planning and execution

$$u_c(t_m, t_e) \quad = \quad 1 - \frac{1}{[\mathcal{K}(t_m)]^b t_e^a} - \mathcal{C}(t_m + t_e + t_{mm}) \tag{11}$$

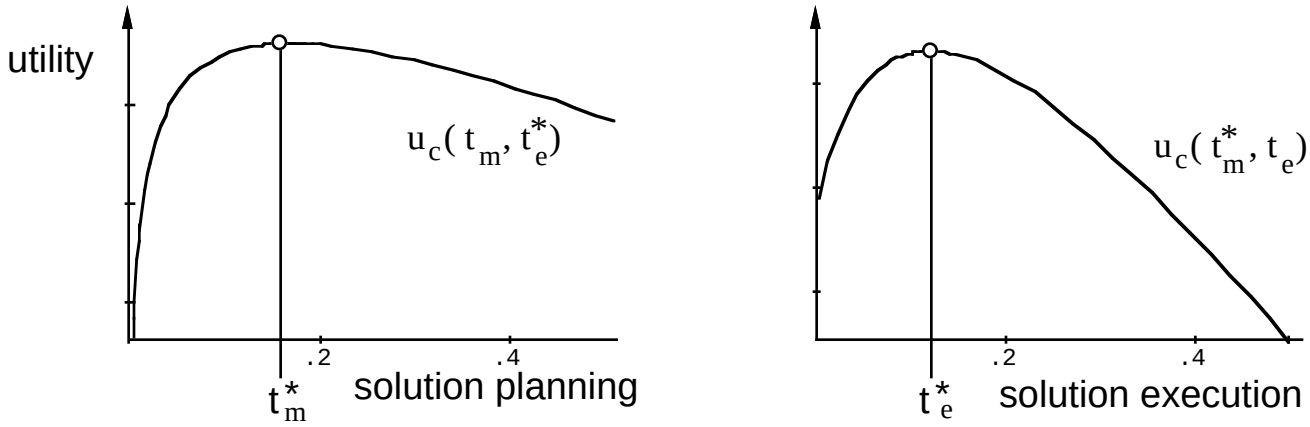


Figure 4: The structure of the optimum. Graph (a) displays the sensitivity of the utility of computation to variation of the time for metareasoning, given an ideal time for execution of the solution. Graph (b) displays the sensitivity of the utility of computation to variation of the time for execution, given an optimal partition of resources for solution planning.

To calculate the ideal amount of time t_m to apply to the process of solution planning, we differentiate Equation 11, in terms of t_e and t_m, and solve the equations simultaneously. We identify the constraint that

$$t_e^* = \frac{a}{b}\frac{\mathcal{K}(t_m)}{\mathcal{K}'(t_m)} \tag{12}$$

Through resubstituting this result, we obtain

$$\mathcal{K}(t_m) = \left(\frac{[b(\mathcal{K}'(t_m))]^{a+1}}{a^a C'(t_m)}\right)^{\frac{1}{a+b+1}} \tag{13}$$

We can substitute different t_m into Equation 13 and solve for t_m^*.[2] We can then determine the optimal execution time, t_e^*, and the ideal value of computation, u_c^*, in terms of t_m^*. Closed-form solutions to Equation 13 can be identified for many families of $\mathcal{K}(t_m)$. Let us derive the ideal planning time and ideal value of computation for the case where the efficacy of solution planning is modeled as a linear increase in k with planning time

$$\mathcal{K}(t_m) = k_o + lt_m$$

where l is a metareasoning-efficiency constant. If we assume a linear cost c with total time to solution, it follows from Equation 13 that

$$t_m^* = l^{-1}\left[\left(\frac{(bl)^{a+1}}{ca^a}\right)^{\frac{1}{a+b+1}} - k_o\right] \tag{14}$$

and

$$t_e^* = \frac{a}{b}\left(\frac{k_o}{l} + t_m^*\right) \tag{15}$$

[2]The existence of a non-zero maxima can be confirmed by checking that the Hessian matrix for the utility function is negative semidefinite at the solution points.

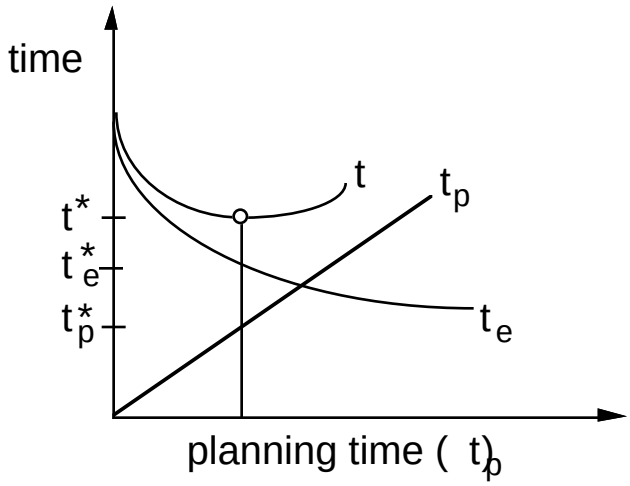


Figure 5: A graph demonstrating relationships between the total time required for solving a goal-directed problem (t^*), and time allocated for solution planning (t^*_m) and execution (t^*_e).

Thus, the ideal planning time and ideal execution time can be quickly calculated from a small number of constant parameters of our model and a constant dictating the urgency of a situation. The ideal value, reached with a system with capabilities described by this model, follows from substituting the ideal solution-planning and execution times into Equation 11.

4 Partition for Goal-Directed Problems

Let us now move from the case of optimizing the value of flexible computation to the special case of minimizing the amount of time required to generate a predefined result. We represent the efficacy of solution planning with a function $\mathcal{P}$ that reports the expected amount of time required for solving a problem, as a function of of the amount of time spent planning the solution strategy. Thus,

$$\mathcal{P}(t_m) \quad = \quad t_e \tag{16}$$

In the general case, we may be uncertain about the relationships among t_m and t_e, and, thus, uncertain about t. In these situations, we are interested in assessing and applying knowledge in the form of probability distributions,

$$\mathcal{P}(t_m) \quad = \quad p(t_e|t_m) \tag{17}$$

conditioned on different amounts of solution planning. We describe solutions to goal-directed computation under uncertainty in [3]. For the deterministic case, the total time for reasoning needed before applying a particular result in the world is

$$t \quad = \quad \mathcal{P}(t_m) + t_m + t_{mm} \tag{18}$$

We seek to minimize t. To determine the optimal partition, we differentiate Equation 18 with respect to solution-planning time and search for a minimum. Thus, we explore solution-planning times where

$$\frac{\partial \mathcal{P}(t_m)}{\partial t_m} \quad = \quad -1 \tag{19}$$

The problem-reduction function $\mathcal{P}(t_m)$ can take on different forms, depending on the relationship identified between execution efficiency and the extent of metareasoning analysis. For example, for the case where solution planning can be expected reduce the complexity of the problem through reformulation, $\mathcal{P}(t_m)$ dictates the reduction in complexity of generating a complete solution as a function of planning time. A problem-reduction function also can represent the expected increases in the rate at which a flexible-computation policy refines a problem. We shall examine this situation for a simple example.

Assume that the abilities of our base-level problem-solver are described by an exponential model described in Equation 9. Unlike our utility-directed examples, where we sought to optimize utility, our goal now is to generate a predefined goal. Assume that our goal is to compute until a result, worth some fraction f of the complete value of a problem, is obtained. We are interested in minimizing the total time of delay, given

$$1 - e^{-\mathcal{K}(t_m)t_e} \quad = \quad f \tag{20}$$

Thus, we wish to minimize Equation 18, subject to the constraints of Equation 20.

$$\mathcal{P}(t_m) \quad = t_e = \quad -\frac{ln(1-f)}{\mathcal{K}(t_m)} \tag{21}$$

Differentiating 21 with respect to solution-planning time, and solving for minima, we find that t_m^* satisfies the constraint

$$\frac{(\mathcal{K}[t_m])^2}{\mathcal{K}'(t_m)} \quad = \quad -ln(1-f) \tag{22}$$

We can verify that we are at a minimum by confirming that the second derivative of the equation for total time is positive at the identified t_m. For a model where $\mathcal{K}(t_m) = K_o + lt_m$, the ideal solution-planning time is

$$t_m^* = \frac{[k^2 - l(k^2 + ln[1-f])]^{\frac{1}{2}} - k}{l} \tag{23}$$

This function tells us that the ideal portion of resources to apply to metareasoning is a simple function of the solution-refinement constant and the desired quality of the goal.

5 The Value of Metareasoning

We can compare the relative optimality of agents of different constitutions and capabilities, as well as the effectiveness of different policies for reasoning and metareasoning. As an example, we can determine the value of adding a solution-planning capability to systems that ideally apply flexible reasoning strategies. For a single case, the value of adding a solution-planning capability for agents, modeled by the inverse-power models presented in Sections 3.2 and 3.3, is just the difference between the ideal value available with and without the additional planning capabilities. We can generalize our analysis, from comparisons of policies and capabilities for specific instances, to the value of an agent over time. We assume that an agent is immersed in an environment for some length of time (e.g., the expected lifetime of the agent).

In considering a set of challenges, it can be essential to consider probability distributions that describe the difficulty of problems and the cost associated with different challenges. Let us assume an *independent-challenge model*, asserting that solutions to individual challenges do not significantly interact [6]. Assume that $\mathcal{A}_1$ is an agent with ideal reflection and solution planning, $\mathcal{A}_2$ is an agent with an ideal-reflection policy, and $\mathcal{I}_i$ is the problem instance at hand. Each problem $\mathcal{I}_i$ is associated with a tuple (c_i, k_i). k_i is an expected base problem-solution rate (k_o) for problem i. Harder problems are associated with smaller k_i and, thus, are solved more slowly. c_i is a linear cost constant (or cost function) incurred with delay in addressing problem $\mathcal{I}_i$. The utility gains of adding a planning capability to our agent with an ideal reflection capability, can be captured by substituting these constants into the solution of the ideal comprehensive utilities the analyses in Sections 3.2 and 3.3.[3] For an environment described by a probability distribution, $p(i)$, and where the frequency that problems challenge an agent is $\mathcal{F}$,

$$\Delta v_a[p(\mathcal{I}_i), \mathcal{F}, t] \;\; = \;\; t\mathcal{F} \int [u_o^*(\mathcal{A}_1, \mathcal{I}_i) - u_o^*(\mathcal{A}_2, \mathcal{I}_i)]\, p(\mathcal{I}_i) di$$

6 Summary and Conclusions

We have constructed and analyzed models that capture fundamental relationships between metareasoning and base-level problem solving. In particular, we exercised general functional forms, relating metareasoning and execution efficiency for utility-directed and goal-directed computation. We identified solutions to metareasoning-partition problems as functions of a small number of constant parameters of the models. The models and solutions for the ideal apportionment of resources to solution planning and execution suggest how knowledge about the efficacy of solution planning can be used to endow agents with an ability to efficiently custom-tailor their use of resources. Finally, we discussed the value of metareasoning in an environment characterized by a distribution over problems, each associated with a difficulty and a cost. Empirical work is underway, within the realm of belief-network reformulation, to acquire knowledge about the efficacy of solution planning, and to represent that knowledge with specific parameterizations of our resource-partition models.

Acknowledgments

We thank Greg Cooper, Tom Dean, and Mathew Ginsberg for stimulating discussions on metareasoning.

References

[1] J.A. Barnett. How much is control knowledge worth? *Journal of Artificial Intelligence*, 22(1):77–89, January 1984.

[3] For simplicity, we do not include time-discounting of future gains and losses here; time discounting can be captured by including a factor that discounts future gains as a function of time.

[2] M. Boddy and T. Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh IJCAI*. AAAI/International Joint Conferences on Artificial Intelligence, August 1989.

[3] J.S. Breese and E.J. Horvitz. Principles of problem reformulation under uncertainty. Technical report, Knowledge Systems Laboratory, Stanford University, February 1990. KSL-90-27.

[4] J.S. Breese and E.J. Horvitz. Reformulating and solving belief networks under bounded resources. Technical report, Knowledge Systems Laboratory, Stanford University, March 1990. KSL-90-28.

[5] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings AAAI-88 Seventh National Conference on Artificial Intelligence*, pages 49–54. American Association for Artificial Intelligence, August 1988.

[6] E.J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of Third Workshop on Uncertainty in Artificial Intelligence*, Seattle, Washington, July 1987. American Association for Artificial Intelligence. Also in L. Kanal, T. Levitt, and J. Lemmer, ed., *Uncertainty in Artificial Intelligence 3,* Elsevier, 1989, pps. 301-324.

[7] E.J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings AAAI-88 Seventh National Conference on Artificial Intelligence*, pages 111–116. American Association for Artificial Intelligence, August 1988.

[8] E.J. Horvitz, J.S. Breese, and M. Henrion. Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, 2:247–302, 1988. Special Issue on Uncertain Reasoning.

[9] E.J. Horvitz, G.F. Cooper, and D.E. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the Eleventh IJCAI*, pages 1121–1127. AAAI/International Joint Conferences on Artificial Intelligence, August 1989.

[10] E.J. Horvitz, H.J. Suermondt, and G.F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of Fifth Workshop on Uncertainty in Artificial Intelligence*, Windsor, Canada, August 1989. American Association for Artificial Intelligence.

[11] S.J. Russell and E.H. Wefald. Principles of metareasoning. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, May 1989. Morgan Kaufman.

[12] S. Shekhar and S. Dutta. Minimizing response times in real time planning and search. In *Proceedings of the Eleventh IJCAI*, pages 238–242. AAAI/International Joint Conferences on Artificial Intelligence, August 1989.

[13] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ, 1947.

Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 6382765, 11 pages
http://dx.doi.org/10.1155/2016/6382765

Research Article

An Efficient Technique for Hardware/Software Partitioning Process in Codesign

Imene Mhadhbi, Slim Ben Othman, and Slim Ben Saoud

Department of Electrical Engineering, National Institute of Applied Sciences and Technology, Polytechnic School of Tunisia, Advanced Systems Laboratory, B.P. 676, 1080 Tunis Cedex, Tunisia

Correspondence should be addressed to Imene Mhadhbi; imene.mhadhbi@gmail.com

Received 27 January 2016; Accepted 10 May 2016

Academic Editor: Michele Risi

Codesign methodology deals with the problem of designing complex embedded systems, where automatic hardware/software partitioning is one key issue. The research efforts in this issue are focused on exploring new automatic partitioning methods which consider only binary or extended partitioning problems. The main contribution of this paper is to propose a hybrid FCMPSO partitioning technique, based on Fuzzy C-Means (FCM) and Particle Swarm Optimization (PSO) algorithms suitable for mapping embedded applications for both binary and multicores target architecture. Our FCMPSO optimization technique has been compared using different graphical models with a large number of instances. Performance analysis reveals that FCMPSO outperforms PSO algorithm as well as the Genetic Algorithm (GA), Simulated Annealing (SA), Ant Colony Optimization (ACO), and FCM standard metaheuristic based techniques and also hybrid solutions including PSO then GA, GA then SA, GA then ACO, ACO then SA, FCM then GA, FCM then SA, and finally ACO followed by FCM.

1. Introduction

The hardware/software partitioning process presents the crucial task of the codesign methodology. It is concerned to decide which functions are to be implemented in hardware components and which ones in software components. This partitioning process aims at finding an optimal trade-off between conflicting requirements to improve the system performance.

Recently, different optimization methods have been undertaken to automate the hardware/software partitioning process.

These optimization methods can be split into exact and heuristic methods. The exact methods, such as Integer Linear Programing (ILP) [1], dynamic programming [2, 3], and branch-and-bound [4], work effectively for smaller graph with several tens of nodes. However, the heuristic methods produce near-optimal solutions even for larger inputs. The heuristic methods can, also, be iterative or constructive. The iterative methods such as PSO [5], Genetic Algorithm (GA) [6], Ant Colony Optimization (ACO) [7], Simulated Annealing (SA) [8], Fiduccia-Matteyeses [9], Kernighan/Lin [10], and Tabu Search (TS) [11] attempt to modify a given solution until no improvement can be done. However, the constructive methods, such as greedy and hierarchical clustering [12], generate a small number of solutions starting from an initial partitioning by selecting and adding components to the partial solution until a complete solution is obtained.

Designers of embedded systems focus on achieving more optimal partitioning solutions by emphasizing a combination between existing optimization methods. They propose to combine partitioning algorithms in order to generate optimal solutions of partitioning in a reduced time. In the literature, designers focus on combining the GA and the TS algorithms [13], the PSO and the Branch-and-Bound algorithms [14], the GA and the SA algorithms [15], and the GA and the PSO algorithms [16]. The given results prove that these combinations produce more accurate solutions than the classical algorithms in terms of cost and execution time metrics. In [17], the authors consider the reliability as a factor when solving the partitioning problem, in addition to the cost and time metrics. They propose to combine the recursive and the linear programming algorithms.

Constructive algorithms are usually suggested to be integrated with iterative algorithms to increase the quality of the generated solution. For example, the authors in [18] propose an algorithm based on clustering algorithm to make the GA algorithm better in bigger-scale embedded system. The proposed algorithm overcomes the shortcoming that GA algorithm execution time is too long to achieve good results in system partitioning.

In this work, a new hybrid method combining clustering FCM algorithm and the PSO algorithm called "FCMPSO" algorithm is proposed. Experimental results indicate that the FCMPSO algorithm is superior to GA, SA, ACO, FCM, and PSO standard algorithms and PSO-GA, GA-SA, GA-ACO, ACO-SA, FCM-GA, FCM-SA, and ACO-FCM hybrid techniques for both binary and extended partitioning approaches.

This paper is organized as follows. In Section 2, related works of the hardware/software partitioning techniques are introduced. The constructed benchmarking scenario model definition for partitioning problem is described in Section 3. In Sections 4 and 5, the formulation of hardware/software partitioning problems as a binary and then as an extended approach is presented. Experimental results and comparison of the proposed FCMPSO algorithm with standards partitioning techniques and hybrid ones are discussed in Section 6. Finally, the paper concludes in Section 7 by briefing the present work.

2. The Used Optimization Algorithms to Solve Partitioning Problems

This section provides some detailed notations and definitions of the PSO algorithm, the FCM algorithm, and our proposed FCMPSO algorithm.

2.1. PSO Algorithm. PSO is a stochastic, iterative population-based evolutionary optimization algorithm. It was developed in 1999 by Shi and Eberhart [19]. It uses the swarm intelligence that is based on social-psychological and biological social principles. By equivalence with the swarm intelligence, each swarm member (particle) takes advantage of private memory and has a degree of randomness in its movement as well as knowledge gained by the whole swarm to discover the best available food source. The problem of a food search can be solved by optimizing a fitness function. The definition of the communication structure (or social network) is obtained by assigning the neighbors for each swarm. All particles, in the search space, have fitness values which are evaluated by the fitness function to be optimized and have velocities which direct their motion in the multidimensional search space. Each particle remembers the information about its best solution and its position in the search space and both are available to its neighbors. In order to update the appropriate changes of its position and velocity, each particle p has a memory holding: the particle "p_{best}" position which presents the best solution the particle has seen by itself and the global best particle location's "g_{best}" that the particle acquires by communicating with a subset of swarms. The ith particle velocity $V_i(t)$ and position $X_i(t)$ updates are based on the following equations:

$$V_i(t+1) = wV_i(t) + C_1 r_1 \left(p_{\text{best}}(i,t) - X_i(t)\right) + C_2 r_2 \left(g_{\text{best}}(t) - X_i(t)\right), \quad (1)$$

$$X_i(t+1) = X_i(t) + V_i(t+1), \quad (2)$$

where w is the inertia factor that takes linearly decreasing values downward from 1 to 0 according to a predefined number of iterations, $V_i(t)$ is the velocity, $X_i(t)$ is the current solution (or position), r_1 and r_2 are uniform random numbers in the range between 0 and 1, and C_1 and C_2 present positive constant parameters called "acceleration coefficient."

2.2. FCM Algorithm. FCM algorithm is a determinist, constructive optimization algorithm. Different studies prove that the FCM outperforms different existing clustering algorithms, that is, the Self-Organization Map (SOM) neural network algorithm [20], k-mean algorithm [21], and hierarchical clustering [20]. It is the most popular fuzzy clustering method, which was originally proposed by Dunn [22] and later had been modified by James [23].

FCM algorithm is efficient, straightforward, and easy to implement. It is based on fuzzy behavior and provides a natural technique for producing clustering where membership weights have a natural interpretation but not probabilistic (determinist). The main goal of the FCM is to minimize an objective function, taking into account the similarity of elements and cluster centers.

Suppose $\Omega = \{1, \ldots, k, \ldots, N\}$ a set of N objects in R^d dimensional space, listed by k. Each object k is represented by a vector of quantitative variables $X_k = \{X_{1k}, \ldots, X_{ik}, \ldots, X_{jk}\}$ defined by j variables indexed by i, where $X_{ik} \in \mathbb{R}$. Suppose $Y = \{1, \ldots, p, \ldots, C\}$ a set of C prototypes listed by p associated with C groups, where each prototype p is also represented as a vector of quantitative variables $Y_p = \{Y_{1p}, \ldots, Y_{ip}, \ldots, Y_{jp}\}$, where $Y_{ji} \in \mathbb{R}$. Suppose $U = |U_{pk}|$ a $C \times N$ matrix of membership degrees, where U_{pk} presents the membership degree of the object k to group p. Its value belongs to the real interval $[0, 1]$.

FCM algorithm aims at finding a prototype matrix Y and a membership degree matrix U that minimize $J(Y, U)$: the objective function called "fitness function." The prototypes that minimize the objective function are updated using the following equation:

$$Y_p = \frac{\sum_{k=1}^{N} \left[\left(U_{pk}\right)^m X_k\right]}{\sum_{k=1}^{N} \left(U_{pk}\right)^m}. \quad (3)$$

The membership degrees that minimize the objective function are updated according to the following equation:

$$U_{pk} = \left(\sum_{i=1}^{C} \left(\frac{\|X_k - V_p\|}{\|X_k - V_i\|} \right)^{1/(m-1)} \right)^{-1}, \quad (4)$$

where m is the level of cluster fuzziness. In the limit $m = 1$, the membership degree converges to 0 or 1, which implies a good partitioning.

FCM algorithm is an effective algorithm. It is faster than the PSO algorithm because it requires fewer function evaluations, but it is sensitive to initial values and usually falls into local optima. The weaknesses of these two algorithms motivate the proposal of an alternative approach based on the combination of FCM and PSO algorithms to form a novel FCMPSO algorithm which maintains the merits of both PSO and FCM algorithms.

2.3. FCMPSO Algorithm. In this work, FCMPSO algorithm was proposed to solve both binary and extended hardware/software partitioning problems. This algorithm takes together advantages of both PSO and FCM algorithms: the PSO algorithm has a strong global search capability, while FCM algorithm produces approximate solutions faster and fails in a local optimal solution easily. Hence, we integrated the FCM algorithm with PSO algorithm to provide near-optimal solution with faster speed.

Firstly, we applied the FCM algorithm to create the uncertain initial partitioning solutions in order to reduce (limit) the research space of the PSO algorithm. Then, we execute the PSO algorithm to have a near-optimal partitioning solution.

The pseudocode of our FCMPSO algorithm is presented in Algorithm 1.

Algorithm 1 (FCMPSO algorithm).

g_{best} = FCMPSO (Ω, C)

Require:

Dataset of FCM parameters:

(1) Select the center of cluster C

(2) Select the number of objects N

(3) Select the maximum number of iterations T

(4) Select the level of cluster fuzziness ($m = 2$)

(5) Randomly initialize U_{pk} ($p = 1, \ldots, C$ and $k = 1, \ldots, N$) of object k to group p

(6) $t \leftarrow 0$; $J(t) \leftarrow 0$; $J(t+1) \leftarrow$ Partitioning Objective Function (U_{pk})

Dataset of PSO parameters:

(1) Select the maximum number of iterations T

(2) Select the population (swarm) size: N

(3) Select the inertia factor "w" that takes linearly decreasing values downward from 1 to 0

(4) Select the uniform random numbers r_1 and r_2 in the range between 0 and 1

(5) Select the acceleration coefficients C_1 and C_2 (positive constant parameters)

FCM algorithm

Repeatedly, *while* $|J(t) - J(t+1)| > \varepsilon$ and $t < T$

(a) Update prototype matrix Y: fix the membership degree matrix U_{pk} and update prototypes using (3)

(b) Update the membership degree matrix U: fix the membership degrees using (4)

(c) $J(t) \leftarrow J(t+1)$

(d) $J(t+1) \leftarrow$ Partitioning Objective Function (U_{pk})

(e) $t \leftarrow t + 1$

end while

Return Matrices **Y** and **U**

PSO algorithm

(1) Initialize particle position X with Y matrix of FCM output algorithm

For each particle position X to D (dimension of Y)

(2) Compute the fitness values of X_i

(3) Compute velocity

(4) Initialize p_{best} to its initial position: $p_{\text{best}} = X_i$

(5) Initialize g_{best} to the minimal value of the swarm: g_{best} = fitness value (p_{best})

End For

Repeat until criteria are met ($t \geq T$)

(1) Update particle's velocity as (1)

(2) Update particle's position as (2)

If $f[X_i(t)] < f[p_{\text{best}}(I, t)]$,

(1) Update the best know position of particle i: $p_{\text{best}}(i) = X_i(t)$

If $f[X_i(t)] < f[g_{\text{best}}(t)]$,

(1) Update the swarm's best position: $g_{\text{best}}(t) = X_i(t)$

(2) $t \leftarrow t + 1$

end

return $g_{\text{best}}(t)$

The complexity analysis of the original PSO and FCM algorithm is as follows:

(i) The PSO algorithm is $O(N + 2NT)$, where N is the swarm population and T is the maximum number of iterations.

(ii) The FCM algorithm is $O(NCT)$, where N is the objects number, T is the maximum number of iterations, and C is the cluster center. In our case, $C = 1$.

(iii) The FCMPSO algorithm is $O(D + 2DT + NT)$, where D is the dimension of the population issue from the FCM algorithm, N is the objects number, and T is the maximum number of iterations.

As can be seen, the computational complexity of the FCMPSO algorithm is mainly affected by the number of objects N and the population dimension issue from the FCM algorithm. The disadvantages of the PSO algorithm are that it is easy to fall into local optima in high-population dimension and has a low convergence rate in the iterative process. It can also be observed that the computational complexity of the

hybrid FCMPSO is accepted when it is applied to solve the high-dimensional and complex problems.

Our proposed FCMPSO algorithm will be tested for two kinds of partitioning approaches: binary and extended ones. The main difference between these two kinds of architectures appears in the number of the used devices and their types. In the binary partitioning approach, the target architecture includes a single hardware processing unit and a single software processing unit or a reconfigurable architecture. However, in extended partitioning approach, the target architecture includes multiprocessing hardware components and several software processing components.

Before starting the hardware/software partitioning process, it is necessary to transform the initial specification into formal specification. The benchmarking scenario model used to validate our proposed hardware/software partitioning problem is presented in the next section.

3. Benchmarking Scenario: Task Graphs

Different benchmarks and applications are used to validate hardware/software partitioning approaches. These applications and benchmarks are varying from each other. The embedded application to be partitioned is generally given as a Direct Acyclic Graph (DAG) that represents the sequence of nodes in the embedded system application.

In this work, we use Task Graph for Free (TGFF) tool to generate a set of 20, 50, 100, 200, 500, 1000, and 2000 nodes graphs. Each graph is donated as $G(V, E)$, where $V = \{V_1, \ldots, V_n\}$ presents the set of tasks and $E\ \{E_{i,j} \mid 1 \leq i, j \leq n\}$ are the set of edges which present the data dependency between two nodes. The partitioning process aims to find a partition P, where $P = (V_H, V_S)$ such that $V_H \cup V_S = V$ and $V_H \cap V_S = \emptyset$. It can generate a deciding partition vector $X = \{X_1, X_2, \ldots, X_N\}$, representing the implementation way of the N task nodes.

Such approach is necessary to avoid the system architecture dependency variation in the system by making parameter changes. The variations in the number of the input/output nodes, the node metrics (i.e., execution time, cost, area, and power), and the software/hardware processor number are randomly assigned in the TGFF file input configuration file.

The obtained graphs are used as a system specification to validate the efficiency of our proposed partitioning algorithm to solve both binary and extended partitioning problems.

4. Hardware/Software Partitioning as a Binary Problem

This section provides the formal description of the binary partitioning problem, especially the used target architecture and the mathematical model of the objective function and the related constraints.

4.1. Architecture Representation. The characteristic of the target system architecture consists of one software processor to execute software tasks and one hardware component (FPGA/ASIC) to implement hardware tasks. Both software and hardware components have their local memory and communicate with each other through a shared bus for communication between hardware and software components, as shown in Figure 1.

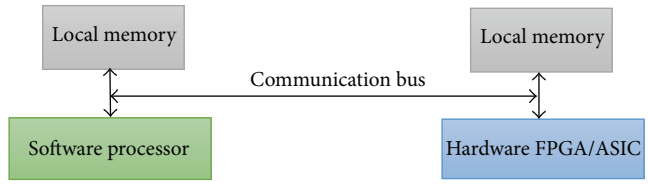


Figure 1: Target architecture of binary partitioning approach.

4.2. Model Formulation. In the binary partitioning approach, each node value must take a value of 0 or 1, where value is 1 if the node value is assigned to a hardware component; otherwise, it is 0 indicating that the task is assigned to a software component. Each node in the DAG is associated with cost parameters, that are (i) software costs (software execution time T_S, memory requirement M), (ii) hardware costs (hardware execution time T_H, used slices rate constraint S_H), and (iii) communication cost (C_{SH}). The last cost refers to the delay required to transfer data from the hardware node to software node and vice versa.

Given identical parameters and input speeds, TGFF can generate identical task graphs. These random DAG graphs allow representation of applications by using input parameters as follows.

(1) Target Architecture. One software processor and one hardware processor related over communication bus.

(2) Software Constraints. It is presented as "software execution time" constraint (T_S) fixed between 200 and 400 (μs) and "memory" constraint (M) fixed between 0 and 20 MB.

(3) Hardware Cost. It is presented as "hardware execution time" constraint (T_H) fixed between 75 and 225 (μs) and "used slice rate" constraint (S_H) fixed between 50 and 150 (slices).

The parameters generated from the TGFF input files for our three DAGs graphs are presented in Table 1.

In Table 1, AllTimeSw means the time when all nodes are implemented in software, while AllTimeHw means the time when all nodes are implemented in hardware. AllCostSw means the memory requirement when all nodes are implemented in software. AllCostHw means the hardware resource utilization when all nodes are implemented in hardware.

The communication costs required between hardware and software tasks are much less than the task processing time; it can be neglected for simplicity.

4.3. Mathematical Constraints Formulation. During the hardware/software partitioning process, based on the constraint definition in the previous subsection, the total hardware/software execution time (T_H/T_S), the used slices rate constraint (S_H), and the memory requirement (M) can be formalized as follows.

Table 1: The parameters and costs of the DAGs generated graphs.

Nodes	20 nodes	50 nodes	100 nodes	200 nodes	500 nodes	1000 nodes	2000 nodes
AllTimeSw	5761	18526	28476	58162	4230	283800	564796
AllCostSw	190	403	949	1745	142338	8463	16719
AllTimeHw	2821	7151	13842	28626	45920	139221	272813
AllCostHw	1862	4532	9727	18737	69260	92294	183506

(i) Execution Time Constraint. T_S and T_H present the execution time of, respectively, software and hardware implemented solution. They can be expressed as follows:

$$T_S(X) = \sum_{i=1}^{N} T_i^S * (1 - X_i),$$
$$T_H(X) = \sum_{i=1}^{N} T_i^H * X_i, \tag{5}$$

where N is the total number of tasks in the system and T_i^S and T_i^H are the software and hardware execution time, respectively, of each ith task. X_i presents a binary variable. Its value is 0 if the i task is assigned to a software component or 1 if the ith task is assigned to a hardware component.

(ii) The Memory Requirement. M presents the memory requirement only for components assigned to software architecture. The total memory is obtained as follows:

$$M(X) = \sum_{i=1}^{N} C_i^{Sw} * (1 - X_i), \tag{6}$$

where C_i^{Sw} is the software cost for a ith task.

(iii) The Used Slices Rate. S_H presents the number of slices that a partitioning solution will use in a particular hardware component. The total used slices rate is obtained as follows:

$$S_H(X) = \sum_{i=1}^{N} C_i^{Hw} * X_i, \tag{7}$$

where C_i^{Hw} is the hardware cost of the ith task.

Partitioning algorithms try to find a trade-off between these conflicting constraints to improve the system performance. It can be modeled as the minimization of the objective function (F.F.) as follows:

$$\text{F.F.}(\%) = 100 * \left(\frac{T_S}{\text{AllTimeSw}} + \frac{T_H}{\text{AllTimeHw}} + \frac{M}{\text{AllCostSw}} + \frac{SR}{\text{AllCostHw}} \right). \tag{8}$$

The target architecture was generally assumed to consist of only one software and only one hardware unit. In recent years, many researchers are committed to resolve the extended hardware/software partitioning problems: for multiprocessor systems with a high quality solution. In the next section, we will introduce the formal description of the extended hardware/software partitioning problem proposed in this work.

5. Hardware/Software Partitioning as an Extended Problem

This section provides the formal description of the extended architecture.

5.1. Architecture Representation. The architectural model of heterogeneous multiprocessors considered, in this section, consists of two general purpose processors and two application-specific components, denoted by W = {Sw1, Sw2, Hw1, Hw2}. Each processor has its local memory (LM) used for the intertasks communication on the same processor. Data in a shared memory is acceded by both hardware and software parts, as presented on Figure 2.

5.2. Model Formulation. In the extended partitioning problem, a node must take values between 0 and N, where N presents the number of processing units. Software nodes are performed by Sw1 and Sw2. The data communication time between tasks is much less than task processing time, and it can be omitted for simplicity. However, hardware nodes are implemented on FPGA or ASIC. Each node is associated with cost parameters that are as follows:

(i) Hardware/software execution time ($T_{Hw1}/T_{Hw2}/T_{Sw1}/T_{Sw2}$).

(ii) Used slices rate constraint (S_{H1}/S_{H2}).

(iii) Memory requirement (M_{S1}/M_{S2}) and communication time (C_{SH}).

A 20 nodes' random task graph is generated based on the TGFF using parameters denoted in Table 2.

In this work, the communication time (C_{SH}) taken between two different components was neglected.

5.3. Mathematical Constraints Formulation. In this work, the fitness function (F.F.) to minimize is defined as follows:

$$\text{F.F.}(\%) = 100 * \left(\frac{T_{Sw1}}{\text{AllTimeSw1}} + \frac{T_{Sw2}}{\text{AllTimeSw2}} + \frac{T_{Hw1}}{\text{AllTimeHw1}} + \frac{T_{Hw2}}{\text{AllTimeHw2}} + \frac{SR_{H1}}{\text{AllCostHw1}} + \frac{SR_{H2}}{\text{AllCostHw2}} + \frac{M_{S1}}{\text{AllCostSw1}} + \frac{M_{S2}}{\text{AllCostSw2}} \right), \tag{9}$$

where we define the following.

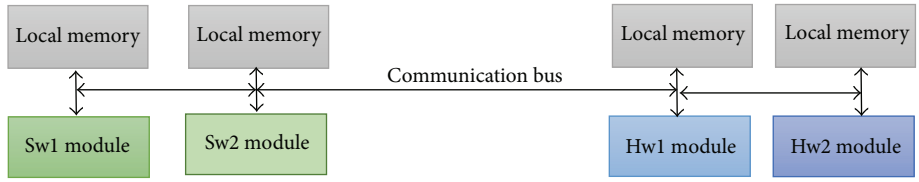


Figure 2: Target architecture of extended partitioning approach.

Table 2: TGFF parameters of partition 20 nodes with multicores architecture.

W	Node assignation	TGFF parameters		
		Execution time (sec.)	Used slices rate	Memory requirement (MB)
Sw1	"00"	T_{Sw1} [200, 400]	—	M_{S1} [0, 20]
Sw2	"01"	T_{Sw2} [100, 300]	—	M_{S2} [30, 50]
Hw1	"10"	T_{Hw1} [75, 225]	S_{H1} [50, 150]	—
Hw2	"11"	T_{Hw2} [50, 150]	S_{H2} [60, 100]	—

(i) Execution Time. T presents the time taken at each node. It can be expressed as follows:

$$\begin{aligned} T_{Sw1} &= \sum_{i=1}^{N} T_i^{Sw1} \quad \text{if } X_i = \text{"00"}, \\ T_{Sw2} &= \sum_{i=1}^{N} T_i^{Sw2} \quad \text{if } X_i = \text{"01"}, \\ T_{Hw1} &= \sum_{i=1}^{N} T_i^{Hw1} \quad \text{if } X_i = \text{"10"}, \\ T_{Hw2} &= \sum_{i=1}^{N} T_i^{Hw2} \quad \text{if } X_i = \text{"11"}, \end{aligned} \tag{10}$$

where N is number of processors. X_i presents a binary variable whose value is "00" if the i task is assigned to a Sw1 component, "01" if the i task is assigned to a Sw2 component, "10" if the i task is assigned to a Hw1 component, and "11" if the i task is assigned to a Hw2 component. T_i is the execution time for the ith task.

(ii) Used Slices Rate. S_H presents the number of slices that a partitioning solution will use in a particular hardware component. The total used slices rate is obtained as follows:

$$\begin{aligned} S_{H1} &= \sum_{i=1}^{N} C_i^{Hw1} \quad \text{if } X_i = \text{"10"}, \\ S_{H2} &= \sum_{i=1}^{N} C_i^{Hw2} \quad \text{if } X_i = \text{"11"}, \end{aligned} \tag{11}$$

where N is the total number of tasks in the system and C_i^{Hw} is the hardware cost for the ith task.

(iii) The Memory Requirement. M_S presents the memory requirement only for components assigned to software architecture. The total memory is obtained as follows:

$$\begin{aligned} M_{S1} &= \sum_{i=1}^{N} C_i^{Sw1} \quad \text{if } X_i = \text{"00"}, \\ M_{S2} &= \sum_{i=1}^{N} C_i^{Sw2} \quad \text{if } X_i = \text{"01"}, \end{aligned} \tag{12}$$

where C_i^{Sw} is the software cost for a ith task.

6. Empirical Results and Discussions

In this paper, we will evaluate the performance of the proposed FCMPSO algorithm in both binary and extended architectures. The experiments are performed on an Intel Celeron CPU having 2.16 GHz processor speed and 2 GHz RAM. Hardware/software optimization algorithms were coded in Matlab environment and they are executed in Windows 7 operating system.

6.1. Empirical Results and Discussion for Binary Partitioning Approach. The binary partitioning problem was solved considering different task graphs ranging from 20 to 2000 nodes. The values of software execution time (T_S), hardware execution time (T_H), memory requirement (M), and used slices rate (S_H) were generated randomly as described in Section 4.2.

First, we have proceed to simulate standards PSO, ACO, GA, and FCM and SA algorithms were performed. The aim is to determine the best parameters of each algorithm that can give the better solutions and compare the partitioning results. We consider the following heuristics algorithms parameters:

(i) For SA algorithm, the parameters used were as follows: initial temperature = 10, final temperature = 0, and $\alpha \geq 0.93$.

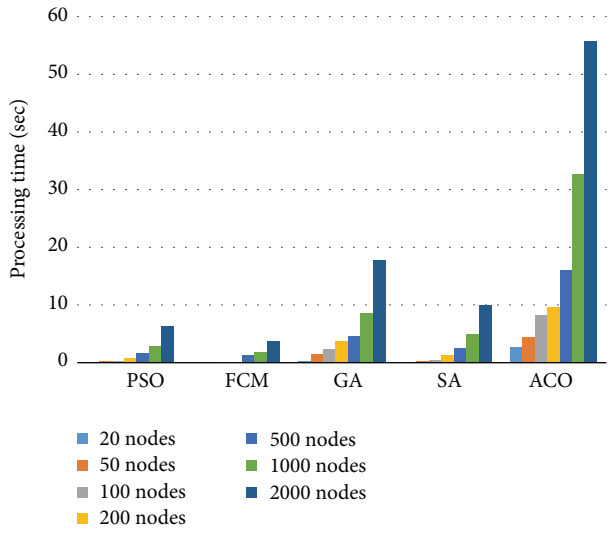


Figure 3: Processing time using standards partitioning algorithms (sec).

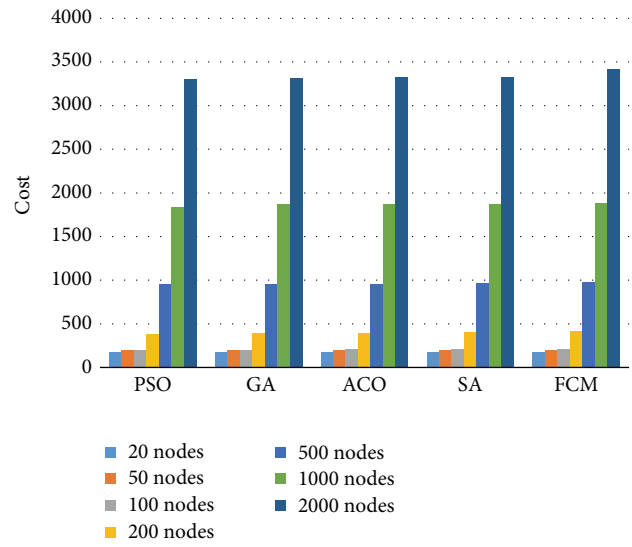


Figure 4: Costs percentage using standards partitioning algorithms.

(ii) For GA algorithm, the parameters used were as follows: population of 100 individuals, selection rate = 0.5, mutation rate = 0.2, and crossover rate = 0.5.

(iii) For PSO algorithm, the parameters used were as follows: population of 100 individuals, $C_1 \geq C_2 \geq 1$, and inertia weight $w = 0.1$.

(iv) For ACO algorithm, the parameters used were as follows: the evaporation parameter is 0.95 and the positive and negative pheromone were 0.06 and 0.3, respectively.

(v) For FCM algorithm, the parameters used were as follows: $U = [0, 2^N]$, cluster center $C = 1$, and the scalar termed $m = 2$, where N presents the number of nodes in the task graph.

Each algorithm was executed 10 times. In each execution time, the evaluations of the objective function were made 100 times. The best solutions were always taken. Performance analysis for both processing time and quality of the cost solution found by the considered algorithms is given in Figures 3 and 4, respectively.

The processing time comparison of standards partitioning algorithms, presented in Figure 3, proves that FCM algorithm is faster than PSO, SA, GA, and ACO algorithms because it requires fewer function evaluations. Figure 3 demonstrates also that the PSO algorithm takes a considerable time comparing to FCM algorithm to discover and evaluate the input random particles because of the large input swarm (50 particles).

Moreover, Figure 4 reveals that the quality result, which gives a measure of the fitness function cost percentage of solution, is found to be the best for PSO algorithm.

Results demonstrate also that PSO algorithm has a strong global search capability, while the FCM algorithm generates approximate solutions faster. Hence, a combination between the FCM and PSO algorithms allows the generation of a near-optimal solution with faster speed.

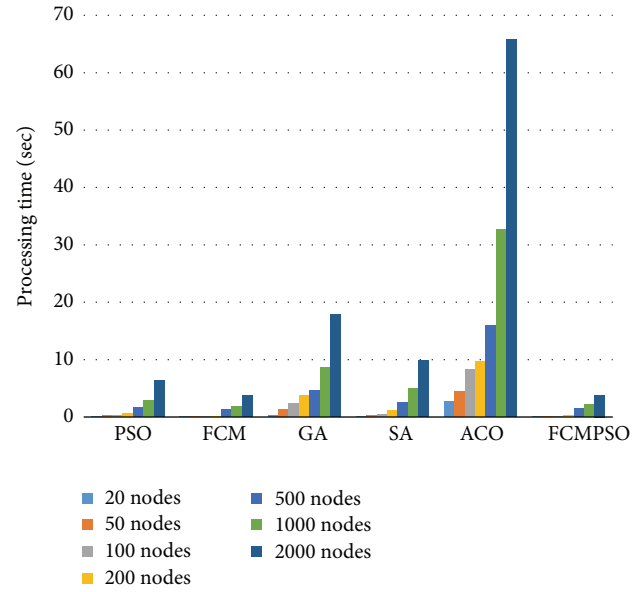


Figure 5: Comparison of processing time of FCMPSO algorithm with standards algorithms.

The simulation results of the proposed FCMPSO algorithm as well as the original FCM, PSO, ACO, GA, and ACO algorithms are presented in Figures 5 and 6.

Figure 5 reveals that the FCMPSO algorithm enhances the processing time convergence of the PSO algorithm when combining it with FCM clustering algorithm by minimizing its input population number and limiting its "local search" space.

The cost comparison provided in Figure 6 reveals the efficiency of the proposed algorithm to generate the optimal solutions comparing with the original PSO and FCM algorithms. We can also observe that the cost of generated

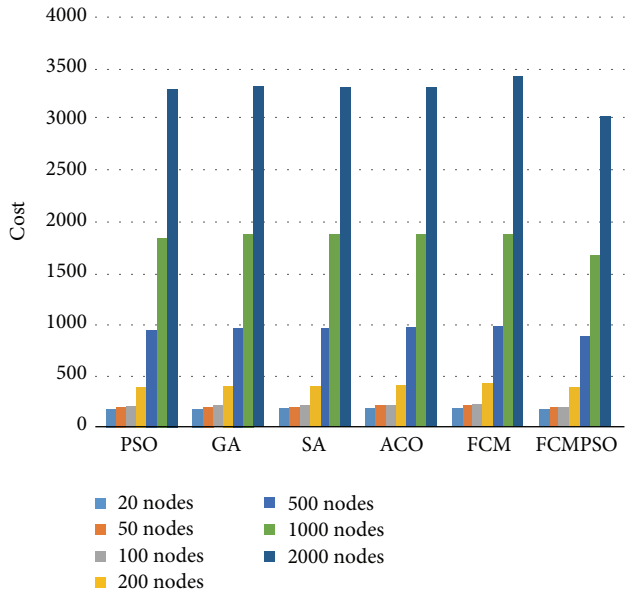


Figure 6: Comparison of cost percentage of FCMPSO algorithm with standards algorithms.

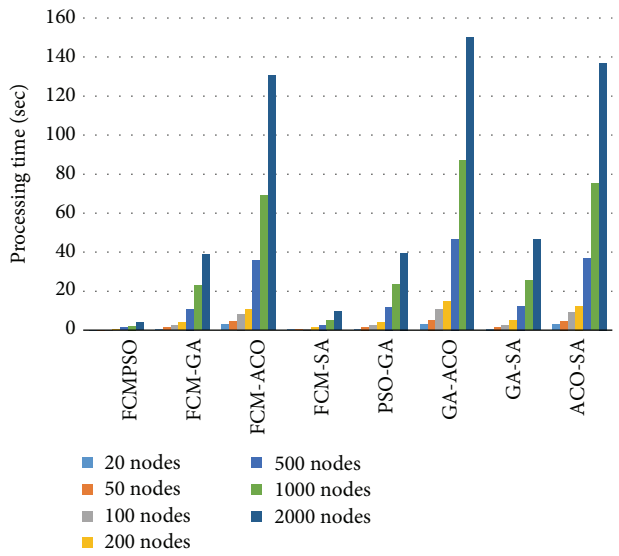


Figure 7: Comparison of processing time of FCMPSO algorithm with hybrid techniques.

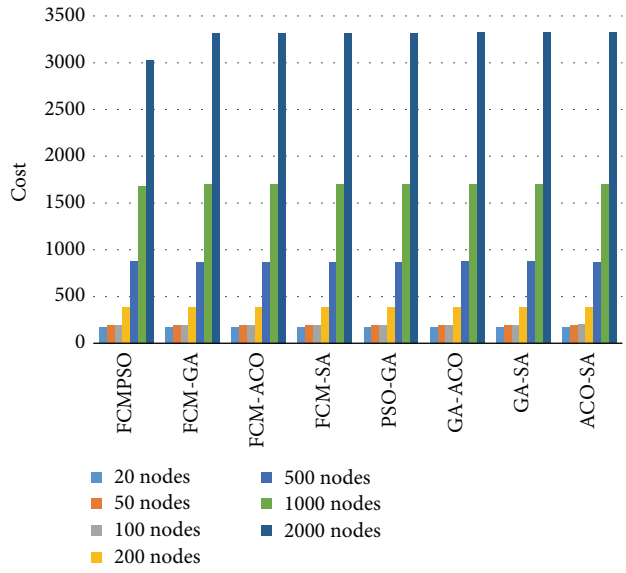


Figure 8: Comparison of cost percentage of FCMPSO algorithm with hybrid techniques.

solutions becomes more and more close when the number of nodes decreases within 2000 nodes' graph size; the variation between the FCMPSO cost and the FCM cost presents 11.11% while it is less than 4% with 20 nodes' graph size. This variation is due to the role that FCM algorithm plays in improving the convergence of the PSO by minimizing its "local search input population" to generate an improved optimal solution.

However, FCMPSO algorithm obtains significant improvements over FCM and PSO algorithms in terms of processing time and generated cost solution in binary partitioning approach.

To measure the efficiency of combining FCM and PSO algorithms, we proposed to compare it to hybrid combinations including PSO then GA, GA then SA, GA then ACO, ACO then SA, FCM then GA, FCM then SA, and finally ACO followed by FCM. Simulated results are shown in Figures 7 and 8.

the algorithm processing time of FCMPSO is 0.033 seconds while that of FCM-SA is 0.24 seconds. This result improves that around 86% of speed reduction is in favor of FCMPSO.

As shown in Figures 9, 10, 11, and 12, the best cost of FCMPSO is 170.19 while it is 170.61 for FCM-GA for 20 nodes' graphs. This result represents around 0.25% improvement in the result quality in favor of FCM-GA.

Recently, many approaches are committed to researching how to improve the performance of hardware/software partitioning in multiprocessor systems. Different solutions have been developed. In the next section, we will prove the efficiency of our proposed algorithm in an extended partitioning approach.

6.2. Empirical Results and Discussion for Extended Partitioning Approach. Extended partitioning approach tests were executed on a task graph containing 20 nodes whose parameters are described in Section 5.2. These nodes are extended for partitioning problem on multiprocessors target architecture, composed of two software units and two hardware units. Results are obtained from 10 times of execution. In each execution, the objective functions were evaluated 100 times. At each time, the best solution was used.

The simulation of the PSO, GA, ACO, SA, FCM, and our proposed FCMPSO algorithms allows the generation of individual solution, as illustrated in Table 3.

Simulation performances of both processing time and cost of the generated solution for the PSO, GA, ACO, SA,

Table 3: Generated solutions for extended partitioning approach.

	Generated solutions
FCM	01/11/10/01/01/11/01/10/10/00/11/01/00/01/01/00/00/00/00/00
PSO	01/01/10/01/11/00/00/01/10/01/01/11/00/10/11/11/00/00/00/00
GA	10/01/10/10/11/01/10/01/01/00/00/10/11/01/01/11/10/00/00/00
SA	01/01/11/00/10/01/01/10/01/10/00/10/00/00/11/11/00/00/00/00
ACO	11/10/01/10/00/00/00/01/00/00/10/11/01/01/11/01/00/00/00/00
FCMPSO	10/00/00/00/11/00/00/00/11/00/01/01/11/00/00/11/10/00/00/00

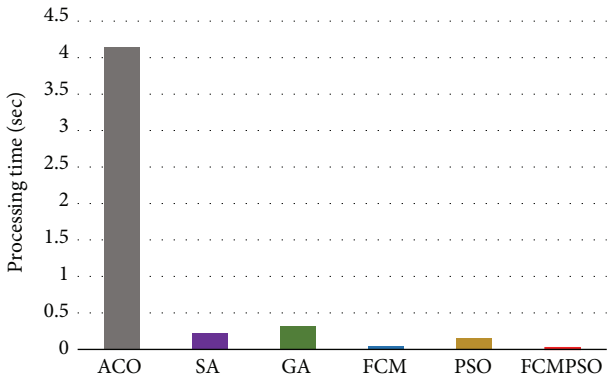


Figure 9: Comparison of processing time of FCMPSO algorithm with standards algorithms.

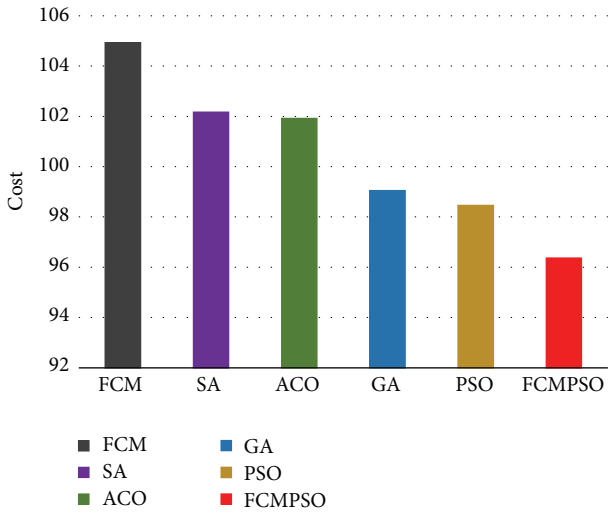


Figure 10: Comparison of cost percentage of FCMPSO algorithm with standards algorithms.

FCM, and our proposed FCMPSO algorithms are shown in Figures 9 and 10.

The obtained experimental results prove, once again, the efficiency of our proposed partitioning algorithm for an extended partitioning approach. Partitioning an embedded application on multiprocessor architecture minimizes the global cost. In fact, in our study, using two hardware units and two software units allows the decrease of global cost by 39.6% for FCM results cost (the worst cost case) and 43.28% for FCMPSO results cost (the best cost case) but took more cumulative run.

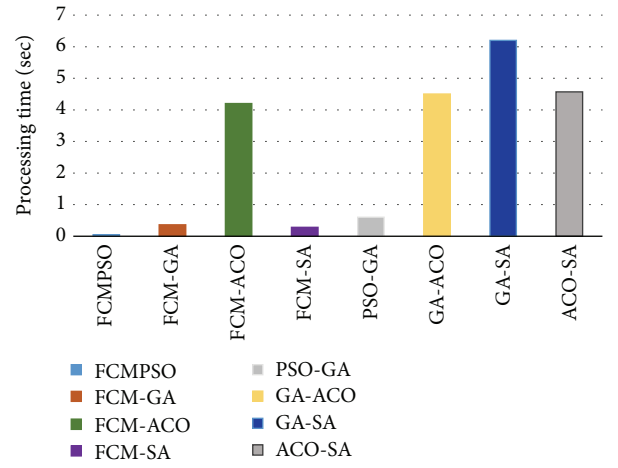


Figure 11: Comparison of processing time of FCMPSO algorithm with hybrid techniques.

To demonstrate the efficiency of our proposed algorithm, we also propose to compare it to some hybrid combinations including PSO then GA, GA then SA, GA then ACO, ACO then SA, FCM then GA, FCM then SA, and finally ACO followed by FCM. Simulated results are shown in Figures 11 and 12.

As expressed in Figures 11 and 12, the best cost was performed by FCMPSO. This result represents around 8% improvement in the result quality in favor of ACO-GA. For the algorithm processing time, the FCMPSO algorithm needs 0.051 sec while ACO-GA algorithm needs 4.57 sec. This result presents around 38% improvement in performance in favor of FCMPSO. Simulation results of both binary and extended partitioning problems demonstrate the efficiency of the FCMPSO algorithms in terms of processing time and generated solution quality.

7. Conclusion

In this paper, the FCM algorithm was integrated with PSO algorithm to form a clustering PSO called "FCMPSO." This solution maintains the merits of both PSO and FCM algorithms, to solve both binary and extended partitioning problems. FCMPSO algorithm applies FCM to the particles in the swarm to improve the generated solution (improve the quality of input swarm) in a fast processing time. We have

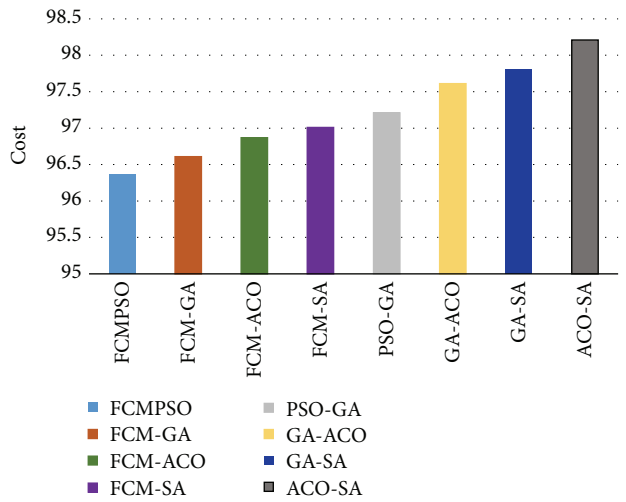


Figure 12: Comparison of cost percentage of FCMPSO algorithm with hybrid techniques.

demonstrated that FCMPSO algorithm can produce better solution with quick search speed than both standards heuristics algorithms and other hybrid partitioning techniques to solve both binary and extended partitioning problems. Our future works will consider the scheduling problem and the communication between different components.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

References

[1] P. Arató, Z. Á. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 1, pp. 136–156, 2005.

[2] M. O'Nils, A. Jantsch, A. Hemani, and H. Tenhunen, "Interactive hardware-software partitioning and memory allocation based on data transfer profiling," in *Proceedings of the International Conference on Recent Advances in Mechatronics (ICRAM '95)*, Istanbul, Turkey, August 1995.

[3] J. Wu and T. Srikanthan, "Low-complex dynamic programming algorithm for hardware/software partitioning," *Information Processing Letters*, vol. 98, no. 2, pp. 41–46, 2006.

[4] K. S. Chatha and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 3, pp. 193–208, 2002.

[5] A. Bhattacharya, A. Konar, S. Das, C. Grosan, and A. Abraham, "Hardware software partitioning problem in embedded system design using particle swarm optimization algorithm," in *Proceedings of the 2nd International Conference on Complex, Intelligent and Software Intensive Systems (CISIS '08)*, pp. 171–176, Barcelona, Spain, March 2008.

[6] P. K. Nath and D. Datta, "Multi-objective hardware-software partitioning of embedded systems: a case study of JPEG encoder," *Applied Soft Computing Journal*, vol. 15, pp. 30–41, 2014.

[7] Z.-H. Xiong, S.-K. Li, and J.-H. Chen, "Hardware/software partitioning based on dynamic combination of genetic algorithm and ant algorithm," *Journal of Software*, vol. 16, no. 4, pp. 503–512, 2005.

[8] Y. Jing, J. Kuang, J. Du, and B. Hu, "Application of improved simulated annealing optimization algorithms in hardware/software partitioning of the reconfigurable system-on-chip," *Communications in Computer and Information Science*, vol. 405, pp. 532–540, 2014.

[9] J. I. Hidalgo and J. Lanchares, "Functional partitioning for hardware-software codesign using genetic algorithms," in *Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology (EUROMICRO '97)*, pp. 631–638, Budapest, Hungary, September 1997.

[10] P. K. Sahu, K. Manna, N. Shah, and S. Chattopadhyay, "Extending Kernighan-Lin partitioning heuristic for application mapping onto Network-on-Chip," *Journal of Systems Architecture*, vol. 60, no. 7, pp. 562–578, 2014.

[11] J. Wu, P. Wang, S.-K. Lam, and T. Srikanthan, "Efficient heuristic and tabu search for hardware/software partitioning," *The Journal of Supercomputing*, vol. 66, no. 1, pp. 118–134, 2013.

[12] D. Göhringer, M. Hübner, M. Benz, and J. Becker, "A design methodology for application partitioning and architecture development of reconfigurable multiprocessor systems-on-chip," in *Proceedings of the 18th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM '10)*, pp. 259–262, IEEE, Charlotte, NC, USA, May 2010.

[13] L. Li, J. Sun, W. Li, Z. Lv, and F. Guan, "Hardware/software partitioning based on hybrid genetic and tabu search in the dynamically reconfigurable system," *International Journal of Control and Automation*, vol. 8, no. 1, pp. 29–36, 2015.

[14] T. Eimuri and S. Salehi, "Using DPSO and B&B algorithms for hardware/software partitioning in co-design," in *Proceedings of the 2nd International Conference on Computer Research and Development (ICCRD '10)*, pp. 416–420, Kuala Lumpur, Malaysia, May 2010.

[15] Y. Jiang, H. Zhang, X. Jiao et al., "Uncertain model and algorithm for hardware/software partitioning," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '12)*, pp. 243–248, IEEE, Amherst, Mass, USA, August 2012.

[16] L. An, F. Jinfu, L. Xiaolong, and Y. Xiaotian, "Algorithm of hardware/software partitioning based on genetic particle swarm optimization," *Journal of Computer-Aided Design & Computer Graphics*, vol. 22, no. 6, pp. 927–942, 2010.

[17] Y. Hou, R. Wang, Y. Jiang et al., "Embedded system design with reliability-centric optimization," in *Proceedings of the IEEE 39th Annual Computer Software and Applications Conference (COMPSAC '15)*, pp. 33–38, IEEE, Taichung, Taiwan, July 2015.

[18] W. Li, L. Li, J. Sun, Z. Lv, and F. Guan, "Hardware/software partitioning of combination of clustering algorithm and genetic algorithm," *International Journal of Control and Automation*, vol. 7, no. 1, pp. 347–356, 2014.

[19] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proceedings of the Congress on Evolutionary Computation (CEC '99)*, pp. 1945–1950, Washington, DC, USA, July 1999.

[20] S. A. Mingoti and J. O. Lima, "Comparing SOM neural network with Fuzzy c-means, K-means and traditional hierarchical clustering algorithms," *European Journal of Operational Research*, vol. 174, no. 3, pp. 1742–1759, 2006.

[21] A. Sheshasayee and P. Sharmila, "Comparative study of fuzzy C means and K means algorithm for requirements clustering,"

Indian Journal of Science and Technology, vol. 7, no. 6, pp. 853–857, 2014.

[22] J. C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, no. 3, pp. 32–57, 1973.

[23] C. B. James, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Kluwer Academic, 1981.

Rational Metareasoning and Compilation for Optimizing Decisions Under Bounded Resources

Eric J. Horvitz*
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
Stanford, California 94305

Several years ago, our research group initiated a project to investigate the use of decision theory as a framework for reasoning about the design and operation of ideal agents under bounded resources. We have studied metareasoning, reasoning, and compilation within the framework of decision theory. Our model of rationality centers on the use of design-time and tractable run-time decision-theoretic analyses to control the detail and completeness of problem-level decision making. Unlike straightforward decision analyses, we apply the principles of decision theory to enriched models that include not only distinctions and outcomes in the world, but also distinctions and outcomes about cognition. In this paper, we shall review some earlier work on rational metareasoning and describe the benefits of integrating deliberative models of decision-theoretic reasoning and metareasoning with several classes of precomputed or *compiled* actions.

1 Introduction

Much of the work in artificial intelligence can be cast as the development of techniques on which agents with limited representational and inferential abilities can rely for success in the face of complex challenges. Over the last 4 years, investigators on the PROTOS[1] project have been developing techniques for generating and evaluating optimal computational behavior under limitations in reasoning resources. We are interested in decision-theoretic techniques for use in offline design, as well as for explicit application in the deliberative machinery of real-world reasoners.

Our earlier research elucidated the value of decision-theoretic metareasoning in the control of complex object-level problem solving. Here we shall highlight the promise for an integrated approach to rationality centering on the development of agents that make use of a variety

*This work was supported by a NASA Fellowship under Grant NCC-220-51, by the National Science Foundation under Grant IRI-8703710 and by the National Library of Medicine under Grant RO1LM0429. Computing facilities were provided by the SUMEX-AIM Resource under NIH Grant RR-00785.

[1]"Protos" is a partial acronym for *p*roject on computational *r*esources and *t*rade*offs*.

of classes of compiled knowledge in addition to apparatus for reasoning and metareasoning. After reviewing some highlights of our earlier work on partial computation and rational metareasoning, we shall discuss the promise of methods for integrating deliberative models of decision-theoretic reasoning and metareasoning with compiled knowledge. Finally, we shall discuss the role of learning in the context of partial compilation for optimizing an agent's behavior.

2 Rationality Under Resource Constraints

Our research has pursued the development of optimal reasoning strategies and behaviors for computational agents within their environments. That we are studying the optimization of reasoning does not necessarily mean that verifiable optimality is our end-goal. Rather, the endeavor can help us to make explicit the relative performance of alternative reasoning methodologies, and to gain additional insight about how we might go about enhancing the activity of our computational agents. More fundamentally, the pursuit of bounded optimality elicits a number of significant questions and research opportunities regarding ideal and resource-constrained *rationality.*

There are competing views on the nature of rationality. Indeed, rational reasoning and behavior has been a hotly debated topic for centuries. A familiar perspective, held by a great number of researchers in the behavioral and decision sciences, is that rational decisions are those that maximize a numerical measure of preference, termed *utility.* Utility is defined by the axioms of utility theory enumerated by von Neumann and Morgenstern over four decades ago [47]. The axioms of utility and probability comprise decision theory.[2]. From a decision-theoretic perspective, a reasoner that optimizes utility is termed *normative.*

2.1 Bounded Optimality

In 1955, Simon noted that we should consider constraints on cognitive resources in generating and evaluating the behavior of a decision maker in a complex situation. However, he and other early pioneers of symbolic-processing models of intelligence quickly moved away from decision theory. Citing the limited abilities of human decision makers and the forgiving nature of many problems in the world, Simon proposed that most intelligent behavior is oriented toward finding relatively simple solutions that are nonoptimal, yet are sufficient or *satisficing* [44,45]. The primary task was viewed as devising computational procedures that could forward the goals of an intelligent decision maker—and not necessarily remaining consistent with decision theory. This theme has stimulated broad artifical-intelligence research on relatively ill-characterized heuristic procedures in a wide array of domains. Beyond machine-intelligence research, a number of economists and psychologists have also investigated issues related to satisficing behavior under resource constraints [46,38,43]. Thus, bounded-rationality research has come to be associated with the centrality of identifying and recreating heuristic satisficing decision-making behaviors.

[2]A detailed review of past and current efforts to apply decision theory for tackling challenging artificial-intelligence problems is found in [27].

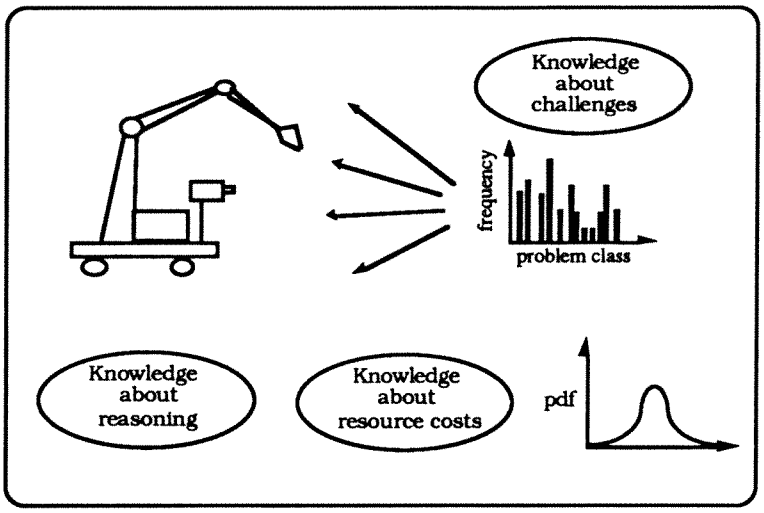


Figure 1: We desire our agents to guide their computation and to act with knowledge about the quality of results expected with increasing quantities of computation, knowledge about reasoning costs, and knowledge about challenges from the environment. Generally, all this knowledge is uncertain, as indicated by the prominence of the probability distribution.

Unfortunately, the nonnormative approaches to problem solving, characterizing the bulk of research under the label of *bounded rationality*, may stray far from the levels of utility that might be achieved through the pursuit of more sophisticated normative analyses. Losses may be especially significant in high-stakes decision making, given complex uncertainties about the world. Such potential losses—and opportunities for great gain—highlight the potential usefulness of decision theory for optimizing the value of behavior under resource constraints.

We use *bounded optimality* to distinguish our work on the optimal design of problem solvers and solution methodologies under bounded resources from traditional nonnormative approaches to reasoning under resource constraints [25]. Bounded-optimal systems perform ideally in some context or set of contexts under expected constraints in reasoning resources, such as in time and memory, given the expected utility structure of a problem. We define such optimality in the context of a decision-theoretic analysis of the costs and benefits of alternative hardware configurations and strategies for determining the best action. As portrayed in Figure 1, we must consider the knowledge about the solution methodologies available, the costs of reasoning resources, such as memory and time, and the expected challenges that will be faced in an environment over some period of time.

In general, we are uncertain about the nature of all of these classes of knowledge. Thus, we must generally reason under uncertainty and use probability distributions (or approximations of such distributions) about the costs and benefits of different reasoning strategies, and of expected challenges from the environment. Other factors include what an agent's uncertainty about future preferences is and how utility assigned to states over time will be weighted and combined in the future.

The determination of true bounded optimality requires proving tight lower-bounds on the solution of problems given the informational and computational constraints at hand. Our pursuit of bounded optimality does not mean we will always be able to prove optimality.

Indeed, bounded optimality may not always be welldefined. However, in the absence of theoretical limits, decision-theoretic meta-analyses can enable us to begin to reason about the relative optimality of agents of different constitutions in partially characterized environments [25,17]. We can also define different *classes* of bounded optimality under broad constraints in the constitution and abilities of reasoners. For example, we may seek and verify *strategic bounded optimality*—optimality given a limited repertoire of available reasoning strategies.

2.2 Epistemic and Autoepistemic Decision Models

The axioms of decision theory define rationality in terms of a *decision model.* Decision models capture knowledge about possible decisions and outcomes, and a state of information about the world in which a reasoner is immersed. Decision theory dictates that a decision-making system or agent should determine ideal actions by performing inference on its decision models. Given a decision model, the goal is to select an action that has the greatest utility.

It is important to note that traditional decision theory has nothing to say about the choice of a rational model. In practice, our agent either must use a predefined model of the world, or dynamically formulate and instantiate a decision model. After a model has been constructed, the agent can perform decision-theoretic inference based on the model.

2.2.1 Models of the Environment

Attempting to apply decision theory in a straightforward manner to detailed representations of the world can produce decisions that would be ideal only in a world of abundant time and memory. Such strategies typically are irrational in the real world of limited resources; the tasks of decision-theoretic model construction and inference are often computationally intensive.

The intuitions of early investigators in artificial intelligence on the inapplicability of implementations of decision theory are supported by recent complexity analyses. Research on the theoretical foundations of the complexity of exact probabilistic inference has demonstrated that automated probabilistic inference is intractable in the worst case [7]. Under constraints on time or on other resources, such as model construction and instantiation, approximations and more poorly characterized heuristic techniques often have a higher expected value than does complete normative reasoning. The potential for suboptimal procedures to dominate "optimal" analyses under resource constraints is easy to demonstrate with a formal analysis [26].

2.2.2 Models of Representation and Inference

Rather than reject the pursuit of a theoretical foundation for ideal belief and action, we have sought to extend coherently the principles of normative rationality to situations of uncertain, varying, and scarce reasoning resources. A central aspect of this research is the development of valuable approximation strategies and the enrichment of the models that are manipulated by decision-theoretic principles. We wish to extend naive decision-theoretic analyses that

make use of models that represent only the challenge at hand with autoepistemic distinctions and relationships.

Our research to date has highlighted the usefulness of models that manipulate distinctions about problem solving in addition to knowledge about the world. We are studying rich multilevel models and associated approximation machinery that are promising in their ability to optimize the behavior of agents under static and varying constraints in reasoning resources. That is, we broaden the attention of decision-theoretic analyses beyond representations of the external world to include in our analyses distinctions about reasoning machinery and procedures in the cognitive world.

3 Utility of Agency

Let us first define some basic terminology necessary for delving deeper into rationality under bounded resources. We use *comprehensive value*, u_c, to refer to the utility attributed to the state of an agent in the world. This value is a function of the problem at hand, of the agent's best default action, and of the stakes of a decision problem. We call the net change expected in the comprehensive value, in return for an allocation of some computational resource to reasoning, the expected value of computation (EVC). The comprehensive utility, at any point in the reasoning process, can be viewed typically as a function of two components of utility: the *object-level* utility, u_o, and the *inference-related* cost function, u_i. The *object-level* utility is the expected utility associated with a computer result or state of the world without regard to the cost of reasoning that may be necessary to generate the state. The object-level utility is a function of a vector of attributes, $\vec{v}$. The *inference-related* component is the sum of the expected disutility associated with, or required by, the process of problem solving. This cost typically is the disutility of delaying an action while waiting for a reasoner to infer a recommendation.

In general, the inference-related cost is a function of a vector of resource attributes, $\vec{r}$, representing the quantity that has been expended of such commodities as time and memory. We have examined several classes of functions describing u_i, including the *urgency*, *deadline*, and *urgent-deadline* situations [26]. *Urgency* refers to the general class of inference-related utility functions that assign cost as some monotonically increasing function of delay. The *deadline* pattern refers to cases where $u_i(\vec{r})$ is 0 or insignificant until a certain level of resource is reached. At this point, computation must halt, and the maximum object-level utility attained before the halt must be reported immediately. Otherwise, the result is worthless. The *urgent-deadline* requires consideration of both the cost and availability of time. These cost functions are common in many real-world applications and are based in such universal interactions as lost opportunity and competition for limited resources.

3.1 Partial Computation

A key notion in rational metareasoning is that of partial results generated by partial computation. The notion of partial computation and its role in rationality has been analyzed by our group [25] and by Dean [8]. This work highlighted the notion that theoretical computer-

science research has been limited in its focus on the analysis of the difficulty of achieving a well-defined final result [3,12]. Although such an assumption can bring useful simplification to analyses of computational complexity, it biases research toward policies that are indifferent to variation in the utility of a result or to the costs and availability of resources. Investigators working on computational complexity implicitly assume only one of two measures of utility to computational behavior: Either a final solution can be computed, which has maximum object-level utility, or a solution is not found in the time available, and the effort is worthless.

Under bounded resources, we do not generally have enough time to perform the computation required to generate an ideal answer. Approximation strategies or solutions of related problems can generate nonoptimal results for some fraction of the computation required for generating an ideal answer: wide variations in the value of a result to an agent, in the availability of resource, and in the cost of reasoning highlight the limitations of a focus on time complexity for termination on final results. We are interested in reasoning about optimizing quality of a solution in the time available. Thus, it is important to enumerate families of *partial results* for different ideal goals.

3.1.1 Partial Results

Partial results $\pi(I)$ result from applying a sequence of computation steps, or a *strategy*, $S_i(I, \vec{r})$, to a problem instance I, with the expenditure of resource $\vec{r}$. That is,

$$S_i(I, \vec{r}) = \pi(I)$$

The partial results are transformations of desired ideal results $\phi(I)$ along one or more dimensions of utility, where

$$0 \leq u_o[\pi(I)] \leq u_o[\phi(I)]$$

More specifically, the object-level utility function maps a real-valued object-level utility to a vector $\vec{v}$ of attributes in an approximation space $\mathcal{A}_o$ for $\phi(I)$ for each partial result. For the purposes of summarization, it can be useful to define a context-independent distance metric $D : \mathcal{A}_o \times \mathcal{A}_o \rightarrow \mathcal{R}$ between points in this space. We relate the difference in utility of $\phi(I)$ and $\pi(I)$ to a function of the context and this distance. An example of a widely-used, context-independent distance among results is the numerical approximation, where D is a simple unidimensional measure of precision (e.g., the result of a Taylor series carried to a particular term). In this case, $\phi(I)$ and $\pi(I)$ are separated by a distance in the space of reals.

Beyond the familiar numerical approximation, we can consider cases where D represents the divergence of $\pi(I)$ from $\phi(I)$ along higher-dimensional and more abstract properties of a computational result. Identifying poorly understood classes of partial results can highlight directions for research on approximation strategies and reasoning under resource constraints. Dimensions in $\mathcal{A}_o$ often are based on the use made of the result and are rooted in human preferences. Simulation methods partially characterize a probability of interest through probabilistically visiting portions of a problem, yielding a sequence of probability distributions over a set of states with additional computation. Randomized approximation algorithms generate results that take the form of inequalities on error bounds on an ideal result.

To highlight general principles of rational metareasoning and to demonstrate the richness possible in partial computation, we examined the problem of metareasoning about the task of sorting. In this work, multiattribute object-level utility functions were used to assign utility to incompletely sorted file of records, based on several different dimensions or attributes of *sortedness* [21,26]. Attributes explored include *disorder*, a measure of the average distance between current locations and expected final locations for records in a file, and *completeness*, the contiguous length of a file containing records sorted into their final positions. The multiattribute utility structure of alternative algorithms define problem-solving trajectories through multiattribute spaces. We examined how an agent could use knowledge about such trajectories to choose the best strategy to apply, given the cost of reasoning.

3.2 Graceful Degradation

Several properties of reasoning strategies and representations are desirable for reasoning under bounded resources [25]. We desire our strategies to provide us with partial results that have object-level value that increases monotonically with allocated resource. We additionally desire our strategies to be relatively insensitive to small reductions in resource fraction. Thus, we desire continuity in the refinement of partial results. Such *incremental-refinement* policies yield immediate object-level returns on small quantities of invested computation, reducing the risk of dramatic losses in situations of uncertain resource availability. This property is especially important for reasoning under uncertain resource constraints [25,8]. Finally, we desire the partial results to converge to an ideal answer with some finite allocation of resource. A class of reasoning strategies called spanning strategies are particularly useful for reasoning under varying resource constraints. *Spanning* strategies are approximation procedures that exhibit monotonicity, continuity, and convergence on an ideal object-level solution with the allocation of a finite quantity of resource.

In our work on metareasoning about fundamental computational tasks, we highlighted the utility of continuity by analyzing the example of sorting under resource constraints. In this study, we compared faster all-or-nothing $O(N \log N)$ sorting strategies [34] with several slower polynomial strategies. Although the latter strategies take longer to sort completely our test files, they have the ability to refine incrementally one or more object-level attributes of a partial sort. Given a variety of file sizes, prototypical utility models, and resource contexts, the slower polynomial sorting approaches were more valuable than the faster approaches. For example, mergesort, heapsort, and quicksort do not generate valuable intermediate results. If a deadline occurs at some time before completion, the result is useless or is associated with a cost. Under conditions of uncertain or poor resource availability, or of high cost of reasoning, we can generate a more valuable result by applying Shellsort, which is more conservative with a $O(N^{1.5})$ time complexity, yet is more graceful because it refines partial sorts continuously with time [26].

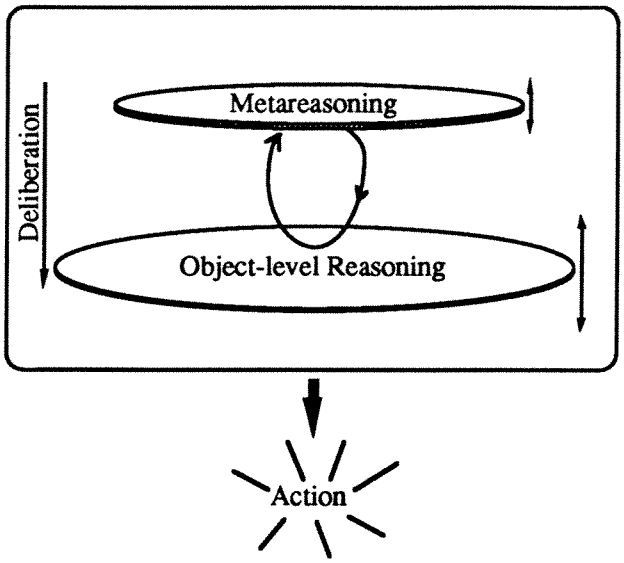


Figure 2: Optimization of the value of computation under uncertain resources and challenges, can necessitate the application of a portion of the available resource to metareasoning about optimal representation and reasoning.

4 Rational Metareasoning

Decision theory provides the foundations for a principled approach to metalevel decision making under the general condition of uncertainty. We shall first examine the general deliberative approach to decision-theoretic reasoning under resource constraints. That is, we shall look at the general notion of optimizing expected utility when we have available alternative or continuously tunable approximations strategies. Then we shall explore the integration of compiled knowledge for reasoning and metareasoning.

We are typically uncertain about the states that will be reached with future computation and about the utility of these states. We approach the problem of rationality under resource constraints by constructing one or more metareasoning problems to represent and control aspects of object-level representation and deliberation. Our goal is to decide which available strategy has the greatest value, given alternative available reasoning methods and possible object-level actions. This approach focuses our attention on the construction of rich multilevel models and on inference within these models.

4.1 Expected Value of Computation

Our goal is to optimize the utility of an agent's decision. To do this, we may have to apply a portion of the available reasoning resource to deliberate about *how* best to optimize utility. A chief component of metareasoning about decision making is determining the expected value of computation of alternative available strategies. We first review the calculation of the EVC.

The initial utility of an agent facing a challenge is $u_o(I)$, where I captures the current

problem instance or *state* of the agent in the world. We are interested in the value of applying strategy S_i to the current instance to generate a better state. There is generally uncertainty in the object-level result that will folllow from the expenditure of computational resources on a strategy. Thus, we must consider probability distributions over changes in relevant attributes. These distributions capture our uncertain knowledge about how allocating costly resource will change $\vec{V}(I)$ to $\vec{V}[\pi(I)]$. For brevity, we shall consider a single distribution over partial results and a unidimensional measure of resource, r. The extension to multiple attributes is straightforward. We must apply knowledge of the form

$$S_i(I,r) = p_{S_i}[\pi^j(I) \mid r]$$

where p_{S_i} is a probability distribution over different possible partial results $\pi^j(I)$, conditioned on the allocation of resource r using strategy S_i.

The EVC is the change in comprehensive utility, u_c, associated with the generation of a result. If we assume that inference-related and object-level utilities can be decomposed, and are related through addition, the EVC is just the difference between the increase in object-level utility and the cost of the additional computation. Let us first consider the EVC without considering the cost of metareasoning itself. The EVC of applying strategy S_i with a quantity of resource r, is

$$\begin{aligned} EVC(S_i,I,r) &= u_c(S_i,I,r) - u_o(I) \\ &= \int_j u_o[\pi^j(I)] \times p_{S_i}[\pi^j(I) \mid r] - u_o(I) - u_i(r) \end{aligned} \quad (1)$$

In complex problems, we may be uncertain about the nature of the functions u_o and u_i used to map a object-level utility to attributes of partial results and disutility to resource expenditures. In such cases, we can extend this formula to sum over different weighted combination functions in addition to considering the uncertainty over different attributes of partial results.

So far, we have included the cost of reasoning about the solution of problem instance I, but we have not included the cost of metareasoning in our formula for computing the EVC. In practice, our goal is to minimize the cost of metalevel computation by elucidating tractable closed-form solutions to EVC estimation. We have found that it is possible to apply knowledge about properties of reasoning strategies and cost functions to develop a tractable economics of computation [28]. For example, consider the curves in Figure 4, which characterize the economics of partial computation. If we know the functions that describe different available spanning strategies, and know that the second derivative of the cost of computation is everywhere nonnegative, we know that EVC is equal to 0 and the comprehensive value of computation is maximized when the first derivative of the object-level utility function is equal to the first derivative of the cost function [23].

As tractable as such EVC analyses may be, however, metareasoning requires an additional allocation of resource; a metareasoner must diminish the resource available for object-level computation by the expected cost of metareasoning. To make explicit the inclusion of metareasoning costs, we denote by EVCM the expected value of computation including the costs of metareasoning. Assume that $r^{\mathcal{M}}$ is a fixed cost of metareasoning. The EVCM is

$$EVCM(S_i,I,\mathcal{M},r) = \int_j u_o[\pi^j(I)] \times p_{S_i}[\pi^j(I) \mid (r - r^{\mathcal{M}})] - u_o(I) - u_i(r) \quad (2)$$

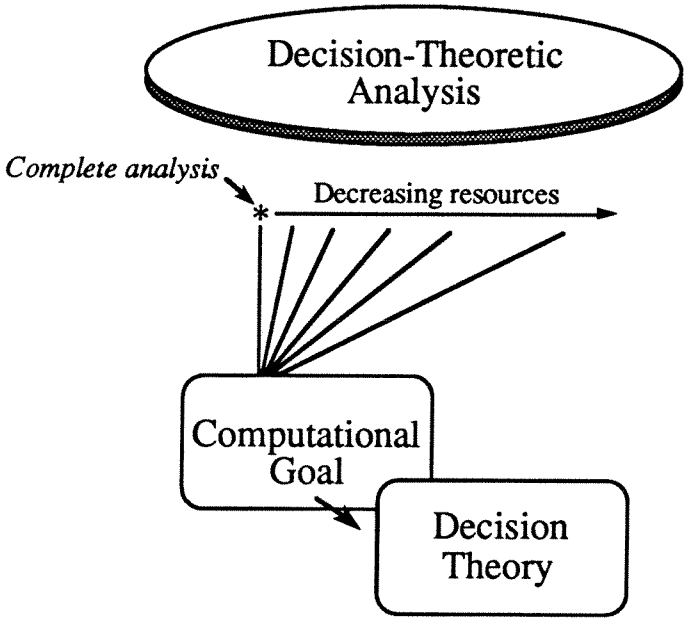


Figure 3: Rational metareasoning about partial computation can be applied to any computational goal, including such fundamental operations as sorting and searching, or to decision-theoretic inference itself. The metareasoning problem focuses on the value of alternative approximations of the object-level decision problem, as the solution ranges from the complete analysis (asterisk) to increasingly poorer approximations (direction of arrow). The rational control of decision-theoretic inference serves as a model of rationality under bounded resources.

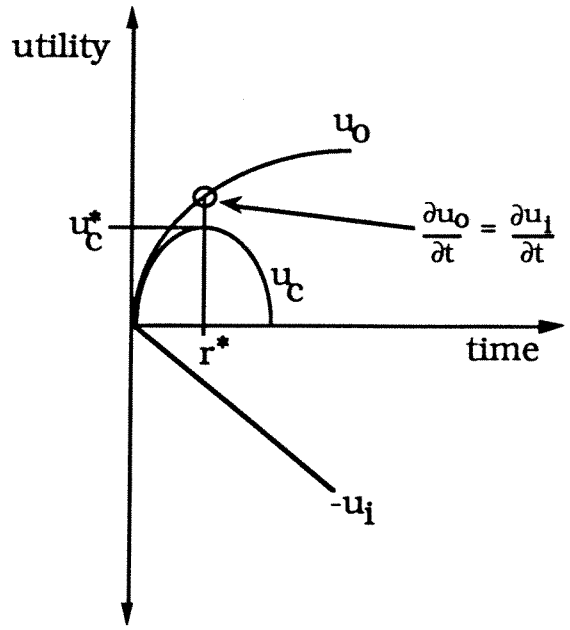


Figure 4: The expected refinement of a result by a spanning strategy, the costs (in this case, a linear cost function), and the comprehensive value of a result as additional resource is allocated to computation. The ideal comprehensive utility u_c^* is generated with the expenditure of an ideal quantity of resource r^*, where the derivatives of the object-level and inference-related utilities are equal.

where $r - r_m \geq 0$, and $\mathcal{M}$ is the metareasoning protocol. This EVCM formula is reflective in that it includes the costs associated with its own calculation. It is clear that the value of our metareasoning apparatus increases as the expected difference in $u_i(r)$ and $u_i(r^{\mathcal{M}})$ grows.

We could attempt to optimize the utility of an agent globally, considering alternative strategy sequences over a large range of resources from our initial state. Global analysis is feasible in some situations. Unfortunately, global analyses can introduce intolerable complexity to metalevel deliberation. Several simplifications can be used as approximate EVC analyses. As an exmaple, in many applications we have only one available reasoning strategy to address a particular challenge. In such cases, we need to consider only the optimality of alternative halting times. We can also reduce the costs of deliberation by performing greedy optimization over a prespecified small quantity of resource R, which increases the previously expended resource r to a new total expenditure r'. In such myopic analyses we continue to deliberate with additional packets of resource until the expected costs of the next allocated R will outweigh the benefits. Thus, a formulation of EVCM, emphasizing the marginal utility of expended additional resource R, is

$$EVCM(S_i, \pi^j(I), \mathcal{M}, R) = \int_{j'} u_o[\pi^{j'}(I)] \times p_{S_i}[\pi^{j'}(I) \mid (R - r^{\mathcal{M}})] - u_o[\pi^j(I)] - u_i(r) \quad (3)$$

where $R = r' - r$, $\pi^j(I)$ is our current result, and $\pi^{j'}(I)$ is the result expected after the next computation. That is, we continue to sum over the expected future utility, weighting each possible state by the probability of achieving that outcome, and to subtract the object-level value of our current state until the expected marginal gain is non-positive. At this

point, we halt, and then act in the world. The total cost of myopic metareasoning includes the costs of metareasoning for each productive computation, in addition to a nonrefundable metareasoning penalty $u_i(r^{\mathcal{M}})$, associated with the last, nonproductive meta-analysis.

Rational metareasoning can be applied to the control of any computational or cognitive task. This point is highlighted by our exploration of the control of computation on tasks ranging from sorting a file of records to decision-theoretic inference. We believe that the optimal configuation of cognitive procedures for generating optimal behavior under bounded resources will frequently dictate some allocation of resource to metareasoning. A current research focus of work in our research group is addressing the value of metareasoning and control, given the architecture of our agents, and the costs of memory and inference. More speculatively, it is feasible that rational metareasoning may well play an important role in the cognitive systems of living creatures developed through long-term optimization under the pressures of competition.

Our approach to rational decision making under bounded resources is to construct a metalevel decision problem that represents distinctions about cognition, and to use this model to reason about and to control the nature and duration of alternative approximations to a complete decision-theoretic analysis. Our metareasoner depends on the existence of object-level approximation strategies that allow us to trade off the quality of an ideal analysis (asterisk in the figure) for more tractable, yet less precise, results. Elucidating and refining techniques for metareasoning and control are only useful if we have flexible problem-solving strategies to control. Thus, we stress that innovation at the object-level will typically be most crucial in constructing reasoners for performing under bounded resources.

4.2 Decision-Analytic Metaknowledge

Metareasoners make use of attributes of problem instances and of reasoning itself that can serve as evidence about the value of future computation. Such handles on the value of alternative reasoning pathways serve to characterize partially the nature of reasoning at lower metalevels and at the object level. As an example, the size and nature of a problem instance can serve as evidence in allowing a metareasoner to estimate the rate at which the quality of a result will increase over time. Another class of attributes that can serve as evidence of future computation is the nature of recent problem-solving behavior. There is rich research opportunity for the design of systems with the ability to inquire continually about and amass metaknowledge for calculating the value of future computation.

Decision-analytic metaknowledge includes knowledge about (1) model construction, (2) inference, (3) metareasoning, and (4) interactions among model construction, inference, and metareasoning. Model-construction metaknowledge captures attributes useful in reasoning about the value of continuing to employ strategies for generating and refining distinctions and relationships in a decision model. Inference metaknowledge includes distinctions useful in estimating the value of future inference. For example, we have been experimenting with inference metaknowledge that involves distinctions about the size and topology of a belief network. Such factors can serve as evidence in allowing a metareasoner to estimate the rate of convergence of upper and lower bounds on probabilities needed for a base-level decision problem. Metareasoning metaknowledge is information about distinctions used to

characterize the expected value of increasing the fraction of time dedicated to metalevel deliberation, or moving to rely on results of a higher metalevel analysis. Finally, interaction metaknowledge captures knowledge about the interaction among model-building, inference, and metalevel deliberation. Such knowledge includes the relationship between models of higher quality and the growth of complexity of the inference task.

4.3 Decisions Under Resource Constraints

So far, we have described general principles of metareasoning without talking about a specific application. Let us now turn to work on the application of metareasoning techniques for controlling decision analysis. The process of decision analysis can be decomposed into the construction of a decision model followed by performance of decision-theoretic inference with the model. We thus must consider resource issues with both phases.

4.3.1 Partial Computation in Decision Making

In decision-theoretic inference, we are interested in partial computation for inference and model construction. Some partial-computation strategies for inference generate results of the form of bounds or distributions over belief by simulation or through the solution of specific portions of a reasoning problem [30]. We can also generate partial computation approaches to inference by incrementally increasing the abstraction of propositions or by decreasing the completeness of dependencies in a decision model [29]. Such completeness and abstraction moduation techniques have been successfully empolyed to control decision-theoretic inference within the Pathfinder reasoning system for pathology diagnosis [19]. Beyond the relatively straightforward partial results for inference and model construction, we can consider other classes of partial results for decision analysis under resource constraints. For example, it may be useful to develop a metric that represents a distance in a conceptual space describing properties of inference; in this regard, a set of partial results might be defined by the probability that a recommendation is inconsistent with one or more axioms of decision theory.

We dwell, in this paper, on inference from models that have been previously constructed. We note, however, that the principles of metareasoning presented apply also to processes for constructing and refining decision models with continuing computation. Decision models generated before the quiescence of the model-refinement process can be considered to be partial models. There are few examples of principled approaches to the construction of decision models. Model construction has been the domain of humans; the machine-intelligence community has proposed few normative approaches to the automation of this task. Exceptions include work on the automated selection of distinctions and relationships for use in a decision model based on the use of a decision-theoretic threshold analysis [18] and a qualitative analysis of important tradeoffs [50].

In metareasoning about belief and decision, we are typically interested in a probability ϕ of a state of the world given some evidence. The value of ϕ could be calculated precisely if an agent had sufficient computational resources. Under the general condition of insufficient time for a complete calulation, an agent can manipulate partial results, which are distributions

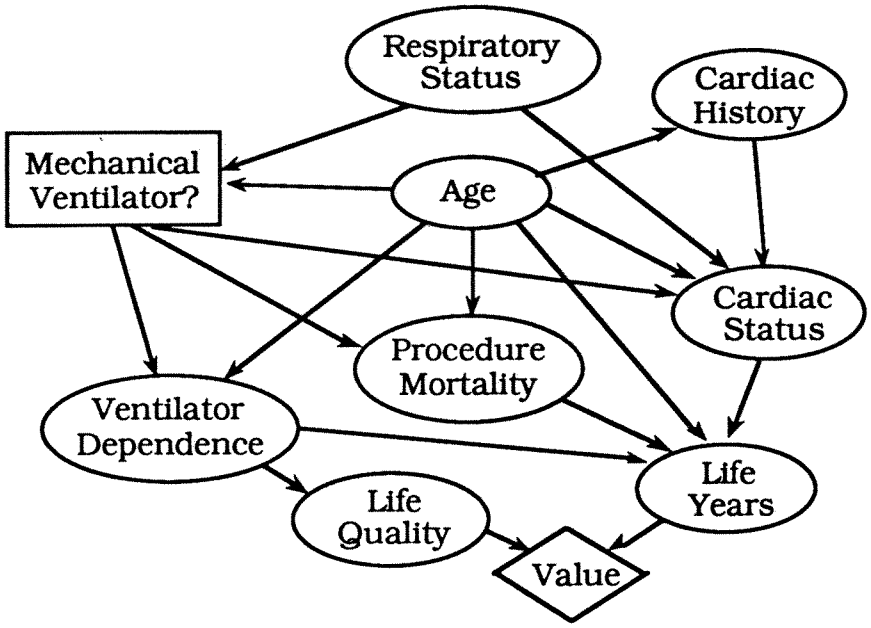


Figure 5: An influence diagram representing uncertain relationships among important distinctions (circular nodes) in intensive-care-unit medicine. Arcs between nodes indicate a dependency between the value of a node and belief assigned to the values of predecessor nodes. The possible actions are represented by a decision node (square). The best decision depends on several beliefs. The diamond represents the utility associated with different outcomes.

over ϕ, $p(\phi)$. Thus, our metareasoning problem is typically of the form

$$EVCM(S_i, p^j(\phi), \mathcal{M}, R) = \int_{j'} u_o[p^{j'}(\phi)] \times p_{S_i}[p^{j'}(\phi) \mid (R - r^{\mathcal{M}})] - u_o[p^j(\phi)] - u_i(r) \quad (4)$$

The EVCM for object-level decision-theoretic computation captures the value of continuing to refine probability distributions about events or predicates that are relevant for making a decision.

4.3.2 The Influence-Diagram Representation

Our decision-theoretic analyses have made use of *influence diagrams* [31] for representing and solving automated reasoning problems. The influence diagram is an acyclic directed graph containing nodes representing propositions and arcs representing interactions between the nodes. Nodes represent a set of mutually exclusive and exhaustive states; arcs capture probabilistic relationships between the nodes. Influence diagrams without preference or decision information are termed *belief networks*. A belief network defines a model for doing probabilistic inference in response to changes in information.

Algorithms for doing inference are expected to have a worst-case time complexity that is exponential in the size of the problem (e.g., the number of hypotheses and pieces of evidence) [7]. Several methods for inference in belief networks avoid intractability by exploiting independence relations to avoid the calculation of the joint-probability distribution. A variety

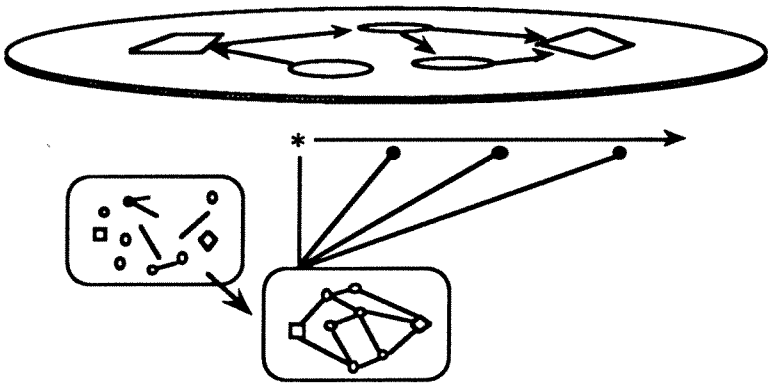


Figure 6: Under resource constraints, it can be valuable to enrich traditional decision models with metalevel decision problems. The metareasoning decision problem considers the value of expected partial results, as the quantity of resources ranges from that needed for a complete analysis to increasingly smaller allocations.

of exact methods has been developed, each designed to operate on particular topologies of belief networks [27].

The complexity of precise inference and the availability of alternative belief-network inference algorithms highlight the need for robust approximation strategies and intelligent control techniques. We have studied models of reasoning based on the application of influence diagrams, representing the metareasoning problem, to control more complicated object-level inference. The schematic portrayed in Figure 6 represents the general framework for rational action under bounded resources. We build a metalevel problem to reason about inference of varying degrees of approximation.

4.3.3 Empirical Study

We studies empirically the application of rational metareasoning to computation and action within alternative utility contexts in time-pressured medical decision making [28]. In some of this work, we endeavored to generate closed-form solutions to Equation 4 by making assumptions about the distributions over a probability of interest. After generating a tractable metareasoning model, we can apply knowledge about the behavior of alternative reasoning strategies. In particular, we have explored the behavior of recently developed graceful approximation methods for probabilistic inference. These strategies include a flexible variant of Pearl's method of conditioning [40], called *bounded conditioning* [30]. This approach satisfies the desirable properties of continuity, monotonicity, and convergence; the algorithm incrementally refines bounds on a probability of interest, continuing to tighten the upper and lower bounds on a probability of interest, with continuing computation, until reaching a point probability.

PROTOS work on the deliberative approach to ideal inference under constraints is captured Figure 7. A metareasoning decision problem is used to control the nature and extent of inference in a complex belief network. The structure of a sample belief network that we have analyzed is represented in the middle of Figure 7. This network represents uncertain

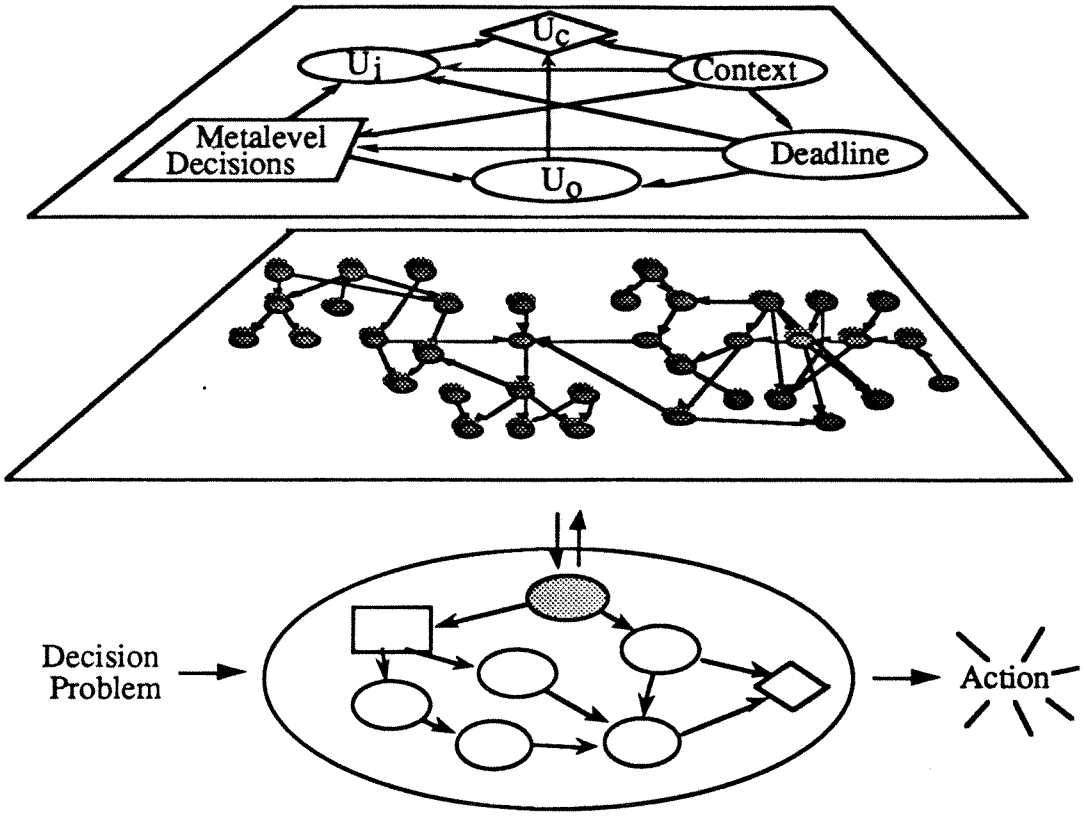


Figure 7: Research on deliberative metareasoning has explored the use of a metalevel decision model to control probabilistic inference in a complex object-level belief network. Object-level decision problems are passed to a metareasoner for evaluation.

relationships between observations and patient pathophysiology in intensive-care medicine. Object-level decision problems, requiring belief-network inference, are passed to the metareasoner, which determines the optimal dwell time.

We have pursued tractable solutions to the EVC by examining parameterized families of distributions. This work is described in more detail in [28]. For example, we have explored the use of rational metareasoning to control the application of probabilistic-bounding methods. In the work on intensive-care medicine, our reasoning system makes use of inference metaknowledge in the control of reflection and action. The inference metaknowledge is in the form a partial characterization of how the probability distribution over propositions of interest will change as reflection continues. This type of knowledge is central in reflection about the value of initiating or continuing decision-theoretic inference, as opposed to that of acting with the current best decision. We found that approximation strategies can typically deliver a large portion of the value of complete computation with a fraction of the resource [28].

4.4 History of Rational Metareasoning

Interest on the decision-theoretic control of computation has blossomed recently. To our knowledge, the general framework of rational metareasoning about partial computation was

first explored and presented by our research group. In 1986, we introduced the notions of partial computation, enumerated desiderata of computation under bounded resources, and described the applicability of decision-theoretic control for optimizing the value of computation under resource constraints [21,22,25]. We proposed the decision-theoretic control of decision theory as a model of rational belief and action. Although PROTOS research has concentrated on metareasoning about decision-theoretic inference and knowledge representation, early investigations demonstrated that multiattribute decision-theoretic control of reasoning had promise for guiding the solution of a variety of tasks, including such fundamental problems as sorting a file of records or searching a large tree of possibilities [24]. To highlight the breadth of applicability of the ideas, we examined the use of the multiattribute decision theory in the control of sorting, within the PROTOS/ALGO system [26]. In 1988, Dean and Boddy independently introduced notions of partial computation [8]. Dean's group later moved to consider a decision-theoretic perspective on computation similar to ours, and has been exploring problems in temporal reasoning [33] and planning under resource constraints [4]. In 1988, Hansson and Mayer described independent work on evidential reasoning and value tradeoffs in search problems [15]. More recently, this team has been more recently examining the use of coherent evidential reasoning about the value of nodes to control search [16]. The same year, Fehling and Breese explored the application of desion-theory to the control of a robot planning problem [11], and Russell and Wefald examined the application of metareasoning to game-playing search [42]. This group has more recently explored single-agent search [49]. Agogino and colleagues have sought to apply principles of decision theory to select the best model to use in real-time reasoning in the control of milling machinery [1]. In related recent work, Doyle described a representation and analysis of a distributed approach to rationality [9], and Etzioni and Mitchell began to analyze the decision-theoretic control of learning [10].

Various aspects of the recent work on rational metareasoning can be viewed as a rediscovery of earlier speculation. The decision-theoretic approach to the control of reasoning was discussed by the statistician I.J. Good in the context of the direction of game-playing search over a decade ago [14]. Good had earlier discussed the explicit integration of the costs of inference within the framework of normative rationality, defining Type I rationality as inference that is consistent with the axioms of decision theory, regardless of the cost of inference, and Type II rationality as behavior that takes into consideration the costs of reasoning [13]. Related work in decision science has focused on the likely benefit of expending additional effort in performing decision analyses [39,48].

5 Compilation and Reflex

So far, we have discussed only deliberative approaches to reasoning and metareasoning. We have been working to move beyond reasoners that must deliberate explicitly about action. We wish to reduce complex deliberation in computer-based reasoners by developing decision-making techniques that rely to some extent on precomputed or *compiled* responses. Such knowledge can be generated at design time or learned by agents over their lifetimes.

There has been a trend in AI research to move away from deliberation toward compiled models of reasoning. For example, recent research on *reactive planning* has centered on the

replacement of unwieldy solution mechanisms and detailed representations of knowledge with compiled *situation–action* rules [41,2,6,32,17]. Such rules enable agents to respond, in reflex fashion, to perceptual inputs. Investigators have sensed that, for many contexts, explicit representations and deliberation will not be necessary for satisficing performance.

5.1 Classes of Compiled Knowledge

We have been investigating several classes of compiled knowledge. These classes of compiled knowledge include (1) *situation–action rules*, (2) *platform rules*, and (3) *resource rules*. Each can play a distinct role at any level of analysis to increase the comprehensive value of decision making under resource constraints.

5.1.1 Situation–Action Rules

The simplest form of compiled knowledge is the *situation–action rule*. An observed situation is linked in reflex fashion to a final action. This form of rule is akin to the reflex response of a person to accidentally touching a hot stove. The state of VERY HOT is linked, without deliberation (besides the relatively low-overhead of retrieval), to MOVE AWAY.

5.1.2 Platform Rules

Platform rules are a generalization of situation–action rules used in conjunction with deliberative mechanisms *to make deliberation more efficient*. Such knowledge gives an agent's deliberative machinery a boost in solving a problem. Platform knowledge reduces the computational burden of reasoning. Compiled platform knowledge can be viewed as a mapping between a problem instance and a precompiled partial result, $\pi^{\mathcal{C}}(I)$. If we require resource $r^{\mathcal{C}}(I)$ to access the compiled knowledge, the availability of a compiled result allows us to substitute $u_o[\pi^{\mathcal{C}}(I)] - u_i[r^{\mathcal{C}}(I)]$ for $u_o(I)$ in our EVC formulae. The overall value of quantities of platform knowledge depends, in part, on the cost of memory and on the frequency of alternative challenges in an agent's environment. The expected gain associated with the storage and use of a particular piece of platform knowledge is the difference in optimal EVC achieved, in response to a challenge, in systems with and without the compiled knowledge. We must weight such gains by the frequency of a challenge over the agent's lifetime, and subtract the cost of purchasing and maintaining memory needed to encode the knowledge.

Platform rules include cached partial results for general classes of problem instances that can be refined with additional computation. Deliberative mechanisms can be designed with explicit reliance on the existence of platform knowledge. As an example, we can cache partial results for belief-network computation to make probabilistic inference more efficient. As another example, we can use platform rules about decision models to generate basic decision-model templates in reaction to salient features of a problem instance. Complementary deliberation can be used to custom-tailor the model to the specific situation at hand.

We can view precomputed, reflex knowledge of different degrees of completeness and levels of detail as lying at points along a compilation–deliberation continuum. The null case of zero

platform knowledge defines the completely deliberative end of the spectrum. At the opposite end of the spectrum are situation–action rules. Situation–action rules can be viewed as a special case of platform knowledge where we do not perform additional deliberation.

5.1.3 Resource Rules

Situation–action rules and platform rules of different levels of compilation (capturing varying fractions of a problem-solving task) help an agent to make decisions in a more timely manner. That is, these classes of compiled knowledge provide a means for making decision making more efficient. Another class of compiled rules comprises reflex behaviors that serve to *generate additional resource*, by reducing the cost of delay, or pushing back a deadline. Resource rules are common in real-world domains that require difficult decisions and activity under time pressure. As an example of a resource rule in medicine, if a patient's blood pressure is falling dramatically, a common reaction is to add more fluids to the patient's circulatory system to raise the blood pressure. This action is typically only a temporary remedy. It serves to generate additional time for gathering information, for reasoning, and for acting to address the fundamental cause of the problem. In the case of our patient with falling blood pressure, the problem may involve a hidden hemorrhage or cardiac difficulties. Resource rules may be knowledge intensive and are typically custom-tailored to the domain at hand.

5.2 A Decision-Theoretic Perspective on Compilation

In most of the work on reactive planning, investigators have overlooked the principles of decision theory in the design and evaluation of compiled models. Thus, from the perspective of decision analysis, the compilation techniques to date typically are suboptimal for the general case of decisions under uncertainty. There have been several discussions of compilation, from the decision-theoretic perspective. Investigators have explored the derivation and consistency of compiled situation–action rules from underlying normative decision models [5], the relationship between deliberation and compiled knowledge of varying degrees of completeness [25], and the relationship of heuristically determined situation–action rules and decision-theoretic models [36]. Two recent analyses have focused on use of the principles of probability and utility theory for compiling knowledge.

A study by Heckerman and associates explores the application of decision theory to the problem of designing optimal sets of compiled situation–action rules. The analysis compares the value of completely deliberative reasoners to that of using only compiled models as a function of these factors. The analysis shows that the optimal compilation depends on (1) the nature and distribution of evidential relationships in a domain, (2) the utilities associated with alternative actions, (3) the costs of run-time delay, and (4) the costs of memory. The study describes optimal design-time rules, in the context of a binary decision problem, for choosing evidence sequences to optimize the value of compiled knowledge given the cost of memory, as well as the means to calculate the best decision, for selected subsets of observed features. A portion of the analysis examines the storage and access of arbitrary sequences of evidence within an efficient tree representation called a *trie*. A sample situation–action trie

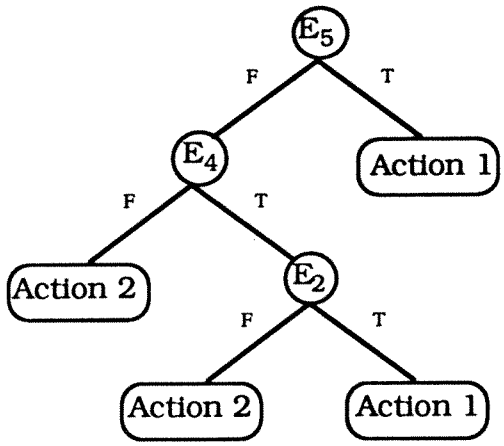


Figure 8: A sample trie containing compiled actions for a binary decision problem. In response to the observation of evidence, an action is immediately indicated.

is depicted in Figure 8. Here the compiled situation–action knowledge addresses the binary decision of whether an agent should take ACTION 1 or ACTION 2. From the example, if only evidence E_5 is observed, ACTION 1 has greatest expected value. If only E_4 is observed, then it is best to embark on ACTION 2. If, however, states E_4 and E_2 are observed, it is best to take ACTION 1.

Herskovits and Cooper have investigated the generation of platform knowledge for probabilistic inference. They compiled a set of conditional probabilities computed from a belief network in an offline manner [20]. The probabilities were selected through a utility-based simulation and also cached in a trie. Such precomputed probabilities can be used to increase the average response time of belief networks to probabilistic queries.

6 Integration of Compiled and Deliberative Reasoning

We are working to optimize the value of behavior under resource constraints by integrating compiled rules with deliberative reasoning. We can build systems that do complete analyses or different types of approximate analyses at different points along the deliberation–compilation continuum. We are interested in developing tools for designing and evaluating ideal configurations of metareasoning, reasoning, and reflex decision-making apparatus of agents, given a set of fundamental deliberative abilities, the expected problems encountered in an environment, and quantities or costs of memory and time. The ideal configuration depends on fundamental constraints on the constitution of the agent and on the nature of the challenges faced from the environment.

As depicted in Figure 9, in the context of our earlier work on deliberative models, we wish to add compiled knowledge to bolster the timeliness and accuracy of decisions made by our agents. In the general case, compiled models are incomplete. As pointed out in [17], if we store compiled knowledge solely as sets of arbitrary situation–action pairs, more than one cached situation may match the observed evidence. As an example, we may cache, as the

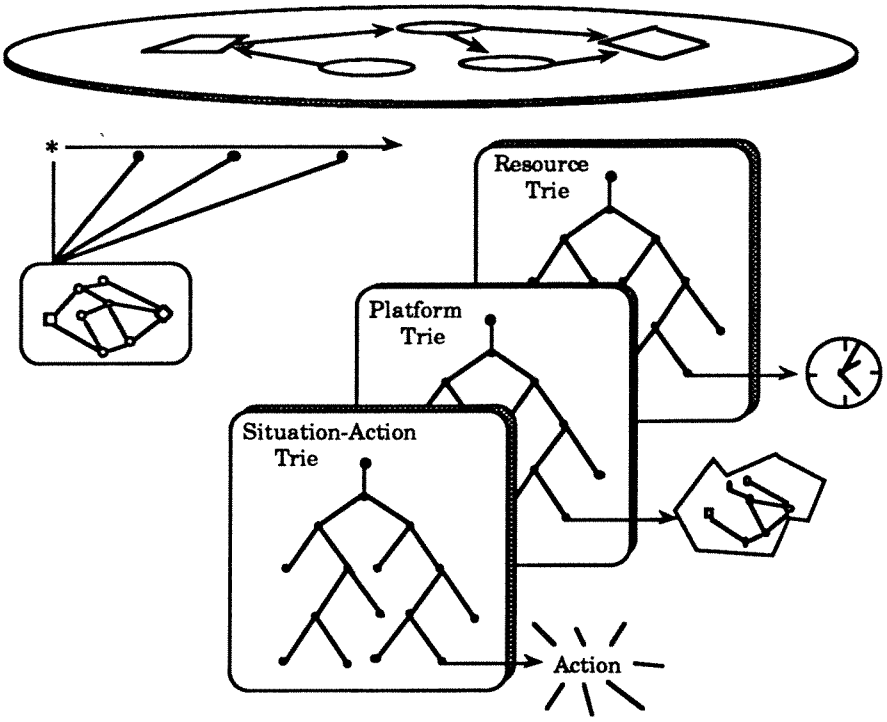


Figure 9: Several different classes of compiled knowledge can be used in an integrated fashion to make decision making more efficient. These classes include situation–action, platform, and resource knowledge. A metareasoner can make use of incomplete characterizations of the value of committing to a compiled situation–action rule versus deliberating about a challenge under expected resource constraints.

left-hand sides of a trie of situation–action rules, situations A, B and A, C and we may then observe the state A, B, C.

Incomplete compilations will typically ignore potentially important details of problems. Thus, the quality of compiled actions typically will be lower than the ultimate quality derived from computing. Under typical resource constraints, however, the compiled models might easily dominate the computed results.

In the case of the use of situation–action rules, the integrative task requires the encoding of knowledge about the relative expected value of deliberative and compiled strategies. This knowledge can be incomplete, and therefore can be represented as approximate or uncertain information. The generation and effective use of platform knowledge requires investigation of the relationships between compiled partial results and deliberation, and encoding of how the use of compiled platform knowledge is expected to enhance deliberation. Finally, we need to enrich our metareasoners such that they can quickly decide between the use of compiled situation–action rules and deliberation, determine the expected value of combining platform knowledge with deliberation under different situations, and consider the benefits and costs of firing resource rules.

6.1 Compilation of Metareasoning

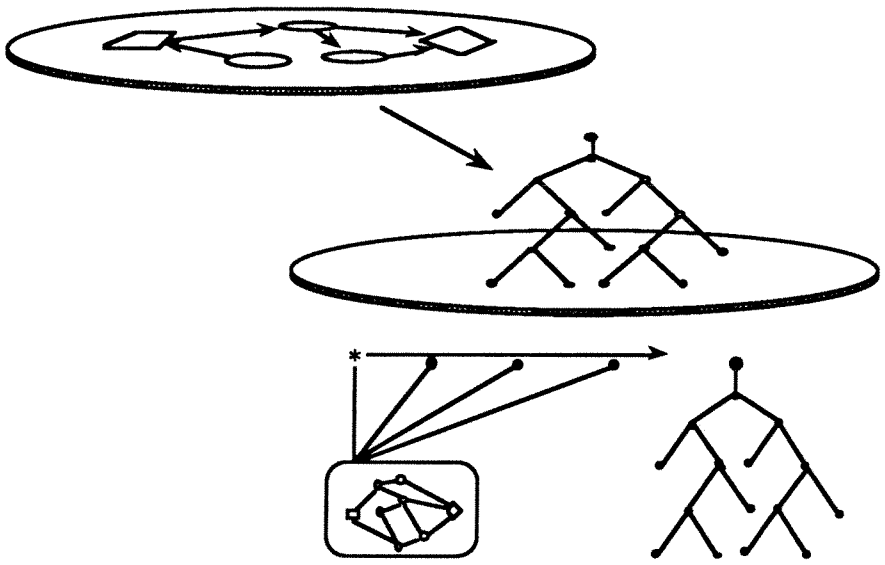


Figure 10: We can attempt to compile a portion or all of the metareasoning problem, depending on several factors, including the complexity of the model. Compilation of the metareasoning problem itself can make metalevel deliberation more efficient, allowing for more complex metareasoning models. Also, metareasoning compilation provides for quick access to object-level situation–action rules.

Metareasoning decision problems are choice candidates for compilation. We seek generally to formulate relatively tractable metareasoners for the control of difficult object-level reasoning so that we can spend a large fraction of available resource on object-level inference. Previous research in our group has explored deliberative models that have relatively tractable closed-form solutions to the value of computation problem. The discovery and application of a tractable closed-form solution to a metareasoning problem can be viewed as a form of metareasoning compilation.

In general, complex metareasoning problems may not allow for tractable closed-form solutions. These problems can pose unwieldy resource burdens on the solution of the problem. Attempting to compile some or all of the metareasoning problem into different forms of situation–*meta*-action rules can allow us to apply more complex metareasoning models.

The simplest compiled metareasoning strategies are default metareasoning policies. Such policies are commonplace in almost all simple object-level reasoners. That is, relatively inflexible object-level strategies, designed in the absence of metalevel control, implicitly impose a default metareasoning policy. Simple compiled metareasoning strategies can also be designed explicitly. These include such straightforward control strategies as the following: *(1) if a situation–action rule is available, take an action determined by the rule's consequent; (2) if it is not available, deliberate for a time dictated by the appropriate situation–meta-action rule.* More general compiled metareasoning models can make use of simple cached estimates of the expected value of deliberation versus the compiled response to a situation. Note that compiled metalevel control gives us instant access to object-level situation-action rules via situation–meta-action rules.

7 Multilevel Models

It may be valuable to custom-tailor metareasoning policies dynamically by adding one or more meta-metareasoners to control the nature and extent of metareasoning itself. The control of metareasoning can have great payoff for computation associated with difficult metareasoning analyses. For example, in the context of a difficult metareasoning problem, and the availability of several approximations to the meta-analysis, we may find great value in the use of a closed-form meta–meta-analysis that can limit metareasoning optimally.

7.1 Utility of Multilevel Metareasoning

There are costs and benefits of adding another level of analysis. The usefulness of a new metalevel for enhancing the object-level behavior of an agent is a function of the value of increased tractability and flexibility achieved through the control of preexisting levels of metareasoning, and of the memory and computation costs incurred by the additional analysis. Valuable meta-metareasoning depends clearly on the existence of flexible metareasoning strategies and on the availability of useful control knowledge at the new level of metareasoning. The optimization of the comprehensive EVC under the pressures of problems posed by an environment, and fundamental constraints on the reasoning capabilities of an agent, could dictate specific configurations of compiled knowledge and deliberative machinery at

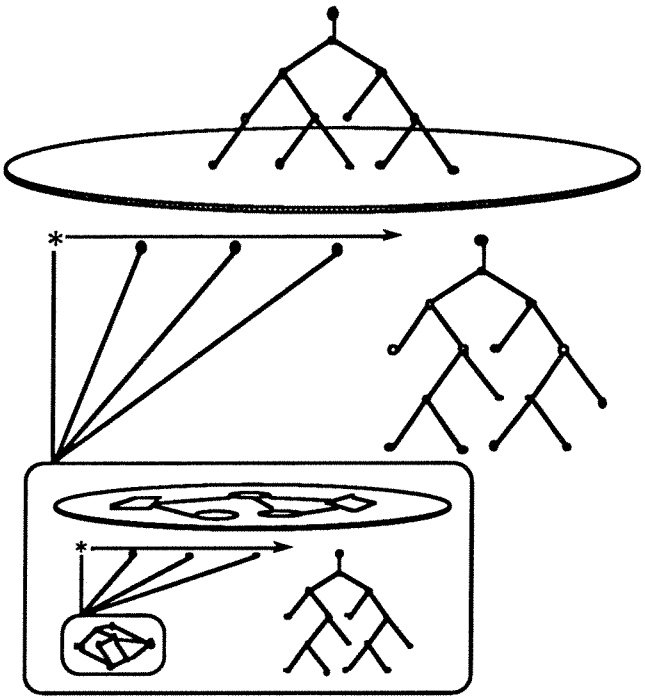


Figure 11: We can gain additional flexibility by adding new levels of metareasoning that apply knowledge about metareasoning to control a primary metareasoner. The model displayed here portrays the use of a compiled situation–action trie to control decisions about the value of explicit metalevel deliberation versus use of compiled rules to select the best metalevel strategy.

each level. Elucidating principles of optimal multiple-level metareasoning is a open problem area.

No matter how many metalevels we add, it is important to endow an agent with the ability to respond, in a reflex fashion, at the object level. In the general case, we wish to have a chain of situation–action rules from any metalevel to the object level, so as to provide efficient access to critical object-level reflexes. A more general integrated model comprises a *meta–meta-analysis* in the form of a tractable deliberative analysis or compiled situation–meta-meta-action rules. This meta-metareasoning problem controls decisions about the amount of metalevel deliberation versus object-level reasoning that should be performed, before an object-level action is taken. A sample compiled meta-metareasoning policy is the following: *(1) if an object-level situation–action rule is available, act; (2) if it is not available, deliberate for a time dictated by the metareasoning model.*

7.2 Addressing Problems of Analytic Regress

We note that questions of analytic regress tend to arise in discussions of metareasoning. If control is so valuable, why not control the controller itself, and why stop there? Will our agents have to grapple with infinite regress to optimize their decisions? Working to enhance the value of object-level computation by introducing metareasoning and control does not necessitate intractable metareasoning or infinite analytic regress. Concerns with analytic regress typically are based on assumptions about metareasoning problems that imply a need for the recursive application of meta-analysis.

One assumption, implicit in some discussions of analytic regress, is that the metareasoning problem will be at least as complex as the base problem and, therefore, will require the same kind of control as the base problem. We speculate that such an assumption of metareasoning complexity is based on a sense that a metareasoning problem must represent a great portion of the base problem that it is attempting to control. We may, indeed, be able to construct metaproblems with a complexity that rivals or exceeds that of the base problem. However, we can frequently build much simpler control problems that greatly enhance the comprehensive value of reasoners. These meta-analyses address particular aspects of object-level problem solving, and make use of specific classes of metaknowledge; they do not capture the full complexity of representation or inference at the base level.

Another assumption is that control decisions are necessarily sensitive to small changes in the accuracy of a metareasoning analysis. If this were so, optimizing the value of control decisions might invoke a large, or an infinite, number of metalevels. We believe, however, that this is the exception, rather than the rule. Within our models, control decisions appear to be insensitive to small changes in the accuracy of the analysis. When this not the case, we can seek to characterize a relevant spectrum of meta-metareasoning implications during the design phase; our experience leads us to be optimistic about our ability to dodge recursive real-time expansion of meta-analysis by analyzing the expected behavior of a metareasoner.

The theme of our metareasoning research is to extend object-level analyses by invigorating the base models with one or several levels of tractable metalevel decision problems and approximations, and to analyze the problem of metareasoning allocation at design time. We

believe that work on tractable, closed-form meta-analyses hold the greatest opportunities for empirical and theoretical research. Nevertheless, there may be cases where the fundamental consitution of an agent and its environment dictates as optimal the application and management of a large or (in theory) infinite, tower of meta-analyses [26]. There has been some discussion of techniques for handling the analytic regress problem. Kripke has discussed a problem with infinite regress that arises in defining truth [35]. He applied a mathematical tool known as a transfinite hierarchy to grapple with the problem. Lipman has peformed analysis of an infinite-regress problem by applying similar techniques within a game-theoretic framework [37]. He demonstrates that, within his model, there is an equivalence between a base decision problem and an infinitely recursive analysis.

We have suggested that optimization problems that imply large hierarchies of metareasoners provide rich problems for convergence and stability analyses. For example, the EVC of an agent, based on a theoretical prescription for an infinite hierarchy of metareasoning analyses, might converge tractably to a positive value in a manner similar to the way the sum of an infinite series may closely approach a real number with a small number of terms. Also, perturbation and stability analyses developed within control theory may be useful in determining optimal configurations of large numbers of interrelated meta-analyses, especially for cases where control decisions are sensitive to minute perturbations of accuracy of analysis at a large number of metalevels. Great sensitivity of control reasoning to the completeness of metareasoning analyses can focus attention on a search for less sensitive, more stable levels of approximate meta-analyses. Convergence and stability analyses hold the promise of dictating finite concrete architectures and policies for an ideal bounded-optimal agent that rely on unwieldy recursive meta-analyses.

8 Learning, Compilation, and Rationality

I would like to end with some discussion on the importance of studying techniques for allowing the deliberative and compiled knowledge of our reasoners to be modified and extended based on an agent's expectations and experience within a specialized environment. From our perspective on rationality, short-term and long-term learning are viewed as local and more permanent compilation strategies, respectively. These dynamic compilation strategies allow an agent that has been endowed with an architecture that supports reflex, reasoning, and metareasoning, and with intrinsic knowledge for surviving in a broadly defined environment, to prosper when immersed in a specialized world. We view learning as the pursuit of optimal EVC and associated comprehensive value, in accordance with the distribution of problems seen in a specialized environment. There are several ways to combine the EVC, associated with an agent's responses to distinct challenges, into a more global value of agency over a period of time. We discuss the valuation of behavior over time in [25].

It may be optimal to allow a reasoner to allocate a quantity of memory for the dynamic generation and caching of platform knowledge. Depending on the constitution of an agent and the problems faced by that agent, it may be optimal to generate specific platform knowledge continually and, afterward, to destroy the knowledge to free up memory for the next local challenge. As an example, it can be useful to generate local expectation-driven partial solutions for probabilistic inference with the bounded-conditioning inference method

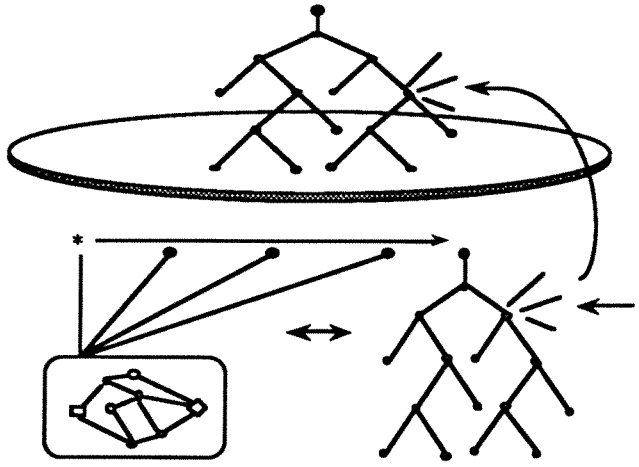


Figure 12: From the perspective of decision-theoretic compilation, learning can be viewed as enhancing or extending classes of compiled knowledge. Such learning can take place at the object level or within the metareasoning problem. In the case of refining platform knowledge, we wish reasoners to make complementary modifications in associated deliberative mechanisms and control policies.

during the idle time between observations in domains where observations impinge on an agent over time [30].

Knowledge about the interaction of situation–action, platform, and resource rules with deliberative mechanisms should be used in design-time selection of the best set of knowledge for a given quantity or cost of memory. Allowing for relatively permanent changes in compiled knowledge can enable agents designed for a range of environments to optimize their behaviors in specific environments. Utility-based rule-selection techniques can increase the effectiveness of reasoning at the object-level and at successive metalevels. Thus, we can view a portion of learning as the construction and storage of sets of reflex, platform, and resource knowledge that are tailored to particular problem tasks. Beyond the custom-tailoring and refinement of compiled knowledge, effective learning might also modify existing deliberative reasoning and metareasoning strategies. Such complementary refinement of deliberation could optimize the use of the compiled knowledge

9 Summary

We have reviewed issues surrounding the pursuit of rational behavior under resource constraints. We have endeavored to optimize the value of decisions by expanding traditional decision models into multilevel analyses that model cognitive processes, in addition to distinctions associated with problem challanges. We described the notion of partial computation, presented a general framework for optimizing the multiattribute utility of computation, and examined some of our work on the application of rational metareasoning to the control of fundamental computation tasks such as sorting, and for deliberation about rational belief and action. After presenting a deliberative approach to bounded-optimal decisions, we discussed more recent research on the integration of different classes of compiled knowledge

with deliberative machinery. In particular, we described the use of *situation–action rules* to gain access to direct action or meta-actions, *platform rules* to enhance deliberation at any level of analysis, and *resource rules* to lower the costs associated with delay, or to fend off deadlines. After discussing the compilation of metareasoning, we presented opportunities for developing integrated multilevel reasoners. Finally, we touched on short-term and long-term learning as compilation strategies that raise the expected value of a agent's behavior. Our investigation continues to pursue rational belief and action through analysis of the rich relationships among reflex, reasoning, and metareasoning.

Acknowledgments

John Breese, Bruce Buchanan, Gregory Cooper, George Dantzig, Tom Dean, David Heckerman, Ronald Howard, Nils Nilsson, Edward Shortliffe, and Patrick Suppes have provided useful feedback on PROTOS research.

References

[1] A. Agogino and K. Ramamurthti. Real-time reasoning about time constraints and model precision in complex distributed mechanical systems. In *Working Notes: Spring Symposium Series on AI and Limited Rationality*, pages 1–5, Stanford University, March 1989. American Association for Artificial Intelligence.

[2] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, Seattle, WA, July 1987. AAAI-87.

[3] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, Menlo Park, CA, 1983.

[4] M. Boddy and T. Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh IJCAI.* AAAI/International Joint Conferences on Artificial Intelligence, August 1989.

[5] J.S. Breese. *Knowledge Representation and Inference in Intelligent Decision Systems.* PhD thesis, Department of Engineering-Economic Systems, Stanford University, 1987. Also available as Technical Report 2, Rockwell International Science Center, Palo Alto Laboratory.

[6] R. Brooks. Planning Is Just a Way of Avoiding Figuring Out What to Do Next. Technical Report Working Paper 303, M.I.T. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1987.

[7] G.F. Cooper. Probabilistic inference using belief networks is NP-hard. *Artificial Intelligence*, 2, 1989. In press.

[8] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings AAAI-88 Seventh National Conference on Artificial Intelligence*, pages 49–54. American Association for Artificial Intelligence, August 1988.

[9] J. Doyle. Artificial intelligence and rational self-government. Technical Report CS-88-124, Carnegie-Mellon University, 1988.

[10] O. Etzioni and T. Mitchell. A comparative analysis of chunking and decision-analytic control. In *Working Notes: Spring Symposium Series on AI and Limited Rationality*, pages 42–45, Stanford University, March 1989. American Association for Artificial Intelligence.

[11] M.R. Fehling and J.S. Breese. A computational model for the decision-theoretic control of problem solving under uncertainty. Technical report, Rockwell International Science Center, April 1988. Rockwell Technical Report 837-88-5.

[12] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., New York, 1979.

[13] I.J. Good. Rational decisions. *J. R. Statist. Society B*, 14:107–114, 1952.

[14] I.J. Good. Dynamic probability, computer chess, and the measurement of knowledge. In E.W. Elcock and Michie D., editors, *Machine Intelligence 2*, pages 139–150. Wiley, New York, 1977.

[15] O. Hanson and A. Mayer. The optimality of satisficing solutions. In *Proceedings of Fourth Workshop on Uncertainty in Artificial Intelligence*, Minneapolis, MN, July 1988. American Association for Artificial Intelligence.

[16] O. Hansson and A. Mayer. Probabilistic heuristic estimates. In *Proceedings of the Second International Workshop on AI and Statistics*, Ft. Lauderdale, January 1989.

[17] D.E. Heckerman, J.S. Breese, and E.J. Horvitz. The compilation of decision models. In *Proceedings of Fifth Workshop on Uncertainty in Artificial Intelligence*, Windsor, Canada, August 1989. American Association for Artificial Intelligence.

[18] D.E. Heckerman and E.J. Horvitz. A principled approach to problem formulation. Technical Report KSL-89-67, Stanford University, Knowledge Systems Laboratory, Stanford, CA, June 1989.

[19] D.E. Heckerman, E.J. Horvitz, and B.N. Nathwani. Toward effective normative expert systems: The Pathfinder project. *Computers and Biomedical Research*, 1989. In press.

[20] E.H. Herskovits and G.F. Cooper. Algorithms for belief-network precomputation. Technical Report KSL-89-35, Stanford University, January 1989.

[21] E.J. Horvitz. Reasoning about inference tradeoffs in a world of bounded resources. Technical Report KSL-86-55, Stanford University, Knowledge Systems Laboratory, Stanford, CA, September 1986.

[22] E.J. Horvitz. Toward a science of expert systems. In Thomas J. Boardman, editor, *Proceedings of the 18th Symposium on the Interface of Computer Science and Statistics*, pages 45–52, Ft. Collins, Colorado, March 1986. American Statistical Association. Also available as Technical Report KSL-86-75: Knowledge Systems Laboratory: Stanford University, March 1986.

[23] E.J. Horvitz. The decision-theoretic control of problem solving under uncertain resources and challenges. Technical Report KSL-87-16, Stanford University, Knowledge Systems Laboratory, Stanford, CA, November 1987.

[24] E.J. Horvitz. Problem-solving design: Reasoning about computational value, tradeoffs, and resources. In *Proceedings of the NASA Artificial Intelligence Forum*, pages 26–43. National Aeronautics and Space Administration, November 1987. Also available as Technical Report KSL-87-64, Knowledge Systems Laboratory, Stanford University, October 1987.

[25] E.J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of Third Workshop on Uncertainty in Artificial Intelligence*, Seattle, Washington, July 1987. American Association for Artificial Intelligence. Also in L. Kanal, T. Levitt, and J. Lemmer, ed., *Uncertainty in Artificial Intelligence 3,* Elsevier, pps. 301-324.

[26] E.J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings AAAI-88 Seventh National Conference on Artificial Intelligence*, pages 111–116. American Association for Artificial Intelligence, August 1988.

[27] E.J. Horvitz, J.S. Breese, and M. Henrion. Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, 2:247–302, 1988. Special Issue on Uncertain Reasoning.

[28] E.J. Horvitz, G.F. Cooper, and D.E. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the Eleventh IJCAI*, pages 1121–1127. AAAI/International Joint Conferences on Artificial Intelligence, August 1989.

[29] E.J. Horvitz, D.E. Heckerman, K. Ng, and B.N. Nathwani. Heuristic abstraction in the decision-theoretic Pathfinder system. In *Proceedings of the 13th Symposium on Computer Applications in Medical Care*. IEEE Computer Society Press, October 1989.

[30] E.J. Horvitz, H.J. Suermondt, and G.F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of Fifth Workshop on Uncertainty in Artificial Intelligence*, Windsor, Canada, August 1989. American Association for Artificial Intelligence.

[31] R.A. Howard and J.E. Matheson. Influence diagrams. In R.A. Howard and J.E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, volume II, pages 721–762. Strategic Decisions Group, Menlo Park, CA, 1981.

[32] L. Kaelbling. An architecture for intelligent reactive systems. In A.L. Lansky and M.P. Georgeff, editors, *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*, pages 395–410. Morgan-Kaufmann, 1987.

[33] K. Kanazawa and T. Dean. A model for projection and action. In *Proceedings of the Eleventh IJCAI.* AAAI/International Joint Conferences on Artificial Intelligence, August 1989.

[34] D.E. Knuth. *Sorting and Searching.* Addision-Wesley, Reading, Massachusetts, 1973.

[35] S. Kripke. Outline of a theory of truth. *Journal of Philosophy*, 72:690–716, 1975.

[36] C. P. Langlotz, E. H. Shortliffe, and L. M. Fagan. Using decision theory to justify heuristics. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 215–219, Philadelphia, PA, 1986. AAAI-86.

[37] B.L. Lipman. How to decide how to decide hot to...: Limited rationality in decisions and games. Technical report, Carnegie Mellon University, Pittsburgh, February 1989.

[38] J.G. March. Bounded rationality, ambiguity, and the engineering of choice. *Bell Journal of Economics*, pages 587–608, 1978.

[39] J.E. Matheson. The value of analysis and computation. *IEEE Transactions on Systems Science, and Cybernetics*, 4:211–219, 1968.

[40] J. Pearl. A constraint-propogation approach to probabilistic reasoning. In L.N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 1*, pages 357–369. Elsevier Science, Amsterdam, 1986.

[41] S. Rosenschein and L. Kaelbling. The synthesis of digital machines with provable epistemic properties. In *Proceedings of the Conference on the Theoretical Aspects of Reasoning About Knowledge*, pages 83–98, Asilomar, CA, 1986. AAAI.

[42] S.J. Russell and E. Wefald. Multi-level decision-theoretic search. In *Proceedings of the AAAI Spring Symposium on Game Playing*, Stanford, CA, March 1988. American Association for Artificial Intelligence.

[43] S. M. Shugan. The cost of thinking. *Journal of Consumer Research*, 7:99–111, 1980.

[44] H.A. Simon. A behavioral model of rational choice. *Quarterly Journal of Economics*, 69:99–118, 1955.

[45] H.A. Simon. The theory of problem solving. *Information Processing*, 71:261–277, 1972.

[46] H.A. Simon. Rationality as process and as product of thought. *J. American Economic Association*, 68(2):1–16, 1978.

[47] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, NJ, 1947.

[48] S.R. Watson and R.V. Brown. The valuation of decision analysis. *J.R. Statist. Society A.*, 141(1):69–78, 1978.

[49] E. Wefald and S. Russell. Estimating the value of computation: The case of real-time search. In *Working Notes: Spring Symposium Series on AI and Limited Rationality*, pages 106–110, Stanford University, March 1989. American Association for Artificial Intelligence.

[50] M.P. Wellman. Formulation of tradeoffs in planning under uncertainty. Technical Report MIT/LCS/TR-427, Laboratory for Computer Science, Massachusetts Institute of Technology, August 1988.

Metareasoning in Modular Software Systems: On-the-Fly Configuration Using Reinforcement Learning with Rich Contextual Representations

Aditya Modi,[1] **Debadeepta Dey,**[2] **Alekh Agarwal,**[2]
Adith Swaminathan,[2] **Besmira Nushi,**[2] **Sean Andrist,**[2] **Eric Horvitz**[2]
[1]University of Michigan Ann Arbor, [2]Microsoft Research Redmond
[1]admodi@umich.edu, [2]{dedey, alekha, adswamin, besmira.nushi, sandrist, horvitz}@microsoft.com

Abstract

Assemblies of modular subsystems are being pressed into service to perform sensing, reasoning, and decision making in high-stakes, time-critical tasks in areas such as transportation, healthcare, and industrial automation. We address the opportunity to maximize the utility of an overall computing system by employing reinforcement learning to guide the configuration of the set of interacting modules that comprise the system. The challenge of doing system-wide optimization is a combinatorial problem. Local attempts to boost the performance of a specific module by modifying its configuration often leads to losses in overall utility of the system's performance as the distribution of inputs to downstream modules changes drastically. We present metareasoning techniques which consider a rich representation of the input, monitor the state of the entire pipeline, and adjust the configuration of modules on-the-fly so as to maximize the utility of a system's operation. We show significant improvement in both real-world and synthetic pipelines across a variety of reinforcement learning techniques.

1 Introduction

The lives of a large segment of the world's population are greatly influenced by complex software systems, be it the software that returns search results, enables the purchase of an airplane ticket, or runs a self-driving car. Software systems are inherently modular, i.e. they are composed of numerous distinct modules working together. As an example, a self-driving car has modules for sensors such as cameras, lidars which poll the sensors and output sensor messages, and a mapping module that consumes sensor messages and creates a high-resolution map of the immediate environment. The output of the mapping module is then input to a planning module whose job is to create safe trajectories for the vehicle. These distinct modules often operate at different frequencies; the camera module may be producing images at 120Hz while the GPS module may be producing vehicle position readings at 1000Hz. Furthermore, they may each have their own set of free parameters which are set via access of a configuration file at startup. For example, the software serving as the driver of a camera in the self-driving pipeline may have a parameter setting for the rate at which images are pulled from the camera and another parameter for the resolution of the images. Similarly, the function of the mapping module may be controlled by a parameter that specifies the maximum amount of memory it is allowed to consume, leading to the continual removal of information about more distant and thus less relevant map content.

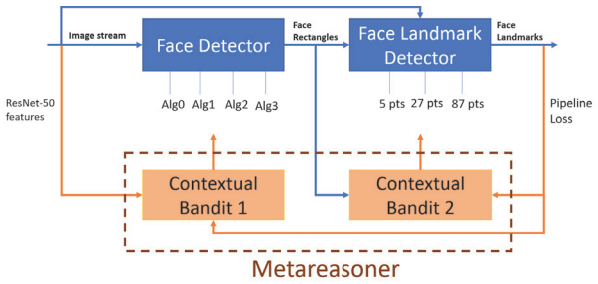


Figure 1: Face and landmark detection modular system. The input is an image stream to the face detection module which outputs locations of faces in the image which are then input to the face landmark detection module which outputs locations of eyes, nose, lips, brows etc on the detected faces. The metareasoning module receives the input stream of images along with intermediate outputs of the face detector to dynamically decide the configuration of the pipeline such that it optimizes the end system loss.

Large software systems typically are composed of a set of distinct modular components. The operating characteristics of all of the components are usually manually configured to achieve system performance targets or constraints like accuracy and/or latency of output. Configurations of parameters may result from the tedious and long-term tuning of one parameter at a time. Once such nominal configurations have been produced, they are then held constant during system execution. The reliance on such fixed policies in a dynamic world may often be sub-optimal. As an example, modules may take different amounts of time depending on the specific contents of the inputs they receive.

As a running example, we illustrate a pipeline for extract-

ing faces with keypoint annotations from images in Figure 1. A natural performance metric for the pipeline might blend the prediction latency and accuracy, where the latency of a face-detection module may vary dramatically based on the number of people in the camera view. In this case, one might prefer switching to a parameter setting which allows the face detector to sacrifice some accuracy but which is much faster hence raising the overall utility of the entire pipeline. Also modules which are upstream from the face detector like the camera driver module might ideally throttle back the rate at which it is producing images since most of these images will not get processed anyways, due to a bottleneck at the face detector module. Attempts to separately optimize distinct modules can often lead to losses in utility (Bradley 2010) because of unaccounted shifts in the distribution of outputs produced by upstream modules.

Revisiting the self-driving car example, a basic utility function is to simply navigate passengers to their destination safely and in a reasonable amount of time. Highlighting the contextuality again, the emphasis on driving time might be higher when trying to get to an important meeting or a flight than going grocery shopping. Furthermore, the utility function will typically be specific to the user and has to be inferred over time. Importantly, this is a complex pipeline-level feedback which is hard to attribute to individual components.

In this work, we leverage advances in representation and reinforcement learning (RL) to develop metareasoning machinery that can optimize the configuration of modular software systems under changing inputs and compute environments. Specifically, we demonstrate that by having a metareasoner continuously monitor the entire system we can switch parameters of each module on-the-fly to *adapt* to changing inputs and optimize a desired objective. We also study the distinction between attainable performance by choosing the best configuration for the entire pipeline as a function of just the initial input, versus further choosing the configuration of each module based on all the preceding actions and outputs. We experiment with a synthetic pipeline meant to require adaptivity to the inputs, and we find that by doing so at each module, we improve by roughly 50% or more over the best constant assignment, and typically by a similar margin over the choice of a configuration just as a function of the initial input. For the face and landmark detection pipeline 1, we use the activations of a pretrained neural network model as a contextual signal and leverage this rich representation of context in decisions about the configuration of each module *before* the module operates on its inputs. We characterize the boosts in utility provided via use of this contextual information, improving 9% or more across different utility functions as opposed to the best static configuration of the system. Overall, our experiments demonstrate the importance of online, adaptive configuration of each module.

2 Related Work

RL to control software pipelines: Decisions about computation under uncertainties in time and context have been described in (Horvitz and Lengyel 1997), which presented the use of metareasoning to guide graphics rendering under changing computational resources, considering probabilistic models of human attention so as to maximize the perceived quality of rendered content. The metareasoning guided tradeoffs in rendering quality under shifting content and time constraints in accordance with preferences encoded in a utility function. Principles for guiding proactive computation were formalized in (Horvitz 2001). (Raman et al. 2013) characterize a tradeoff between computation and performance in data processing and ML pipelines, and provide a message-passing algorithm (derived by viewing pipelines as graphical models) that allows a human operator to manually navigate this tradeoff. Our work focuses on the use of metareasoning to replace the operator by setting the best operating point for any pipeline automatically.

(Bradley 2010) proposed using subgradient descent coupled with loss functions developed in imitation learning in order to jointly optimize modular robotics software pipelines which often involve planning modules, when the modules are differentiable with respect to the overall utility function. This is not suited to most real-world pipelines with modules described not with parameters but lines of code. In this work we instead develop fully general methods, which only assume the ability to evaluate the pipeline. Another form of pipeline optimization is to accordingly pick or configure the machine where each module should be executed. Methods in this ambit (Mirhoseini et al. 2017) are complementary to this work in that optimizing the pipeline configuration per se remains a problem even with optimal device placement.

RL in distributed system optimization: The use of machine learning for optimizing resource allocation in distributed systems for data center and cluster management has been very well studied (Lorido-Botran, Miguel-Alonso, and Lozano 2014; Demirci 2015; Delimitrou and Kozyrakis 2013; 2014). Many of these techniques use supervised learning as well as collaborative filtering for resource assignment, which rely on the assumption of having a rich set of processes in the training data and might as a result suffer from eventual data bias for new workloads. Most recently, the use of reinforcement learning for learning policies which dynamically optimize resources such that service level agreements can be better satisfied has received a lot of attention especially with the rise of 'deep' reinforcement learning ((Li 2017)). Methods using model-free methods (Mao et al. 2016; Xu, Rao, and Bu 2012) have shown promise as modeling such large-scale distributed systems is a challenge in itself. Similarly, RL has found impressive success in energy optimization for data centers (Gao 2014; Memeti et al. 2018).

RL for scheduling in operating systems: Even at the single machine level, RL has found promise for thread scheduling and resource allocation in operating systems. For example (Fedorova, Vengerov, and Doucette 2007; Hanus 2013) use RL-based methods to learn adaptive policies which outperform the best statically optimal policy (found by solving a queuing model) as well as myopic reactive policies which greedily optimize for short term outcomes. The problem of scheduling in operating systems however differs from pipeline optimization in two fundamental ways. First, the operating system (as well as the scheduler) is oblivious to accuracy dependencies between different processes or threads. Second, due to either architectural or generality con-

straints, schedulers do not optimize process-level parameters but mainly focus on machine configuration.

3 Problem Definition

3.1 Formal Setting and Notation

A pipeline of M modules can be viewed as a directed graph where each node j is a module and an edge from j to k represents module k consuming the output of j as its input. We assume the graph does not have any cycles. Without loss of generality, let the modules be numbered according to their topological sort; i.e. j refers to the index of a module in a linear ordering of the DAG[1]. For each module j, we have a set of possible configurations—these are the actions that are available for the metareasoner to choose from. We denote this set by A_j. A module j can then be viewed as a mapping from its inputs $x \in {S_j}^{in}$ to outputs $z \in {S_j}^{out}$, and each configuration $a \in A_j$ implies a different mapping. As a running example, we will consider the face detection pipeline of Figure 1. The pipeline contains two modules with module 1 having 4 choices and module 2 having 3 choices. The input space to the first module ${S_1}^{in}$ is the space of images (possibly in a feature space). The output space ${S_1}^{out}$ is the same as ${S_2}^{in}$ and can encode the image, the locations of faces in the image, and the latency induced by the first module.

The quality of a pipeline's operation is measured using a loss function denoted by $L : {S_M}^{out} \mapsto \mathbb{R}$. In the example pipeline of Figure 1, the outputs from the landmark detector can be labeled by human evaluators to assess accuracy and L can be a complex trade-off between the latency incurred by the overall pipeline in processing an image vs. the accuracy of the detected landmarks. If labels are not available, accuracy might be inferred from proxies such as an incorrect denial of authentication for a user based on the landmark detector output, which can be observed when the user authenticates via alternative means such as a password. Crucially, we only observe the value of this loss-function for the specific outputs $z \in {S_M}^{out}$ that the pipeline generates based on a certain configuration of actions at each module in response to an input x. We highlight that the loss function L can be any function mapping the pipeline's final output and system state to a scalar value, such as a passenger's satisfaction with a ride in a self-driving car as discussed in Section 1.

A metareasoner can be represented as a collection of (possibly randomized) policies $\pi := \{\pi_1 \dots \pi_M\}$, where $\pi_j : {S_j}^{in} \mapsto \Delta(A_j)$ specifies a context-dependent configuration of the module and $\Delta(A_j)$ is the set of distributions over the action set A_j. We abuse the notation for ${S_j}^{in}$ here to denote any succinct representation of the preceding pipeline component's outputs, actions and system state variables which are needed to choose the appropriate action for module j. The pipeline receives a stream of inputs and we use t to index the inputs. At time t, the pipeline receives an initial input $x_t^1 \in S_1^{in}$, based on which an action $a_t^1 \sim \pi_1(x_t^1)$ is picked at the first module and it produces an intermediate output z_t^1. This induces the next input $x_t^2 \in {S_2}^{in}$ at the second module, at which point the policy π_2 is used to pick the next action and so on. At each intermediate module j, the input x_t^j depends on the outputs of all its parents in the DAG corresponding to the pipeline and we assume that the input spaces ${S_j}^{in}$ are chosen appropriately so that a good metareasoner policy for module j can solely depend on x_t^j instead of having to depend explicitly on the outputs of its predecessors. As a result, the interaction between the metareasoner and the environment can be summarized as follows:

1. $x_t^1 \in {S_1}^{in}$ is fed as input to the pipeline.
2. metareasoner chooses actions for each module based on the output of its predecessors and induces a trajectory: $(x_t^1, a_t^1, z_t^1, \dots, x_t^M, a_t^M, z_t^M)$; eventual output of the pipeline is z_t^M.
3. Observe loss $L(z_t^M)$.

Formulated this way, the task of the metareasoner can be viewed as an episodic fixed-horizon reinforcement learning problem, where the state transitions are deterministic (although the initial input can be highly stochastic, such as an image in the face detection example). Each input processed by the pipeline is an episode, the horizon is M, actions chosen by policies for the upstream modules affect the state distribution seen by downstream policies. The feedback is extremely sparse with the only loss being observed at the end of the pipeline. The goal of the metareasoner is to minimize its average loss: $\frac{1}{T}\sum_{t=1}^{T} L(z_t^M)$, and the ideal metareasoner can be described as:

$$\underset{\pi_1 \dots \pi_M}{\arg\min} \sum_{t=1}^{T} \mathbb{E}_{\pi_1,\dots,\pi_M}\left[L(z_t^M) \mid x_t^1\right] \doteq J(\pi). \qquad (1)$$

Our goal is to learn a metareasoner during the live operation of the pipeline. Since we only observe pipeline losses for the current choices of the metareasoner's policies, we must balance exploration to discover new pipeline configurations, and exploitation of previously found performant configurations. In such explore-exploit problems, we measure the average loss accumulated by our adaptive learning strategy as a benchmark; a lower loss is better. A better learning strategy will quickly identify good context-dependent configurations and hence have lower average loss as T increases.

3.2 Challenges

In this section we highlight the important challenges that a metareasoner needs to address.

Combinatorial action space: Viewing the entire pipeline as a monolithic entity, with an aim to find the best fixed assignment for each module with no input dependence, leaves the metareasoner with combinatorially many choices to consider. This can quickly become intractable even for modest pipelines (e.g. See Figure 2), despite the use of the simplest possible static policy class.

[1]For our solution methods, all we need is the raw input given to a module, current system parameters, and statistics of modules which have finished the computation. In general though, having an architectural graph helps because if the architecture is different from a pipeline, many modules may have finished the computation but not all them may be relevant to the current module. Thus, we use the topological sort for a DAG pipeline to establish a dependency order and easily fix an informative state representation for the policy.

Adaptivity to inputs: Having a static action assignment per module is overly simplistic in general and we typically need a policy for manipulating configurations that is context-sensitive. For example, in Figure 4, we observe that the number of faces in the input image implies a fundamentally different trade-off between latency and accuracy.

Credit Assignment: Since we only observe delayed episodic reward, we do not know which module was to blame for a bad pipeline loss. For non-additive losses, this is especially challenging (III et al. 2018).

Exploration: Pipeline optimization offers a fundamentally challenging domain for exploration. Though we employ ideas from contextual bandits here for exploration, we anticipate future directions that explore by using pipeline structure to derive better learning strategies.

4 Methods

The methods we outline now each address some of the challenges in Section 3.2. The simplest strategy is a non-adaptive approach that effectively handles combinatorial actions (Section 4.1) to return a locally optimal static assignment. A simple context-sensitive strategy that views the problem as a contextual bandit, and is vulnerable to a combinatorial scaling with pipeline size (Section 4.2). Finally, the most sophisticated strategy we develop produces a context-adaptive policy, exploits pipeline structure to learn per-module policies and uses policy-gradient algorithms to quickly reach a locally optimal configuration policy (Section 4.3).

4.1 Greedy Hill Climbing

The simplest (infeasible) strategy for pipeline optimization with input examples $x_1^1, \ldots, x_T^1$ is to brute-force try every possible configuration for each of the T inputs and pick the configuration that accumulates the lowest loss. This strategy will identify the best non-adaptive (i.e. context-insensitive) configuration, but needs $T \cdot \prod_{j=1}^{M} |A_j|$ executions of the pipeline to find this configuration. Since this is intractable even for modest values of T and A_j (especially in real-time), we now describe a tractable alternative to find an approximately good configuration via random co-ordinate descent.

Rather than identifying the best configuration, suppose we aim to find a "locally optimal" configuration – that is, for every module, if we held all other module configurations fixed then deviating from the current configuration can only worsen the pipeline loss. To achieve this, we begin by randomly picking an initial configuration. In each epoch, we first sample K out of T examples uniformly with replacement from the dataset, where K is a hyperparameter that can be set based on the available computational budget. We then choose one of the modules $j \in \{1, 2, \ldots, M\}$ uniformly at random and keep the configurations of all other modules fixed. We cycle through every possible action for that module (using, for instance, $K/|A_j|$ examples for each choice of action at this module) and pick the action that achieves the lowest accumulated loss. We then repeat this process until our training budget is exhausted, or we cycled through every module without changing the configuration (which indicates a local optimum). This is akin to a greedy hill-climbing strategy, and has been used in many diverse applications of combinatorial optimization as an approximate heuristic, for instance in page layout optimization (Hill et al. 2017). More sophisticated variants of this approach can use best-arm identification techniques during each epoch, but fundamentally, this strategy finds an approximately optimal context-insensitive policy.

4.2 Global Bandit from Initial Input

For many real-world pipelines, the modules' characteristics are sensitive to the initial input, meaning that a context-insensitive policy can be very sub-optimal w.r.t. the pipeline loss. This motivates our approach to find a context-adaptive policy using contextual bandit (henceforth CB) algorithms.

A CB algorithm receives a context x_t in each round t, takes an action $a \in A$ and receives a reward r_t. The algorithm learns a policy $\pi : x \mapsto \Delta(A)$ that is context-sensitive and adaptively explores/exploits to maximize $\sum_t r_t$. In our setting x_t is the input example to the pipeline, A is the Cartesian product of all module-specific configurations and the reward is simply the negative of the observed pipeline loss.

In our experiments, we use a simple CB algorithm that uses Boltzmann exploration (see e.g. (Kaelbling, Littman, and Moore 1996)). Concretely, the policy is represented by a parametrized scoring function $s_\theta : {S_1}^{in} \times A \mapsto \mathbb{R}$. The score for each global configuration is computed $s_\theta(x, a)$ and the policy is a softmax distribution of these scores:

$$\pi_\theta(a \mid x) = \frac{\exp(\lambda s_\theta(x, a))}{\sum_{a'} \exp(\lambda s_\theta(x, a'))}, \tag{2}$$

where $\lambda > 0$ is a hyperparameter that governs the trade-off between exploration and exploitation. The score function is typically updated using importance-weighted regression (Bietti, Agarwal, and Langford 2018) (henceforth `IWR`); that is, if we observe a reward r_t after configuring the pipeline with action $a_t \sim \pi_\theta(a \mid x_t)$, then the score function is optimized to minimize $\frac{1}{\pi_\theta(a_t|x_t)}(r_t - s_\theta(x_t, a_t))^2$.

These contextual bandit algorithms can very effectively find context-sensitive policies π and adaptively explore promising configurations. However, by viewing the entire pipeline as one monolithic object with combinatorially many actions, they cannot scale to even moderate-sized pipelines.

4.3 Per-module Bandit: Using Immediate Observations

The contextual bandit approach of Section 4.2 does not scale well with the size of the pipeline, but it does guarantee (under mild assumptions, like an appropriate schedule for λ, see e.g. (Singh et al. 2000)) that we will eventually find the best context-adaptive policy expressible by our scoring function s_θ. It also does not capture the outputs of prior modules in choosing the configuration at a successor, which can be vital such as when a previous module incurs a large latency. Suppose we again relax the goal to instead find an approximately good "locally optimal" policy. Our key insight is to now employ a CB algorithm for each module, so that the algorithm for module j only needs to reason about A_j actions. Moreover, for each module, the metareasoner can use up-to-date

information (e.g. latencies introduced by upstream modules) as part of the context for the downstream bandit algorithm.

One can again perform a variant of randomized co-ordinate ascent as in Section 4.1, holding all but one module fixed and running a CB algorithm for that module. This ensures that each bandit algorithm faces a stationary environment and can reliably identify a good context-sensitive policy quickly. However, this can be very data-inefficient; we will next sketch an actor-critic based reinforcement learning algorithm that can apply simultaneous updates to all modules.

Suppose we consider stochastic policies of the form (2) for a module j, but where $x_t^j \in S_j^{in}$ and $a_t^j \in A_j$. A common approach to optimize the policy parameters θ is to directly perform stochastic gradient descent on the average loss, which results in the policy gradient algorithm. Specialized to our setting, an unbiased estimate of the gradient for the parameters θ_j of π_j, that is $\nabla_{\theta_j} J(\theta)$ (recall (1) is given by $L(z_t^M)\nabla_{\theta_j} \log \pi_\theta(a_t^j \mid x_t^j)$ since the loss is only incurred at the end. Typically, policy gradient techniques use an additional trained critic $C(x_t^j)$ as a baseline to reduce the variance of the gradients (Konda and Tsitsiklis 2000). We train the critic to minimize the mean squared error between the observed reward and the predicted reward, $(L(z_t^M) - C(x_t^j))^2$.

5 Experiments

The algorithms discussed in the previous section are tested on two sets of pipelines: a synthetic pipeline with strong context dependence and a real-world perception pipeline. For all our experiments, we use a PyTorch based implementation (Paszke et al. 2017) with RMSProp (Hinton, Srivastava, and Swersky 2012) as the optimizer. For hyperparameter tuning, we perform a grid search over the possible choices. The common hyperparameters for both methods are: (i) Learning rate $\in \{0.0001, 0.0004, 0.001, 0.005\}$, (ii) Minibatch size $\in \{5, 10, 20, 50, 100\}$ and (iii) ℓ_2-weight decay factor $\in \{0.01, 0.05, 0.1, 1\}$. All our plots include 5 different runs with 5 randomly chosen random seeds with standard error regions. The specific details for each algorithm are as follows:

Greedy hill-climbing For finding the greedy step in each iteration, we use a minibatch of 1000 samples per action ($K = A_j * 1000$). The procedure is run until it converges to a fixed assignment. In the plots, we outline this as the non-adaptive baseline with which each method is compared. The final assignment obtained by the procedure is evaluated using Monte Carlo runs with sufficiently large number of samples from the input distribution (synthetic pipeline) or using samples present in a holdout set (face detection pipeline).

Global contextual bandit The policy parameters consist of a single policy that maps the input x_t^1 to a configuration for the entire pipeline, and policy class is a neural network with a single hidden layer. The inverse temperature coefficient for Boltzmann exploration, λ, is considered to be a hyperparameter. We use the `IWR` loss with minibatches to perform updates to the policy.

Per-module contextual bandit The policy function at each module is a single hidden layer neural network with a softmax layer at the end. We use the policy gradient update rule as discussed in Section 4.3. The context for each module is the concatenation of the sequence of actions chosen for previous modules, current latency and the initial input to the system. Additionally, for each module, we implement a critic which predicts the final loss of the pipeline for the given context as described in Section 4.3. The critic is again a single hidden layer neural network with a single output node and is trained using squared loss over the observed and predicted loss. We use the same learning rate for both networks. We use minibatches for training the networks for each module and these are concurrently updated for each minibatch. In addition, we also use entropy regularization weighted by `ent_wt` with the policy gradient loss function (Haarnoja et al. 2018).

For hyperparameter tuning, we choose the setting with the minimum cumulative loss across the input stream.

Method	Hyperparameter choices
Global CB	$\lambda \in \{0.1, 0.3, 1, 5, 10\}$
Per-module CB	`ent_wt` $\in \{0.01, 0.03, 0.1, 0.3, 1\}$

Table 1: Algorithm specific hyperparameter choices

At a high-level, our experiments seek to uncover the importance of adaptivity to the inputs in configuring the pipeline. To capture practical trade-offs, we consider loss functions which combine the latency incurred for an input, along with the accuracy of the final prediction.

5.1 Synthetic Pipelines

We begin with an illustrative synthetic pipeline designed to highlight: (1) benefits of adaptivity to the input over a static assignment, and (2) infeasibility of the global CB approach for even modestly long and representationally simpler pipelines. The structure of the synthetic pipeline with n modules is a linear chain of length n. Each module has two possible actions: 0 and 1 (cheap/expensive action) which incur a latency cost of 0 and 1 respectively. Inputs to the pipeline consist of uniformly sampled binary strings from $\{0, 1\}^n$, with the i_{th} bit encoding the preferred action for module i. If the i_{th} bit is set to 0, both actions give an accurate output and if it is 1, only the expensive action gives an accurate output. If we make an incorrect prediction at module i, then the final prediction at the end of the pipeline is always incorrect. At each episode t, we provide an input to the pipeline by first sampling a random binary string as mentioned above, but then add uniform noise in the interval $[-0.3, 0.3]$ to each entry and this perturbed input constitutes the initial context x_t^1 for the pipeline. The loss function for the final output of the pipeline is $\ell(a) := \frac{4}{n^2}(\text{latency} - n/2)^2 + \text{error}$.

We center the latency term at $n/2$, which is the latency of the optimal policy that routes each input perfectly to the cheapest action that makes the correct prediction for it and the normalization keeps this term in $[0, 1]$. The second term measures the error in the eventual prediction, which requires each module to make an accurate prediction. The value is set to 1 for an incorrect output and 0 otherwise. While the initial input encodes the optimal configuration,suited to global CB, there is further room to adapt. When module i makes an error, then all modules $j > i$ should pick the cheap action.

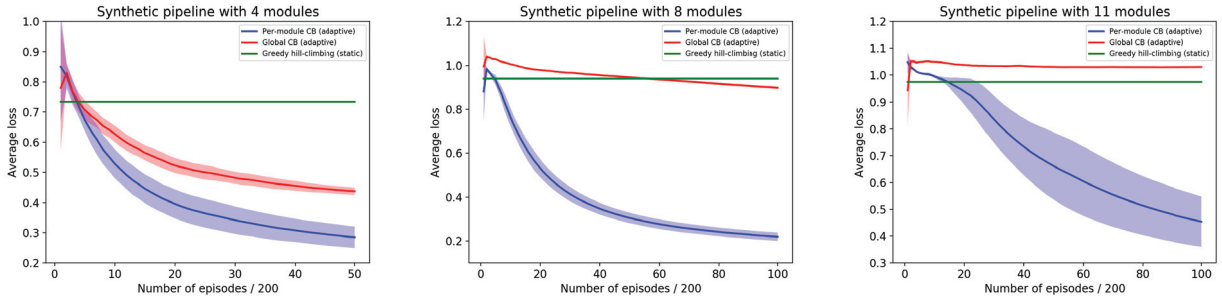


Figure 2: Average loss as a function of the number of examples for the synthetic pipeline. The flat line corresponds to the expected loss of the best constant assignment. The shaded region represents one standard error over 5 runs.

We show results of our algorithms for $n = 4, 8$ and 11. For static assignments, we compute both the solution of the greedy hill climbing strategy and a brute force search over all assignments, which results in similar average losses under the input distribution. The context for each module for per-module CB contains the pipeline's input, a binary string to denote upstream actions and the current latency. We use ReLU activations with the number of hidden layers for each network in our experiments as the average of input dimension and the output dimension. For instance, for global CB, the number of hidden units for $n = 4$ is $h = 10$.

We show the evolution of the average loss as a function of the number of examples for different values of n in Figure 2. Our results show significant gains for being adaptive over the constant assignment baseline in all the plots. For $n = 4$, the total number of assignments is 16 and it can be clearly seen that global CB is effective when compared to the per-module counterpart. However, global CB is slower in convergence than per-module CB. For $n = 8$, the difference between the two is more pronounced as the per-module CB method converges rapidly. For $n = 11$, the total number of assignments for the pipeline is 2048 and global CB completely fails to learn a better adaptive policy. The per-module CB has a slower convergence in this harder case, but still improves upon the best constant assignment extremely quickly.

Figure 3: Example face and landmark detections from COCO validation set. (Left) Face detected (blue rectangle) and landmarks detected within the face (blue dots). The red dots represent undetected groundtruth face landmarks by the pipeline. (Right) False face detections and wrong landmarks within the rectangles.

5.2 Face and Landmark Detection

Pipeline and dataset: We use a two-module production-grade real-world perception pipeline service to empirically study the efficacy of our proposed methods (Figure 1). The first module is a face detection module which takes as input an image stream and outputs the location of faces present in the image as a list of bounding box rectangles. This module has four different algorithms for detecting faces. The exact details of the algorithms are proprietary and hence we only have black-box access to them. We benchmarked the latency and accuracy of the algorithms on 2689 images from the validation set of the 2017 keypoint detection task of the open source COCO dataset (Lin et al. 2014). COCO has ground truth annotations of up to 17 visible keypoints per person in an image. We notice that not only do each of the algorithm choices have large variation in latency and accuracy on average when compared to each other, more crucially their latencies and accuracies vary drastically with the number of true faces present in the incoming images, i.e. they are *context dependent*. Specifically, we observe that latency drastically increases with the number of faces present in the image. Figure 4 shows the latencies of all four detection algorithms vs. number of true faces present in the image. Note that different algorithms have different latencies *on average* with Algorithm 0 being the fastest (~ 0.2 seconds) and Algorithm 3 the slowest (~ 2.5 seconds).

The second module is a face landmark detection module which takes as input the original image and the predicted face rectangles output by the face detection module and computes the location of landmarks on the face like nose, eyes, ears, mouth etc. There are three different landmark detector algorithm choices: 5 points, 27 points or 87 points landmark detector. Again we observe in our benchmarking that the landmark detector which outputs 87 points takes the most time at 0.25 ms per image on average vs. 0.17 ms and 0.08 ms per image for the 27 and 8 points algorithms respectively. Since the landmark detectors are applied on each face rectangle detected by the face detector, the computational time required goes up proportional to the number of faces. Figure 3 shows example face detections and landmarks detected on images from the validation sets of the COCO dataset.

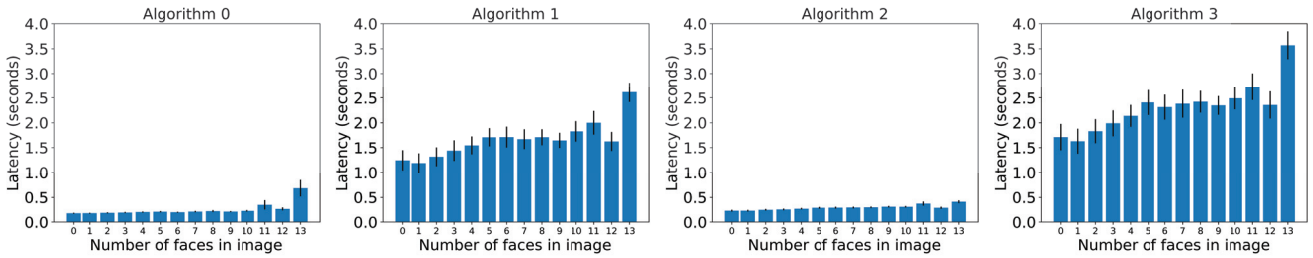


Figure 4: Face detection algorithm choices vs. latency in seconds as a function of true number of faces present in the image. Algorithm 0 and 2 are much faster than Algorithm 1 and 3. All algorithms exhibit increasing latencies as the number of faces goes up in the image.

Accuracy calculation: For evaluating when a prediction by the face detection module is a true/false positive/negative, we closely follow the scheme laid out in the COCO Keypoints evaluation page (Lin et al. 2014). Specifically a rectangle location on the image is considered a true positive if it is within 30 pixels of a ground truth face annotation which is quite conservative as the images we use are all resized to constant size of 1280(W) $\times$ 960(H) pixels. Otherwise, it is marked as a false positive. Ground truth faces which are not "covered" by any of the predicted faces cause an entry in the false negative count. If an image contains no faces and the face detection module also predicts no faces then we count such scenarios as true negatives.

For the face landmark module, we mark a prediction as a true positive if it is within 5 pixels of the ground truth landmark, else a false positive. All landmarks not "covered" by any of the predicted faces are counted as false negatives. Since the COCO keypoint annotations include only 17 keypoint annotations on the entire human body including only 5 face landmarks, we don't penalize predictions of the 27 or 87 landmark detection algorithms which are not within threshold distance of any ground truth landmark as that unfairly counts as false positives (due to lack of ground truth annotations).

Results: The dataset of 2689 images is divided into train and test sets of size 2139 and 550 respectively. For training, we use minibatches randomly sampled from the training set and test curves are plotted using the average loss over the complete test set.[2] We use the embedding from the penultimate layer of ResNet-50 (He et al. 2016) as the contextual representation for each image for both adaptive methods. Thus, the context is a 1000 dimensional real valued vector. For per-module CB, the first module's policy network gets the embedding as input whereas the second one gets additional concatenated values of number of faces detected by module 1 and its latency. All networks here have a hidden layer with 256 units with ReLU activations. For evaluating the final loss function of the pipeline, we consider three metrics:

- **Pure latency:** Squared loss between the pipeline's latency and a threshold t_0: $\ell(a) := (\text{latency} - t_0)^2$. For fair treatment across all inputs, just minimizing latency might not always be the actual goal and will result in a trivial policy which learns to pick the least expensive option for all inputs. Our metric incentivizes the system to stay around the *threshold* and penalizes under-utilization.
- **Latency and accuracy:** In addition to the squared distance, we now consider the false negative rate (FNR) of the pipeline for the landmarks detected in each image. Since false negative rate is always in $[0, 1]$, it is robust to different number of landmarks in different images as well as different number of predicted landmarks(5, 27 and 87), unlike a direct classification error. In this case, $\ell(a) := (\text{latency} - t_0)^2 + \text{FNR}$.
- **Latency, accuracy and false detection penalty:** For the face detection module, in many cases there are non-zero false positives. This further increases the number of false positive landmarks for those cases and therefore we add another penalty of the false discovery rate for face detection.

In our experiments, we choose a value of $t_0 = 1.3$ and $t_0 = 1.7$ for all three loss functions for the pipeline. Note that, if one tries to optimize total latency of the pipeline, then the non-adaptive solution of choosing the cheapest action for both modules works well. Therefore, we choose the bell shaped squared loss for latency which reflects the specification of aiming for a target latency. We show the test set performance in Figure 5. For $t_0 = 1.3$, we see a more interesting contrast across the two methods, whereas, for $t_0 = 1.7$, both Global CB and Per-module CB exhibit similar performance. Per-module CB and global CB show improvement for all loss functions against the constant assignment baseline found by greedy hill climbing. The numbers in Table 2 show the context-dependency of the pipeline. The benefits of algorithms which are able to effectively utilize context (Global CB and Per-module CB) is really highlighted in the parts of the dataset with more faces. As the number of faces in an image increases, the percentage gain increases as well. The observed gains of approximately 15, 22 and 24 percent in respecting the utility function are arguably significant for sensitive mission-critical applications.

[2] As the number of episodes is much larger than the size of our data set, algorithms can overfit to the data set. So we evaluate the average test performance on held out examples as a proxy.

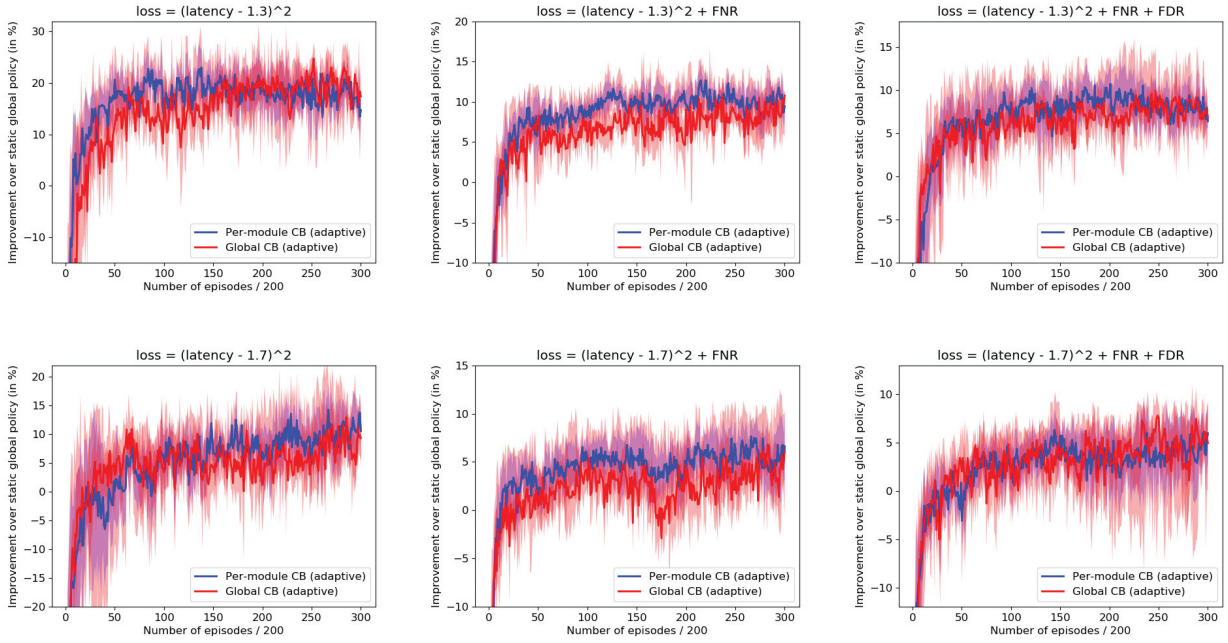


Figure 5: Test performance curves for the Face Detection and Landmark pipeline with $t_0 = 1.3$ (top) and $t_0 = 1.7$ (bottom). The Y-axis is the performance percentage improvement over static global policy after every 200 episodes of learning on held-out examples. The plots use a latency-based loss (left), latency and FNR (middle) and latency, FNR and FDR (right). The adaptive approaches significantly improve over the best fixed configuration in all cases. Shading represents std. error across 5 runs.

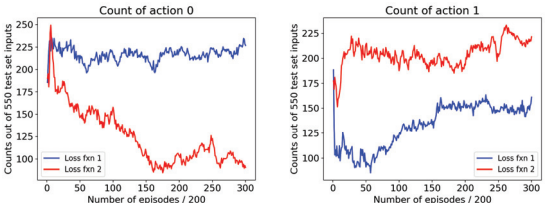


Figure 6: Action counts in module 2 for per-module CB

#Faces	#Images	Global CB	Per-module CB
$\geq$3	124	11.82%	15.23%
$\geq$4	83	18.58%	22.51%
$\geq$5	63	23.04%	24.12%

Table 2: Performance percentage improvement over static global policy broken down by the number of true faces in the image for latency and accuracy loss with $t_0 = 1.3$.

Although these two methods are hard to distinguish on average, we think this is due to the small length of the pipeline and the intermediate context for the second module's policy not being very informative. In order to show that the adaptivity to the final loss function influences the chosen actions, a comparison between the action counts for two different loss functions can be seen in Figure 6.

6 Discussion and Conclusion

We observe that contextual optimization of software pipelines can provide drastic improvements in the average performance of the pipeline for any loss function. Our experiments show that for small pipelines, both global CB and per-module CB can give potential improvement over a static assignment. However, these experiments should only be considered as a controlled study of the power of contextual optimization and there are additional caveats which we defer for future work:

Computational overhead: In addition to the pipeline's latency, any metareasoning module will add to the cost. In our experiments, the total time for inference and updates is less than 5-7 `ms` per input which is orders of magnitude less the the pipeline's latency. Moreover, making the pipeline configurable in real-time might induce further communication/data re-configuration costs. We focus on the potential improvements from adaptivity in this paper and leave the engineering constraints for future work.

Non-stationarity during learning: For the per-module CB algorithm, the input given to each network is ideally the input for the corresponding module. Changes in the configuration can vary the distribution of the inputs to these modules drastically. The pipelines in our experiments do not showcase this issue. We ignore this aspect in our current exposition and leave a more involved study to future work.

In summary, we presented the use of reinforcement learning to perform real-time control of the configuration of a modular system for maximizing a system's overall utility. We

employed contextual bandits with a holistic representation and showed significant improvement with use of metareasoning in our experiments. Future directions include studies of scaling up the mechanisms we have presented to more general systems of interacting modules and the use of different forms of contextual signals and their analyses.

Acknowledgements

Part of this work was done while AM was at Microsoft Research. AM acknowledges the concurrent support by a grant from the Open Philanthropy Project to the Center for Human-Compatible AI and NSF grant CAREER IIS-1452099.

References

Bietti, A.; Agarwal, A.; and Langford, J. 2018. A contextual bandit bake-off. *arXiv preprint arXiv:1802.04064*.

Bradley, D. M. 2010. Learning in modular systems. Technical report, Carnegie-Mellon University, Pittsburgh PA.

Delimitrou, C., and Kozyrakis, C. 2013. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *ACM SIGPLAN Notices*, volume 48, 77–88. ACM.

Delimitrou, C., and Kozyrakis, C. 2014. Quasar: resource-efficient and qos-aware cluster management. In *ACM SIGARCH Computer Architecture News*, volume 42, 127–144. ACM.

Demirci, M. 2015. A survey of machine learning applications for energy-efficient resource management in cloud computing environments. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 1185–1190. IEEE.

Fedorova, A.; Vengerov, D.; and Doucette, D. 2007. Operating system scheduling on heterogeneous core systems. In *Proceedings of the Workshop on Operating System Support for Heterogeneous Multicore Architectures*.

Gao, J. 2014. Machine learning applications for data center optimization.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 1856–1865.

Hanus, D. 2013. *Smart scheduling: optimizing Tilera's process scheduling via reinforcement learning*. Ph.D. Dissertation, Massachusetts Institute of Technology.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hill, D. N.; Nassif, H.; Liu, Y.; Iyer, A.; and Vishwanathan, S. 2017. An efficient bandit algorithm for realtime multivariate optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, 1813–1821.

Hinton, G.; Srivastava, N.; and Swersky, K. 2012. Neural networks for machine learning, lecture 6a: Overview of mini-batch gradient descent. URL:https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Horvitz, E., and Lengyel, J. 1997. Perception, attention, and resources: A decision-theoretic approach to graphics rendering. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, 238–249. Morgan Kaufmann Publishers Inc.

Horvitz, E. 2001. Principles and applications of continual computation. *Artificial Intelligence* 126(1-2):159–196.

III, H. D.; Langford, J.; Mineiro, P.; and Sharaf, A. 2018. Residual loss prediction: Reinforcement learning with no incremental feedback. In *International Conference on Learning Representations*.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4:237–285.

Konda, V. R., and Tsitsiklis, J. N. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*, 1008–1014.

Li, Y. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*, 740–755. Springer.

Lorido-Botran, T.; Miguel-Alonso, J.; and Lozano, J. A. 2014. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing* 12(4):559–592.

Mao, H.; Alizadeh, M.; Menache, I.; and Kandula, S. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 50–56. ACM.

Memeti, S.; Pllana, S.; Binotto, A.; Kołodziej, J.; and Brandic, I. 2018. Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review. *Computing* 1–44.

Mirhoseini, A.; Pham, H.; Le, Q. V.; Steiner, B.; Larsen, R.; Zhou, Y.; Kumar, N.; Norouzi, M.; Bengio, S.; and Dean, J. 2017. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2430–2439. JMLR. org.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *NIPS Autodiff Workshop*.

Raman, K.; Swaminathan, A.; Gehrke, J.; and Joachims, T. 2013. Beyond myopic inference in big data pipelines. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 86–94.

Singh, S.; Jaakkola, T.; Littman, M. L.; and Szepesvári, C. 2000. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning* 38(3):287–308.

Xu, C.-Z.; Rao, J.; and Bu, X. 2012. Url: A unified reinforcement learning approach for autonomic cloud management. *Journal of Parallel and Distributed Computing* 72(2):95–105.

A Survey of Machine Learning Applications for Energy-Efficient Resource Management in Cloud Computing Environments

Conference Paper · December 2015

DOI: 10.1109/ICMLA.2015.205

2015 IEEE 14th International Conference on Machine Learning and Applications

A Survey of Machine Learning Applications for Energy-Efficient Resource Management in Cloud Computing Environments

Mehmet Demirci
Department of Computer Engineering
Gazi University
Ankara, Turkey
Email: mdemirci@gazi.edu.tr

***Abstract*—Ensuring energy efficiency in data centers is a crucial objective in modern cloud computing because it reduces operating costs and complies with the goals of green computing. Researchers strive to develop optimal policies for resource management in the cloud, which has many components such as virtual machine placement, task scheduling, workload consolidation, and so on. Machine learning has a major role to play in these efforts. In this paper, we provide a detailed survey of recent works in the literature which have employed machine learning (ML) to offer solutions for energy efficiency in cloud computing environments. We also present a comparative classification of the proposed methods. Furthermore, we enrich this survey by studying non-ML proposals to energy conservation in data centers, and also how ML has been applied towards other objectives in the cloud.**

***Index Terms*—Energy Efficiency, Resource Management, Data Centers, Cloud Computing, Machine Learning**

I. Introduction

Cloud computing has matured and established itself as an indispensable part of information technology in recent years. As a computational paradigm enabling economies of scale, when configured and utilized effectively, cloud computing offers substantial benefits in terms of computation power while reducing costs and saving energy. Large data centers are the places where the idea of cloud computing comes to life. Through virtualization technology, it becomes possible for many users to share data center resources and services, thus avoiding having to set up their own infrastructure to do things which can be done on the cloud.

Resource management in the cloud (or in data centers) signifies the decisions governing the allocation of processing power (or CPU time), memory, storage, network bandwidth etc. among different services requesting them. Successful resource management may mean different things for cloud providers depending on their priorities and objectives, which may involve balancing the load, maximizing revenue, minimizing response time or the usage of server resources, bandwidth, electricity etc.

According to previous research [1], approximately 15% of data center costs are electrical utility costs. Reducing these costs may be possible through innovations in hardware technology, as well as efficient operation and management of data center networks. Intelligent resource management with the objective of maximizing energy efficiency can help to increase savings. Therefore, developing optimal or near-optimal strategies for minimizing energy usage is a worthwhile endeavor.

Energy efficiency in cloud computing has been a popular research topic over the last decade. A number of works, some of which will be discussed in Section II, have proposed different kinds of optimization solutions to the problem of minimizing energy costs in cloud computing environments. There are also many examples of applying machine learning techniques to resource provision and management in the cloud with various objectives. This survey paper focuses on the intersection of the two aforementioned groups of works: We provide a survey of machine learning-based proposals to reducing energy usage in data centers. Our goal is to shed light on this branch of energy efficiency research, present the state of the art to machine learning researchers, and assist them in developing novel approaches that can produce successful solutions.

The rest of the paper is organized as follows. In Section II, we take a look at cloud resource management from two directions: optimizations for energy-efficient resource management, and machine learning solutions targeting objectives besides energy efficiency. In Section III, we converge on the main topic in this work: how machine learning has been used by researchers to improve energy efficiency in the cloud. Section IV includes a comparative discussion of the research reviewed in this paper. Finally, Section V summarizes the paper and gives possible directions for future work in this area.

II. Energy Efficiency and Machine Learning in the Cloud

This section provides context for the main topic of the paper by first introducing other methods besides machine learning for energy efficiency in the cloud, and then reviewing some machine learning applications to general cloud resource man-

This work was supported in part by TUBITAK Grant 114C093.

978-1-5090-0287-0/15 $31.00 © 2015 IEEE
DOI 10.1109/ICMLA.2015.205

agement in the literature, where the objective is not directly energy-related.

A. Optimizing for Energy Efficiency

Technologies such as parallel programming and virtualization enabled multiple users and applications sharing computing resources, i.e., multi-tenancy. Server clusters, although not at the scale of modern data centers, are environments where multi-tenancy should be supported. Heath et al. [2] focused on the problem of distributing client requests to the servers in a heterogeneous cluster so that a good tradeoff between throughput and energy savings will be found. They designed a cluster with the ability to configure itself to optimize for a variety of metrics (or a combination thereof), among which energy consumption was featured and reducing it was chosen as a main objective. They developed analytical models for request distributions and resource utilization, and then used simulated annealing to find a request distribution that minimizes power-to-throughput ratio. Their method offered over 40% savings in energy consumption.

Chen et al. [3] designed algorithms to minimize the number of running servers via dynamic provisioning and distribute the load to these servers in such a way that saves up to 30% energy. Urgaonkar et al. [4] devised an online control algorithm to perform admission control and resource management in a data center using Lyapunov Optimization. They formulated stochastic optimization problems whose solutions maximize a joint utility combining application throughput and the amount of energy saved.

Beloglazov et al. [5], [6] proposed policies for energy-efficient cloud resource allocation and scheduling algorithms to meet quality of service demands while maintaining low power usage. They made use of dynamic virtual machine (VM) allocation and live migration (movement from one physical node to another) to achieve reduced energy consumption. The algorithm for VM placement is a Modified Best Fit Decreasing algorithm, and the algorithm for deciding which VMs to move looks to minimize the number of VM migrations needed to lower CPU utilization below a threshold at all hosts.

Gandhi et al. [7] employed a combination of predictive and reactive resource provisioning to satisfy service-level agreements (SLA) while saving up to 35% energy. The proposed system analyzes long-term demand histories to establish a base workload, feeds this base workload to a predictive controller that allocates just enough resources to meet SLA requirements, and augments it with a simple reactive controller to handle cases where actual demand is higher than the predicted demand.

Van et al. [8] developed dynamic VM provisioning and placement managers that strive to ensure SLA compliance while reducing energy consumption, and place VMs on the minimum number of physical machines via live migration. Consolidating tasks in a data center can reduce the number of servers needed, yet released resources may still consume energy in the idle state. Lee and Zomaya [9] considered idle power draw in addition to active energy consumption and developed heuristics to minimize total energy consumption.

Xu and Fortes [10] used a genetic algorithm supported with fuzzy multi-objective optimization to simultaneously minimize the amount of wasted resources, energy consumption and thermal dissipation costs.

Shen et al. [11] presented CloudScale, a prediction-based online system for adaptive cloud resource allocation. Their resource demand predictor uses a fast Fourier transform to identify a signature that can be used to estimate future demands. If a signature is not found, the predictor then employs a discrete-time Markov chain. Furthermore, CloudScale uses preemptive VM migration to avoid conflicts, along with dynamic voltage and frequency scaling to save energy. The result is that CloudScale can save 8-10% total energy consumption.

Heller et al. [12] introduced ElasticTree, a power manager with a focus on the data center network elements (links and switches). ElasticTree monitors traffic conditions in the data center, and simply turns off the switches and links if they are not needed. The authors used a combination of linear programming, greedy bin-packing, and topology-aware heuristic approaches to achieve up to 50% reduction in network energy usage.

Berl et al. [13] provided a more comprehensive survey of energy-efficient cloud computing solutions.

B. Machine Learning for the Cloud

There exist many examples of machine learning based solutions to various resource management problems in the cloud. In this subsection, we will discuss several such solutions with varying objectives. It is important to note that although these solutions were not explicitly designed for energy efficiency, some of them may reduce energy consumption as a side effect.

Liao et al. [14] used machine learning to find the best configuration for memory prefetchers. They employed a variety of algorithms including nearest neighbor, naive Bayes, C4.5 decision tree, Ripper, support vector machines, logistic regression, multi-layer perceptron, and radial basis function.

Bodik et al. [15] applied statistical machine learning models, such as linear and LOESS regression, to optimal control for data centers.

Predicting how much time and resources will be spent by applications is necessary to be able to schedule jobs efficiently. Matsunaga and Fortes [16] viewed this as a supervised machine learning problem and extended the Predicting Query Runtime algorithm [17] to the regression problem, calling their modified method PQR2. Using Weka, they compared PQR2 to a group of machine learning algorithms including k-nearest neighbors, linear regression, decision tree, radial basis function, and support vector machine. They showed that PQR2 offers the best accuracy among these algorithms. In another work on web applications, Jiang et al. [18] combined machine learning (linear regression) with time series analysis to predict the number of requests and decide whether to increase or decrease the number of active VMs.

Islam et al. [19] utilized error correction neural network and linear regression along with sliding window to predict resource usage patterns in the cloud. They evaluated prediction accuracy using data generated by running TPC-W [20] on Amazon EC2 and demonstrated the effectiveness of their approach. They also tried different sliding window sizes and concluded that neural network with optimal sliding window size performs better than linear regression. Gong et al. [21] developed another system called PRESS that predicts cloud resource demands to perform elastic resource scaling using statistical machine learning methods. Bankole and Ajila [22] evaluated their resource demand prediction models built using support vector machine, neural network and linear regression. They concluded that SVM attains the best results in terms of response time and throughput.

Maximizing profits is a crucial goal for cloud providers. To this end, Xiong et al. [23] proposed SmartSLA, a resource management system that consolidates multiple VMs into a single physical machine to reduce costs while complying with tenant SLAs. Machine learning comes into play in learning a model describing how different resource allocations to clients correspond to potential profits. After this, the module responsible for resource allocation uses the learned model to adjust allocations and maximize profits. The authors realized that simple linear regression was unsatisfactory for their objectives, so they turned to the regression tree model and added a boosting approach called additive regression to decrease the prediction error.

Several researchers applied reinforcement learning to resource allocation and management in the cloud. Xu et al. [24] took a unified reinforcement learning approach to determine the optimal configurations for VMs in a cloud computing environment. Dutreilh et al. [25] integrated reinforcement learning solutions in an automated controller for the cloud. Their workflow presented three key components for tuning the model: Q-function initialization, convergence speedups, and performance model change detection.

Barrett et al. [26] proposed a parallel Q-learning approach to reduce the amount of time required to zero in on the optimal resource scaling policy during online learning. The authors stated that their approach was the first application of parallelized reinforcement learning to improving convergence times. Their evaluation which used multiple learning agents (varying from 2 to 10) showed that parallelization was able to provide meaningful reduction in convergence times. Rao et al. [27] viewed cloud resource management as a task suitable for distributed learning, and utilized reinforcement learning to develop a mechanism where each VM acts as an autonomous agent in the learning process.

Kundu et al. [28] applied refinements to artificial neural network and support vector machine algorithms to model the relationship between application performance and the resources allocated to the VM hosting the application. They argue that their results are significantly better than those provided by non-refined machine learning methods or regression approaches. Huang et al. [29] employ support vector regression technique in conjunction with a genetic algorithm to reduce application service response times in the cloud.

Virtual network (VN) embedding [30] is closely related to cloud resource management. In data centers, the substrate network in the data center network, and incoming VN requests must be met with efficient resource allocations. Mijumbi et al. [31] design a reinforcement learning algorithm to perform substrate resource management. Their main objective is increasing VN acceptance ratio, i.e, the fraction of incoming VN requests that are successfully answered.

III. Machine Learning Solutions to Energy-Efficient Cloud Resource Management

In this section, we review a number of machine learning-based proposals for enabling energy efficiency in cloud computing environments.

There are many factors influencing energy consumption in data centers such as power distribution, the heat produced by data center operations and resulting cooling costs, and the management of computing load [32]. Most of the current solutions offered for energy efficiency in data centers focus on optimally distributing the computing load so that the minimum number of machines will be activated to satisfy application demands.

Vasic et al. [33] proposed DejaVu, a resource management system for the cloud that learns from the results of previous resource allocations. The learning phase of DejaVu takes about a week of service use when workloads and their corresponding resource allocations are identified. In actual use, DejaVu automatically classifies each a workload to check if it matches a previously encountered workload. Depending on the classification result, it either reuses the previous allocation, or orders the service to reconfigure itself. The authors argued that their efficient mechanism for adapting to new workloads would result in lowered energy costs as it allows load consolidation.

Demand forecasting is a key problem in data center management, and good forecasting techniques can lead to optimal allocation strategies that minimize energy consumption. Prevost et al. [34] utilized neural network and auto-regressive linear prediction to forecast future demand profiles. Performance results of a multi-layer perceptron model and a linear auto-regressive predictor were compared. The authors concluded that the linear predictor was able to produce a more accurate model. Similarly, Duy et al. [35] turn to a neural network based predictor for load forecasting in the cloud. They used the results of the forecast to turn off unused servers and conserve energy. Using historical demand data to train their system, the authors were able to demonstrate that their scheduling algorithm was able to save over 40% energy.

Dabbagh et al. [36] developed another framework for predicting future virtual machine requests and associated resource requirements. This framework uses this predictor to reduce energy consumption by putting unneeded machines into sleep mode. The techniques used are k-means for clustering, and stochastic Wiener filter for workload prediction. Using Google

TABLE I
CATEGORIZATION OF ML PROPOSALS TO ENERGY EFFICIENCY IN THE CLOUD

Year	Authors	Learning Model	Objective (Energy saving method)
2007	Tesauro et al. [40]	Hybrid (Reinforcement + Supervised)	Adjusting CPU frequency
2007	Tesauro et al. [41]	Hybrid (Reinforcement + Supervised)	Power-aware server allocation
2010	Duy et al. [35]	Supervised	Intelligent scheduling to turn off unused servers
2010-11	Berral et al. [38], [39]	Supervised	Power-aware task scheduling and consolidation
2011	Prevost et al. [34]	Supervised	Load prediction leading to optimal resource allocation
2011	Chen et al. [32]	Reinforcement	Spatially-aware load placement to reduce cooling costs
2012	Vasic et al. [33]	Supervised	Workload classification leading to load consolidation
2014	Dabbagh et al. [36]	Unsupervised	Request forecasting to put unused machines to sleep

traces collected over 29 days, the authors showed that their framework achieved near-optimal energy efficiency.

A few solutions go beyond this simple objective of activating the minimum number of physical machines. Chen et al. [32] considered power distribution and cooling in deciding where to place the computing load. They used a model-based reinforcement learning approach to learn the thermal distribution resulting from different workload placements, and then predict the thermal distribution of incoming workloads under various placement alternatives in order to pick the optimal one. The authors called their method spatially-aware workload management (SpAWM), and deployed it using VMWare's ESXServer virtualization infrastructure. They used the Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library (PyBrain) [37] to implement their online algorithm, which combines neural networks and reinforcement learning Evaluation results showed a $2 - 3°\mathrm{C}$ decrease in temperature, which led to saving $13\% - 18\%$ cooling energy.

Machine learning methods can be used as complementary tools in building holistic solutions to energy efficiency in the cloud. Berral et al. [38], [39] employed supervised machine learning methods to predict resource consumption by different tasks and SLA-related metrics such as response time for a given workload, and integrated their predictor in a system that performs power-aware task scheduling and consolidation. The algorithms they used were linear regression to predict CPU usage at each host, and a more complex machine learning algorithm called M5P to predict power consumption. Experiments were carried out using real workloads, and demonstrated that their methods offered substantial power savings while slightly decreasing performance.

Certain solutions in the literature apply machine learning at the level of individual servers in the data center. Tesauro et al. [40] presented a reinforcement learning approach to simultaneously manage performance and power consumption by intelligently adjusting CPU frequency online. Their approach is tunable in that it uses a simple objective function that subtracts power consumption multiplied by a modifiable coefficient from a performance-based utility. They employed the Sarsa(0) update rule, and trained a multilayer perceptron neural network. Their paper detailed some specific innovations in applying reinforcement learning and justified their use through experimental evaluation.

In another work, Tesauro et al. [41] proposed a hybrid reinforcement learning approach supported with neural networks, this time for enabling intelligent server allocation decisions. By combining the flexibility of reinforcement learning, which does not require explicit models, and the ability of model-based policies to quickly reach high performance, they were able to attain successful results while avoiding the potential performance problems associated with online reinforcement learning.

IV. DISCUSSION

Table I presents a categorization of the proposals summarized in the previous section. We observe that supervised methods are widely used, and among these artificial neural network and linear regression models are the most common. A couple of solutions [40], [41] augment reinforcement learning with supervised techniques (such as multi-layer perceptron) to try and get the best of both worlds.

Most of the proposals have employed machine learning in predicting future workloads in the data center. Once this prediction is made, a variety of algorithms can be used to maximize energy conservation. A straightforward goal is to minimize the number of servers needed and simply turn off unused machines or put them in sleep mode. A more creative approach to resource management involves factoring thermal distribution into workload placement decisions. Moving the computing load in such a way that the resulting thermal distribution will necessitate the minimum amount of cooling energy was proved to offer meaningful power conservation [32].

An interesting area of research that is lacking in the literature is the interaction between seemingly conflicting cloud resource management objectives, such as energy efficiency and fault tolerance. Energy conservation is usually enabled through server consolidation where computing load is placed on the lowest number of servers possible. However, in case of a failure, this policy could lead to weakened resilience and fault tolerance. Although redundancy is typically not a friend of energy efficiency, selective server and virtual machine replication has the potential to preserve energy efficiency while maintaining resilient operation in the data center. Machine learning could well be the key in finding intelligent ways to determine the scope of such replication, that is, selecting where and when to replicate.

V. Concluding Remarks

In this paper, we have provided an extensive review of machine learning applications towards energy efficient management of cloud data centers. The most common use of machine learning in this context is for predicting future resource demands. Accurate estimation of these demands can allow cloud providers to develop intelligent resource management policies which rely on task scheduling and consolidation to turn on the minimum number of machines in the data center, thus conserving energy. In addition, researchers have found ways to reduce cooling costs by paying attention to the thermal distribution resulting from different workload placements. On the scale of individual machines, machine learning has been used to save energy by optimally configuring CPU frequency.

Many works in the literature combine energy efficiency with a performance objective such as SLA satisfaction, or a monetary objective such as maximized profits, and seek a joint optimization or study the tradeoffs. One suggestion for potential future work is pairing energy efficiency with a different sort of objective such as reliability, survivability, fault tolerance, or security. Another avenue for future work in this area could be energy-efficient configuration of both virtual network and data center network topologies using machine learning techniques.

References

[1] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[2] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr, and R. Bianchini, "Energy conservation in heterogeneous server clusters," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2005, pp. 186–195.

[3] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services." in *NSDI*, vol. 8, 2008, pp. 337–350.

[4] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*. IEEE, 2010, pp. 479–486.

[5] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 826–831.

[6] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[7] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah, "Minimizing data center sla violations and power consumption via hybrid resource provisioning," in *Green Computing Conference and Workshops (IGCC), 2011 International*. IEEE, 2011, pp. 1–8.

[8] H. N. Van, F. D. Tran, and J.-M. Menaud, "Performance and power management for cloud infrastructures," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 329–336.

[9] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[10] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*. IEEE, 2010, pp. 179–188.

[11] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 5.

[12] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks." in *NSDI*, vol. 10, 2010, pp. 249–264.

[13] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis, "Energy-efficient cloud computing," *The computer journal*, vol. 53, no. 7, pp. 1045–1051, 2010.

[14] S.-w. Liao, T.-H. Hung, D. Nguyen, C. Chou, C. Tu, and H. Zhou, "Machine learning-based prefetch optimization for data center applications," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, p. 56.

[15] P. Bodık, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters," in *Proceedings of the 2009 conference on Hot topics in cloud computing*, 2009, pp. 12–12.

[16] A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 495–504.

[17] C. Gupta, A. Mehta, and U. Dayal, "Pqr: Predicting query execution times for autonomous workload management," in *Autonomic Computing, 2008. ICAC'08. International Conference on*. IEEE, 2008, pp. 13–22.

[18] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 58–65.

[19] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, 2012.

[20] "TPC-W," http://www.tpc.org/tpcw/, [Online; accessed 26-August-2015].

[21] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *Network and Service Management (CNSM), 2010 International Conference on*. IEEE, 2010, pp. 9–16.

[22] A. A. Bankole and S. A. Ajila, "Predicting cloud resource provisioning using machine learning techniques," in *Electrical and Computer Engineering (CCECE), 2013 26th Annual IEEE Canadian Conference on*. IEEE, 2013, pp. 1–4.

[23] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacigümüş, "Intelligent management of virtualized resources for database systems in cloud environment," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 2011, pp. 87–98.

[24] C.-Z. Xu, J. Rao, and X. Bu, "Url: A unified reinforcement learning approach for autonomic cloud management," *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 95–105, 2012.

[25] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow," in *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, 2011, pp. 67–74.

[26] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[27] J. Rao, X. Bu, C.-Z. Xu, and K. Wang, "A distributed self-learning approach for elastic provisioning of virtualized cloud resources," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*. IEEE, 2011, pp. 45–54.

[28] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling virtualized applications using machine learning techniques," in *ACM SIGPLAN Notices*, vol. 47, no. 7. ACM, 2012, pp. 3–14.

[29] C.-J. Huang, Y.-W. Wang, C.-T. Guan, H.-M. Chen, and J.-J. Jian, "Applications of machine learning to resource management in cloud computing," *International Journal of Modeling and Optimization*, vol. 3, no. 2, p. 148, 2013.

[30] A. Fischer, J. F. Botero, M. Till Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 4, pp. 1888–1906, 2013.

[31] R. Mijumbi, J.-L. Gorricho, J. Serrat, M. Claeys, F. De Turck, and S. Latré, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[32] H. Chen, M. Kesavan, K. Schwan, A. Gavrilovska, P. Kumar, and Y. Joshi, "Spatially-aware optimization of energy consumption in con-

solidated data center systems," in *ASME 2011 Pacific Rim Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Systems*. American Society of Mechanical Engineers, 2011, pp. 461–470.

[33] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini, "Dejavu: accelerating resource allocation in virtualized environments," in *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1. ACM, 2012, pp. 423–436.

[34] J. J. Prevost, K. Nagothu, B. Kelley, and M. Jamshidi, "Prediction of cloud data center networks loads using stochastic and neural models," in *System of Systems Engineering (SoSE), 2011 6th International Conference on*. IEEE, 2011, pp. 276–281.

[35] T. V. T. Duy, Y. Sato, and Y. Inoguchi, "Performance evaluation of a green scheduling algorithm for energy savings in cloud computing," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–8.

[36] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-efficient cloud resource management," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. IEEE, 2014, pp. 386–391.

[37] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber, "Pybrain," *The Journal of Machine Learning Research*, vol. 11, pp. 743–746, 2010.

[38] J. L. Berral, Í. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavaldà, and J. Torres, "Towards energy-aware scheduling in data centers using machine learning," in *Proceedings of the 1st International Conference on energy-Efficient Computing and Networking*. ACM, 2010, pp. 215–224.

[39] J. L. Berral, R. Gavalda, and J. Torres, "Adaptive scheduling on power-aware managed data-centers using machine learning," in *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*. IEEE Computer Society, 2011, pp. 66–73.

[40] G. Tesauro, R. Das, H. Chan, J. Kephart, D. Levine, F. Rawson, and C. Lefurgy, "Managing power consumption and performance of computing systems using reinforcement learning," in *Advances in Neural Information Processing Systems*, 2007, pp. 1497–1504.

[41] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "On the use of hybrid reinforcement learning for autonomic resource allocation," *Cluster Computing*, vol. 10, no. 3, pp. 287–299, 2007.

Future Generation Computer Systems 29 (2013) 451–459

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Graph partitioning algorithms for optimizing software deployment in mobile cloud computing

Tim Verbelen *, Tim Stevens, Filip De Turck, Bart Dhoedt

Ghent University – IBBT, Department of Information Technology, Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium

ARTICLE INFO

Article history:
Received 8 September 2010
Received in revised form
19 June 2012
Accepted 14 July 2012
Available online 20 July 2012

Keywords:
Distributed systems
Graph algorithms
Deployment optimization
Cloud computing
Mobile computing

ABSTRACT

As cloud computing is gaining popularity, an important question is how to optimally deploy software applications on the offered infrastructure in the cloud. Especially in the context of mobile computing where software components could be offloaded from the mobile device to the cloud, it is important to optimize the deployment, by minimizing the network usage. Therefore we have designed and evaluated graph partitioning algorithms that allocate software components to machines in the cloud while minimizing the required bandwidth. Contrary to the traditional graph partitioning problem our algorithms are not restricted to balanced partitions and take into account infrastructure heterogenity. To benchmark our algorithms we evaluated their performance and found they produce 10%–40% smaller graph cut sizes than METIS 4.0 for typical mobile computing scenarios.

1. Introduction

Nowadays, the emergence of cloud computing is leading to a new paradigm of utility computing [1], where computing power is offered on an on-demand basis. Users are able to access applications, storage and processing over the Internet, via services offered by cloud providers on a pay-as-you-use scheme. The advantages for the end users are reduced cost, higher scalability and improved performance in comparison to maintaining their own private computer systems, dimensioned for peak load conditions. Moreover the elasticity of the cloud reduces the risks of overprovisioning (underutilization) or underprovisioning (saturation) [2].

The usage of the cloud is not only beneficial for web-based applications, but can also be used for other applications composed of many service components following the service-oriented programming paradigm. Some of these service components may have high needs regarding CPU power or memory consumption, and should therefore be executed on dedicated server machines in the cloud rather than on a regular desktop PC or mobile terminal, for example recognition components in an object recognition or speech to text application.

The adoption of the cloud paradigm poses the problem where to deploy software components, given the many options in terms of available hardware nodes in even moderate scale data centers. This deployment optimization is important to both the cloud user and the cloud provider in order to reduce costs. All components need to be deployed on a machine with sufficient CPU power while the communication overhead between different machines is preferably minimized as this introduces extra latency and network load. This problem can be modelled as a graph partitioning problem where a weighted graph of software components has to be partitioned in a number of parts representing the available machines. Moreover the optimal deployment can change over time, and thus in order to realise an optimal deployment a fast algorithm is desired.

An important scenario in this respect is cloud overspilling. In this scenario, a company offloads work from its own private infrastructure to a public cloud infrastructure on peak moments, as shown in Fig. 1. This enables the company to dimension its infrastructure for the average workload instead of the peak workload, reducing the underutilization and the cost of the private infrastructure. This situation is typical in digital document processing where one faces strict month-end or year-end deadlines, and thousands of batches of documents are to be processed. Typically, document processing systems support workflows consisting of a few tens of components (including content ingest, reformatting, layouting, merging, versioning, logging, output generation and printing components). As many of these components come in different versions, and potentially need to be instantiated for each customer separately, the number of components in such a scenario quickly amounts to a few hundreds.

A second use case is situated in the area of integrated simulation tools for engineering purposes [3]. These integrated tools typically involve multi-physics simulation (e.g. structural analysis, acoustic

* Corresponding author.
E-mail address: tim.verbelen@intec.ugent.be (T. Verbelen).

0167-739X/$ – see front matter
doi:10.1016/j.future.2012.07.003

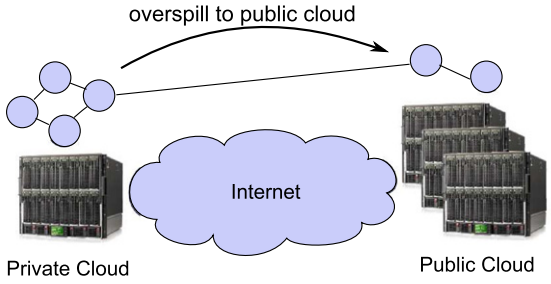


Fig. 1. Work is offloaded from private infrastructure to a public cloud on peak moments, reducing underutilization (and the cost) of the private infrastructure.

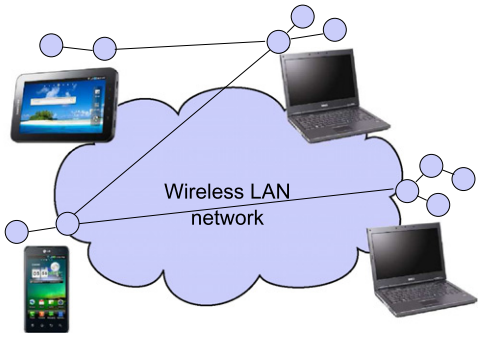


Fig. 2. A cloudlet consisting of four devices connected by a wireless network sharing their resources. Application components are distributed among all devices in the cloudlet, in order to enhance the user experience of all users.

simulation and engine dynamics simulation in the case of designing a new automobile engine), and the number of simulation tools involved can easily amount to 10–20 for small engineering projects to a few 100 individually deployable components for a realistic engineering project. In such engineering endeavours, often parameter sweeps are executed to optimize the design (or to assess the sensitivity of the resulting design performance w.r.t. these parameters), necessitating multiple instances of these simulation components running concurrently, in order to arrive at realistic design times. Again, we end up with a component graph containing a few 100–1000 components.

This overspilling problem also arises in the context of mobile computing, where the cloud can be used to enhance the capabilities of a mobile device. Due to the restricted CPU power of mobile devices, the idea is to offload parts of the application at runtime to a cloud infrastructure [4]. The question then is which parts to offload and to deploy on which machines in the cloud – and how many – in order to spread the load while keeping the needed bandwidth low. In this case, the complexity of the partitioning problem depends on the granularity of the offloading, as offloading can be done on component [4], Java class [5] or method [6] level, resulting in graphs of tens, hundreds or thousands of components.

A special case of deployment optimization occurs when multiple mobile devices connected via a wireless network share their resources in order to enhance the user experience of all users, in a so called "cloudlet" [7], as shown in Fig. 2. This is the case when no internet uplink is available for offloading to the cloud, or when cloud offloading is not beneficial due to a high WAN latency. Because the bandwidth is a scarce resource and shared between all devices in the wireless LAN, a global optimization is needed taking into account all application components of all devices.

As an use case, we mention a mobile augmented reality application. When casting such an application into a component framework, the number of independently deployable components amounts to 5–10 [7], with different components for tracking camera movements, building a 3D map of the environment, recognizing objects, detecting collisions between objects, rendering a 3D overlay, etc. Other applications, such as 3D games, are reported to consist of 10–20 components [8]. Assuming a few tens of users connected to the same cloudlet and hence sharing computing and network resources, the number of components easily exceeds 100. As these users are connected through heterogeneous devices (a mix of low-end and high-end devices), an optimal deployment guaranteeing a minimal quality of experience for all users should be aimed for.

In this paper we present algorithms to partition a software application, composed of a number of components, on a number of interconnected machines in the cloud with different capacities while minimizing the communication cost between the components. In Section 2 related work regarding graph partitioning and task allocation on the grid is discussed. Section 3 more formally describes our problem and in Section 4 algorithms are proposed that solve the problem. In Section 5 the different algorithms are evaluated and compared regarding solution quality and execution time, and the influence of different parameters is discussed. In view of the use cases mentioned in this introduction, we focus on graphs containing 100–1000 components for this evaluation. We also compare our solutions to METIS 4.0 for partitioning graphs in k balanced partitions and show the applicability of our algorithms in the mobile offloading scenario. Finally Section 6 concludes this paper.

2. Related work

2.1. Graph partitioning

Graph partitioning is a fundamental problem in many domains of computer science, such as VLSI design [9], parallel processing [10] and load balancing [11]. The graph partitioning problem tackles the problem of dividing a graph in k equal sets while minimizing the edges between the sets. When $k = 2$ this is also referred to as the min-cut bipartitioning problem. Finding a good solution for this problem is known to be NP-Hard [12]. In the following we give a brief overview of the state-of-the-art and recent advances in graph partitioning. For a more detailed survey of graph partitioning techniques we refer to [13,14].

A first class of algorithms are the so called move-based approaches, which try to iteratively improve the partition by vertex moves or swaps between the parts, such as the Kernighan–Lin (KL) algorithm [15]. By choosing moves that introduce a cost reduction of the graph cut this algorithm converges to a local optimum. Fiduccia and Mattheyses introduced a number of optimizations to the KL algorithm which led to a linear time algorithm for graph partitioning [16]. These move-based algorithms can be combined with stochastic methods such as simulated annealing [17], particle swarm optimization [18] or ant colony optimization [19] in order to escape from local optima. The biggest disadvantage of iterative improvement methods is that their performance deteriorates as the graphs get larger.

In order to partition large graphs the multilevel approach has become widely adopted [20,21,17–19,22]. The main idea is to iteratively coarsen the initial graph by merging vertices according to a matching until a small graph with a similar structure remains. This graph can then be partitioned with a spectral method [23,20] or a greedy graph growing algorithm [24]. Next the graph is again iteratively uncoarsened and a local improvement heuristic such as the KL algorithm is applied at each level. The multilevel scheme is also used in state-of-the-art graph partitioning libraries such as METIS [24], SCOTCH [25] and JOSTLE [26].

Recent work in graph partitioning explores methods based on diffusion [11] or maximum flow [27]. Also the combination of known techniques can result in new heuristics. Chardaire et al. use a PROBE (Population Reinforced Optimization Based Exploration) heuristic, combining greedy algorithms, genetic algorithms

and KL refinement [28]. Loureiro and Amaral propose a greedy graph growing heuristic combined with a local refinement algorithm [29]. Martin uses a genetic algorithm improved with spectral methods and KL refinement [30].

All these methods partition the graph in a predefined number of parts of equal sizes. In the context of cloud computing, not all machines have equal capacity and also the number of machines that has to be used is not predefined, thus these algorithms cannot be readily used. Therefore, our algorithms take a number of possible machines with different capacities as input with the only constraints that no machine can exceed its maximum capacity and every component has to be deployed on exactly one machine. However, we can still use the ideas from the original graph partitioning problem to calculate a good deployment.

2.2. Task allocation on the grid

In the context of grid computing the graph partitioning problem is used for the allocation of parallel tasks on heterogenous infrastructure. Many proposed algorithms such as MiniMax [31], VHEM [32], QM [33] PaGrid [34], and MinEX [35] use the multi level paradigm in combination with a refinement algorithm, while others such as PART [36] use simulated annealing.

These algorithms all focus on parallel applications based on mesh models, such as fluid dynamics, where a big task is split up in smaller parallel tasks that are each executed on one of the processors in the grid. The goal is then to execute all these tasks as fast as possible, thus minimizing the execution time of the slowest node. In our problem the execution time metric makes less sense, since we focus on applications on the cloud, where components generate load as long as the end user runs the application. Also the structure of the task graph for the grid is based on the input mesh, while in our case the graph of cloud applications is based on the software components and their interactions, which is structured differently and of a more modest size.

3. Problem statement

Let $G = (V, E)$ be an undirected graph where $V = \{v_1, v_2, \ldots, v_N\}$ is a set of N vertices and E is a set of edges connecting the vertices. The vertices represent units of deployment in a distributed software system and the edges represent communication overhead between those units. Each vertex v_i is assigned a cost w_i that indicates the amount of resources (i.e. CPU power) this component needs. $C = (c_{ij})$ is the adjacency matrix of G, i.e. if there exists an edge between v_i and v_j then c_{ij} equals the weight of this edge, otherwise $c_{ij} = 0$. The edge weights represent the cost of communication (i.e. bandwidth) between different software components.

The infrastructure is also modelled as an undirected graph $S = (K, L)$, where K is a set of available machines and L the links between the different machines. Each machine has a maximum capacity of M_m and each link is assigned a cost to use this link. From this graph a matrix $B = (b_{mn})$ can be deduced where element b_{mn} represents the cost to exchange data between machine m and machine n.

The goal now is to assign each vertex v_i to one of the machines so that the total data exchanged between the machines weighted by B is minimized. More formally, let the decision variables X_{im} represent the graph cut. The value of X_{im} is equal to 1 if component i is deployed on machine m, and 0 otherwise. We want to minimize the sum of the edge weights of the edges between nodes deployed on different machines, thus we introduce variable h_{ij}, which is equal to 1 if components i and j are deployed on different machines, 0 otherwise. Then the objective function to minimize becomes the weight of the graph cut (GC):

$$\mathrm{GC} = \sum_{i,j} h_{ij} \times c_{ij} \times b_{P(i)P(j)}. \tag{1}$$

With the function $P(j)$ returning the machine on which vertex j is deployed. Variables h_{ij} can be expressed in terms of X_{im} as follows:

$$h_{ij} = 1 - \sum_{m} X_{im} \times X_{jm}. \tag{2}$$

Two more constraints are needed to fully describe our problem.

$$\forall k : \sum_{i} w_i \times X_{im} \leq M_m \tag{3}$$

$$\forall i : \sum_{m} X_{im} = 1. \tag{4}$$

Eq. (3) states that the total amount of resources used by nodes on a machine cannot exceed the maximum capacity of that machine. Eq. (4) makes sure that every node is deployed on exactly one machine.

4. Algorithms description

4.1. Integer linear programming (ILP)

The problem defined in Section 3 can be seen as an Integer Linear Programming (ILP) problem, thus an ILP solver (IBM ILOG CPLEX [37]) can be used to determine the optimal solution for this problem. The amount of time and resources needed to solve this problem grows exponentially with the size of the graph, so heuristics are needed to find a good solution faster. We can still use the ILP solution for smaller graphs to benchmark our algorithms that exhibit better scaling behaviour, possibly at the expense of optimality.

4.2. Multilevel graph partitioning (MLKL)

In a first heuristic we use a multilevel refinement strategy as first proposed by Hendrickson and Leland [20]. The idea is to coarsen down the graph by merging connected vertices until a small graph is obtained. Then this graph is partitioned and uncoarsened again, while optimizing the partition in each uncoarsening step. Thus, the partitioning consists of three phases:

Coarsening. The graph G is coarsened in a sequence of smaller graphs $G_1, G_2, \ldots, G_m$ such that $|V_1| > |V_2| > \cdots > |V_m|$.

Initial partitioning. A partition P is computed on the coarsest level of the graph.

Uncoarsening and refinement. Partition P of graph G_m is uncoarsened back through all the intermediate graphs. In each uncoarsening step a refinement algorithm is executed in order to find a better partition.

4.2.1. Coarsening

During coarsening, from graph G_i a graph with fewer vertices G_{i+1} is created by collapsing edges and combining the vertices connected by those edges. When an edge is collapsed the two vertices connected by the edge are reduced into one whose weight is the sum of the weights of both vertices. When both vertices have an edge to a third node, these two edges are collapsed in one edge with a summed edge weight. Every iteration of coarsening, a matching – a set of edges without common vertices – is created and the matched vertices are combined. In order to find a small edge cut, it is beneficial to collapse heavy weighted edges, since these are unlikely to be in the best cut.

Our coarsening algorithm uses the *heavy-edge matching* (HEM) algorithm which is a widely used coarsening scheme proposed by Karypis and Kumar [24]. The vertices of the graph are visited in

a random order and each vertex u is matched with an unmatched neighbour v such that the weight of the edge (u, v) is the maximum over all valid incident edges of u. In order to be able to map the vertices of the coarsest graph on the different machines we added one more constraint to only match two vertices when the sum of their vertex weights is smaller than the size of the smallest part.

4.2.2. Initial partitioning

After coarsening the problem consists of choosing a feasible deployment of the software components on the infrastructure at hand. We assume that there are enough machines and resources available to find such a deployment. This problem reduces to a simple bin packing problem that can be solved with a first-fit algorithm.

The vertices are ordered by descending vertex weight and the machines are ordered by descending maximum capacity. For each vertex the list of machines is iterated and it is assigned to the first one that has enough capacity left, then the capacity of that machine is diminished with the vertex weight of that vertex.

Algorithm 1 Calculate initial partition

```
vertices : OrderedList
machines : OrderedList
for all vertex v_i ∈ vertices do
  for all machine m ∈ machines do
    if w_i ≤ M_m then
      P(i) ← m
      M_m ← M_m − w_i
      break
    end if
  end for
end for
```

4.2.3. Uncoarsening and refinement

In the final step the graph is gradually uncoarsened again and a KL-like algorithm is applied to improve the initial partition found in the previous step. The proposed algorithm is based upon the refinement algorithm of Hendrickson and Leland [20]. The fundamental idea of the algorithm is the concept of the gain associated with moving a vertex to a different part. The gain reflects the net change in cut size that would result from switching a vertex from one machine to another. More formally, the gain introduced by moving vertex i from set m to set n, the corresponding gain g_{mni} can be expressed as

$$g_{mni} = \sum_{j \in V \& P(j)=n} c_{ij} \times b_{mn} - \sum_{j \in V \& P(j)=m} c_{ij} \times b_{mn} + \sum_{j \in V \& P(j)=p, p \neq m, p \neq n} c_{ij} \times (b_{mp} - b_{np}) \quad (5)$$

where again the function $P(j)$ returns the current part of vertex j. The base refinement algorithm is presented in pseudocode as Algorithm 2, and is composed of two loops. Each pass of the outer loop will try moving vertices around to find a better cut. This outer loop ends after a pass that did not improve the best partition so far, implying that the algorithm has reached a local optimum. To avoid the algorithm getting stuck in an infinite loop a vertex can only be moved once during a single pass of the outer loop. The inner loop will iteratively select a vertex to move, i.e. the vertex having the largest gain, subject to some additional rules. Notice that this also could be a move with a negative gain. This gives the algorithm the possibility to escape to some extent from getting stuck in a local optimum. When a vertex is moved the gains of all its neighbours should be updated. This process completes when no more suitable move is found.

Algorithm 2 KL Refinement

```
Bestpartition ← Currentpartition
Compute initial gains
Compute ∀p : Free(p) the amount of free space on part p
repeat
  MovedList ← EmptyList
  repeat
    repeat
      Select vertex move, not in MovedList with highest gain g_mni
      if ∃p : Free(p) < 0 then
        if Free(m) < 0 and Free(n) − w_i > Free(m) then
          MoveFound ← True
        end if
      else
        MoveFound ← True
      end if
    until MoveFound
    if MoveFound then
      Perform move and add move to MovedList
      Update gains of neighbors of moved vertex
      if Currentpartition < Bestpartition then
        Bestpartition ← Currentpartition
        if No possible moves with positive gain then
          Break
        end if
      end if
    end if
  until No more moves possible
until No better partition found
return BestPartition
```

When a better partition is found, the inner loop will only accept moves with positive gain and then start another pass. The rationale for this is that when the algorithm finds a better partition, it will try to descend as deep as possible to get the local optimum. Then, a new iteration is started, and the moves that were in *MovedList* in the previous iteration can again be selected to come closer to the optimum.

Instead of creating balanced partitions, we need to make sure that no machine gets overloaded. A naive rule in this respect would be to forbid vertex moves that would violate the capacity condition of the target. However, this rule could cause to a move to local optima, from which there is no escape (see Fig. 3).

Therefore we do allow a move to a part that exceeds the maximum part weight, provided that there is no other part with excessive weight. If there is a part s with too much weight, then we choose the vertex with the highest gain that moves away from m and reduces the amount of overload. This way situations as in Fig. 3 will also converge to an optimum ($m0 = \{n0, n2\}, m1 = \{n1, n3, n4\}$).

When two possible moves have equal gain, we prefer moves that do not exceed the target parts boundary. If that does not differentiate the possible moves, we prefer those to a part that is not empty. When still no differentiation is made between two moves, ties are broken randomly.

Instead of continuing until all vertices are moved, one can stop the algorithm earlier to save time, for example when the graph cut of the current partition deviates by more than a certain cutoff threshold of the best solution found at that moment. Proper choices for this threshold can reduce the execution time while not

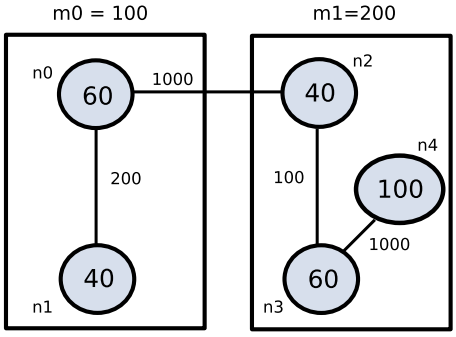


Fig. 3. This partition cannot be optimized when no moves exceeding the target parts maximum boundary are allowed.

sacrificing much solution quality. If a threshold value of 0 is chosen, the algorithm will not accept any moves with negative gain, and the algorithm will behave like a steepest descent algorithm ending in the nearest local minimum.

4.3. Simulated annealing (SA)

A second approach followed for solving the k-partitioning problem is based on simulated annealing (SA), a combinatorial optimization technique introduced by Kirkpatrick et al. [38] and independently by Cerny [39], inspired by the cooling process of metal. To use the SA technique in the context of the k-partitioning problem, we use it as a refinement technique after bin packing, inspired by Johnson et al., who showed the effectiveness of SA for the graph bipartitioning problem [40].

The SA algorithm moves from one solution to a neighbour solution by moving a vertex from one part to another. A move will be accepted with probability $\exp(g/T)$, in which g is the gain of a move, as introduced in Section 4.2, and T a temperature parameter that is lowered gradually over time. The algorithm in pseudocode is shown as Algorithm 3. In order to find the optimum in situations such as described in Fig. 3, vertex i is moved with positive gain to a part that has not enough capacity left with probability $\exp(\frac{Free(p)-w_i}{T})$, where $Free(p)$ is the amount of free space on part p. By making this also dependent on the temperature this assures that towards the end of the algorithm it will converge to a valid solution without causing overoccupied parts.

The performance of SA is very dependent on the choice of the different annealing parameters: the initial temperature T_1, the cooling schedule, the epoch length L, and the stopping condition [40]. Park and Kim later proposed a better set of annealing parameters that led to better results [41], on which we have based our parameter values.

4.3.1. Initial temperature T_1

Kirkpatrick et al. [38] propose a value of T_1 high enough to make the initial probability of accepting transitions to be close to 1. However, a too high initial temperature may lead to an unnecessary long computation time, as pointed out in [40,41], where it is suggested that the initial temperature is chosen such that the fraction of accepted uphill transitions at the initial temperature is equal to a parameter P_1, called the initial acceptance probability. The initial temperature is then calculated as $T_1 = \frac{\overline{\Delta}}{P_1}$, where $\overline{\Delta}$ is calculated with the initial negative gains only.

4.3.2. Cooling function

The cooling function will gradually decrease the temperature until the algorithm reaches its stopping condition. We adopt a simple geometric cooling function, in which the temperature at the kth epoch is given by $T_k = \alpha \times T_{k-1}$, where $\alpha (0 < \alpha < 1)$ is a parameter called the cooling ratio. Johnson et al. [40] showed that other cooling functions such as linear or logarithmic functions do not affect the performance.

Algorithm 3 Simulated Annealing

```
Bestpartition ← Currentpartition
Temperature ← StartTemperatureT1
repeat
  counter ← 0
  repeat
    counter + +
    Calculate gain g to move vertex i to part p
    if g ≥ 0 then
      if Free(p) ≥ w_i then
        Perform move
      else
        Perform move with probability e^(Free(p)−w_i / Temperature)
      end if
    else
      if Free(p) ≥ w_i then
        Perform move with probability e^(g / Temperature)
      end if
    end if
    if Currentpartition < Bestpartition then
      Bestpartition ← Currentpartition
    end if
  until counter ≥ L (epoch length)
  Temperature ← α.Temperature
until stopcondition
return BestPartition
```

Table 1
Simulated annealing parameter values used in our SA refinement algorithm.

Parameter	Value
P_1	0.627
α	0.908
s	50
$F_{\min}$	0.02
I	5

4.3.3. Epoch length L

The epoch length L is the number of moves tried at each temperature level. Johnson et al. state that the epoch length should be chosen proportional with the neighbourhood size. In our case each vertex can be moved to one of the other machines, thus the epoch length becomes $L = s \times N \times (K - 1)$ with N the number of vertices and K the number of machines used. Experiments showed 50 to be a good value for s.

4.3.4. Stopping condition

The stopping condition decides when the algorithm reaches the frozen state and further search for a solution should be stopped. Trivial stopping conditions terminate the algorithm when the epoch count reaches a predefined maximum, or when the temperature drops below a pre-selected final temperature. We use Johnson's stopping rule [40], which increments a counter by one at the end of an epoch and the fraction of accepted moves is less than a predefined limit $F_{\min}$ and the counter is reset to 0 when a better solution is found. The SA is terminated when the counter reaches a predetermined limit I.

The actual values for the different parameters we use are shown in Table 1, which are based on the ones stated in [40,41] and lead to good results for our problem.

Because of the randomness in Simulated Annealing, different runs can lead to different solutions. The quality of the solution can be improved by executing the SA algorithm multiple times and

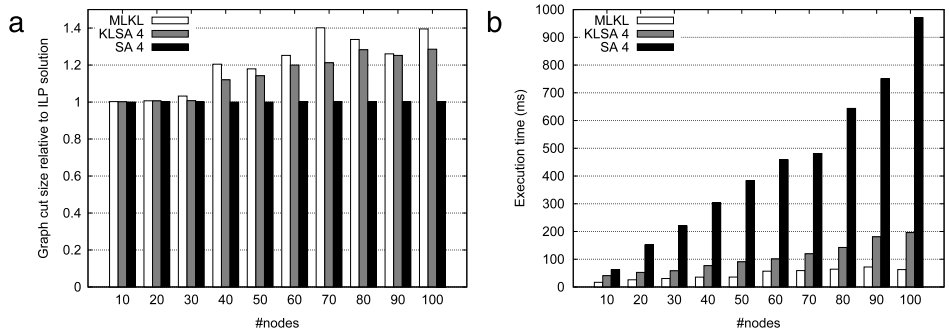


Fig. 4. The left shows the median of the resulting graph cut size of the MLKL, SA and KLSA relative to the optimal ILP solution. Simulated annealing reaches the optimum in more than 50% of the cases, while MLKL performs worse. KLSA is a bit better than MLKL but cannot achieve the quality of SA. The right shows the execution time of the different algorithms. Better solution quality comes at a price: KLSA and SA use respectively 2–10 times more execution time than MLKL.

taking the best solution. This comes of course at the cost of more computation time.

4.4. Hybrid algorithm (KLSA)

Because of the random factor of SA, it will typically explore a larger solution space than KL-based refinement, and hence arrive at better solutions at the cost of more computation time. To combine the best of both approaches, a hybrid approach is pursued in this section. The hybrid algorithm very much resembles the multi-level algorithm: only at the coarsest level the partition is first refined with a few runs of SA. At the coarsest level a better solution can be found with SA while the extra computation cost remains relatively low. In further uncoarsening iterations KL refinement is used again, since this is faster. Moreover, due to the characteristics of coarsening and uncoarsening, the globally optimal solution is expected to be situated in the neighbourhood of the coarsest graph, and thus the local optimum where KL gets stuck in the end is more likely also the global optimum. Again, the SA refinement at the coarsest level can be performed multiple times in order to get a better solution.

5. Evaluation results

To evaluate our algorithms we generated test graphs with different node sizes. Graphs were generated with the Eppstein power law generator [42] and the weights of the nodes and edges are assigned according an exponential distribution with parameter λ equal to 0.1 and 0.005 respectively. Although the actual graph structure of an application will depend on the software design, we chose these graph configurations as most design principles aim to reduce the dependencies between different software components thus resulting in graphs with few nodes with many neighbours. The resulting node weights are restricted between 1 and 100, which could intuitively represent the percentage CPU time needed for this component on a single processor core.

These nodes have to be deployed on machines with randomly generated capacities of 100, 200, . . . , 800, which represents machines equipped with single, dual, . . . , eight core processors. The number of machines is chosen such that their total capacity is about 150% of the total node weight of the graph. For each graph size we generate 100 graphs and machine weights and calculate the best deployment with our different algorithms implemented in Java, which are evaluated according to the resulting graph cut size and processing time. All calculations were performed on Intel Xeon L5420 (2.5 GHz) quad core processors.

First we compare our algorithms with the theoretically optimal ILP solution for small graph sizes to show the effectiveness of our approach. For bigger graphs we compare our three heuristics on solution quality and execution time and look at the influence of the cutoff threshold for KL refinement and the number of executions for SA. To benchmark our algorithms to the state-of-the-art we compare the solution quality of our algorithms with METIS 4.0. Finally we also show the effectiveness of our algorithms in the offloading scenario where a mobile device distributes its load to the cloud.

5.1. Evaluation of MLKL, SA and KLSA

To evaluate how good deployments are found for cloud applications, we assume that the communication links between the different servers in the datacenter are equally weighted, and thus $\forall m, n : b_{mn} = 1$. Since the graphs represent interacting software components, we also assume that the graph size of interest is a few hundred nodes. For small graphs we can compare solutions to the optimal ILP solution, as shown in Fig. 4.

Both the SA and KLSA algorithm use the best solution of 4 parallel executions of simulated annealing. For small graphs simulated annealing performs best as it results in the optimal solution in more than 50% of the cases. The hybrid solution tends to give slightly better results than MLKL. However better solution quality comes at the cost of execution time: KLSA and SA use respectively 2–10 times the execution time of MLKL.

For larger graphs the ILP solver takes too long to find a solution so we compare the algorithms relative to the MLKL algorithm. We will also discuss the influence of different parameters: the cutoff threshold for the KL refinement and the number of parallel executions of the SA refinement.

Fig. 5 on the left shows the solution quality of the MLKL algorithm for values 0, 100, 500 and 1500 for the cutoff threshold. KL refinement is stopped when the graph cut of the current partition deviates by more than a certain cutoff threshold of the best solution found at that moment. A cutoff threshold of 0 means that no moves with negative gains are allowed and then the refinement behaves as a steepest descent algorithm. The higher the threshold, the more it approaches the original solution without threshold. As the graph size gets larger, less improvement is reached by allowing moves with negative gains. On the right the average execution time of the different algorithms is plotted. Using a cutoff threshold can lower the execution time at the cost of solution quality. Choosing a threshold high enough (1500 in our case) leads to a small profit in execution time without sacrificing much solution quality.

Fig. 6 shows the solution quality of the SA algorithm compared to the MLKL algorithm (left) and the execution time (right) for different number of execution runs. For smaller graphs SA finds better solutions than MLKL, but as the graph size gets larger MLKL becomes better, as due to the increase in search space SA will less likely find the random move that leads to a better solution. More execution runs of the SA algorithm leads to better solution quality at the cost of more execution time. Since the experiments were conducted on quad core machines there is a much bigger increase between 4 and 8 than between 1 and 4. As already seen for small

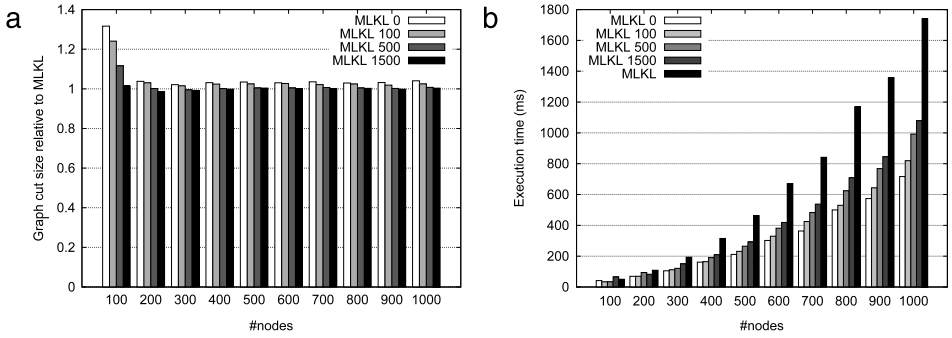


Fig. 5. The left graph shows the influence of a cutoff threshold on the solution quality relative to the MLKL algorithm without cutoff threshold, while the right shows the execution time. A lower cutoff threshold lowers execution time at the cost of a lower solution quality. The improvement of solution quality by allowing moves with negative gains is lowered as graph size gets larger.

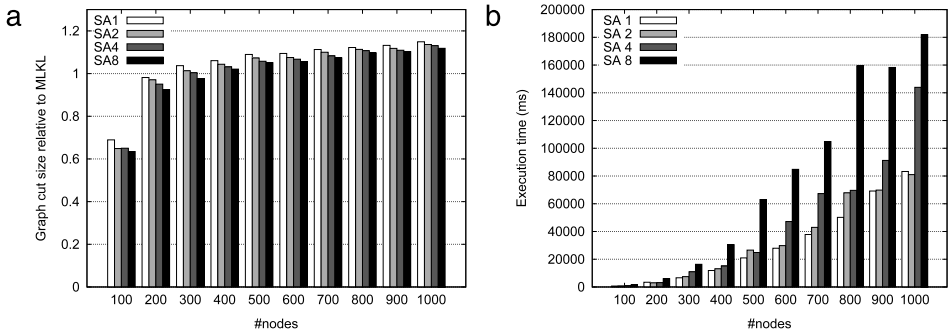


Fig. 6. The left graph shows the solution quality of SA relative to the MLKL algorithm, while the right shows the execution time. For smaller graphs SA results in better partitions, but uses much more execution time than MLKL.

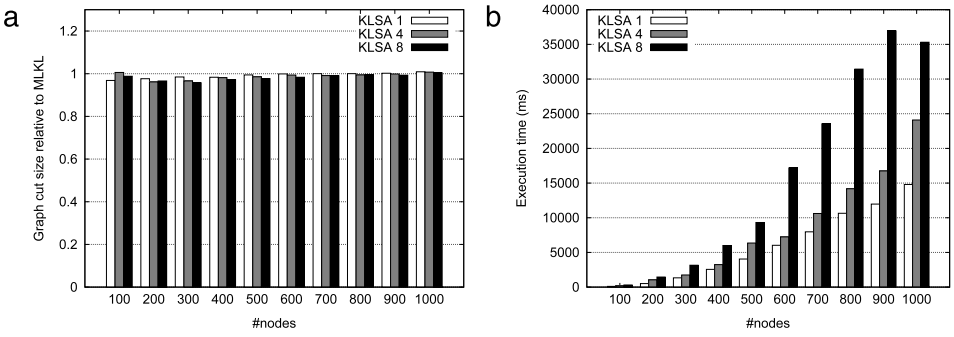


Fig. 7. The left graph shows the solution quality of KLSA relative to the MLKL algorithm, while the right shows the execution time. KLSA results in slightly better partitions than MLKL, but this improvement diminishes as the graph size gets larger. Regarding execution time it performs better than SA, but it is still slower than MLKL.

graphs in Fig. 4, the execution time of SA is much higher than that of MLKL.

The hybrid approach (KLSA) shown on Fig. 7 is a few percent better than MLKL, but this improvement becomes smaller as the graph size gets larger. Again this improvement comes at the cost of execution time. Since the expensive SA refinement is now only executed at the coarsest level, its execution time is lower than SA but still much higher than MLKL.

5.2. Comparison with METIS 4.0

A special case of our problem statement, where $\forall m, n : b_{mn} = 1$ and $\forall m : M_m = (\sum_i w_i)(1+\beta)/K$ represents the traditional graph partitioning problem that partitions the graph in K balanced partitions with imbalance factor β. For this problem we can compare our algorithm to well known graph partitioning software such as METIS [24].

Fig. 8 shows the solution quality of our MLKL algorithm relative to the solution of KMETIS to partition our graphs into 2, 4, 8, 16,

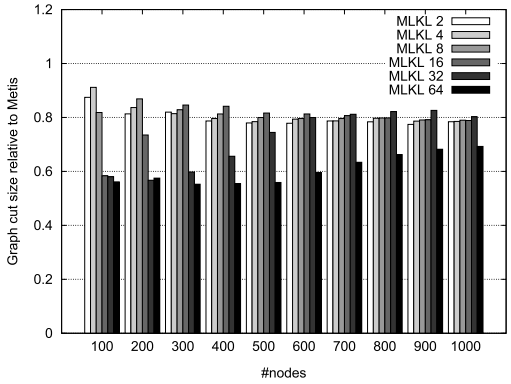


Fig. 8. Solution quality of the MLKL algorithm relative to METIS 4.0 to partition the graph in 2, 4, 8, 16, 32 or 64 balanced parts. Our algorithm results in 10%–40% lower graph cuts.

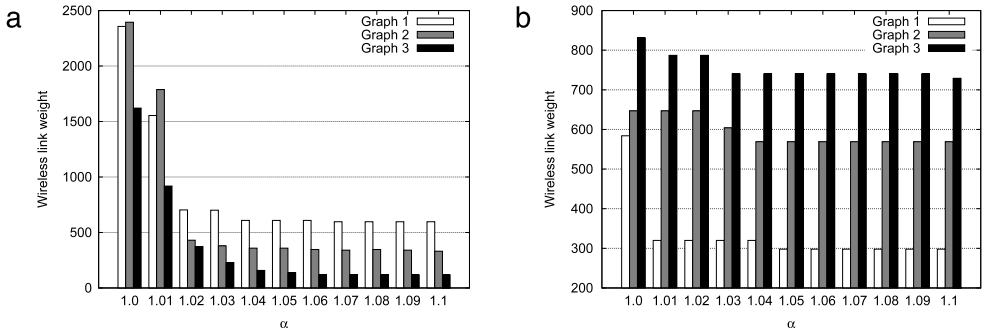


Fig. 9. By increasing the parameter α the bandwidth on the wireless link can be influenced. For fine grained graphs composed of many small components shown on the left the influence is more gradual than for the coarser graphs shown on the right.

32 and 64 balanced parts. Although both algorithms are based on the multilevel partitioning paradigm and KL-like refinement, our algorithm results in 10%–40% lower cut sizes. This is due to the fact that our algorithm allows more imbalance during the refinement, and a faster cutoff during refinement of METIS, as it is designed to scale to larger (>10,000 vertices) graphs.

5.3. Offloading scenario

Until now we always assumed that all application components are deployed in the cloud, where all servers are interconnected by the same links and thus $\forall m, n : b_{m,n} = 1$. However, in the context of mobile cloud computing where the cloud is used to offload parts of the application from the mobile device to the cloud, an important limiting factor will be the wireless link connecting the device to the internet. Depending on the technology and the signal strength the available bandwidth between the mobile device and the cloud – and thus the optimal deployment – will vary. In order to calculate an optimal deployment in this case, the b_{mn} parameters can be used to attach more importance to the wireless link and try to limit the amount of data exchanged via that link.

To calculate the optimal deployment in the offloading scenario, we added one extra machine m_{device} with a small capacity of 25 to our randomly generated list of machines and locked one of the nodes to this machine. This represents, for example, a node that takes care of the input or the GUI and must be deployed on the mobile device. The connectivity weights between the machines are set to 1 between the server machines, but to $\alpha \geq 1$ for each link between a server and m_{device}. When increasing α the algorithm will calculate a new deployment and will try to lower the bandwidth on the wireless link.

Of course the granularity with which one can influence the bandwidth on the wireless link will depend on the design of the software and the granularity of the components. If the software is composed of many small components one will be able to gradually tune the bandwidth requirements, but when the software is composed of a few big components fewer possibilities are available. Fig. 9 shows the weight on the wireless link as a function of α for six graphs. On the left results are shown for three graphs composed of 500 small ($\lambda = 0.5$) components, while on the right the results for three coarser graphs of 200 bigger ($\lambda = 0.1$) components are plotted. The more fine grained graphs on the left can be tuned more gradually than the coarser graphs on the right.

6. Conclusions and future work

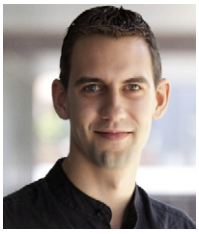
In this paper we presented algorithms for partitioning software graphs for deployment on the cloud. The best partition is calculated taking into account the infrastructure heterogenity. In contrast to deployment optimization for computational grids, we do not minimize the execution time of a set of mesh structured tasks, but we focus on minimizing the bandwidth between software components. A multilevel KL-based algorithm is presented as a fast partitioner which allows realtime deployment calculations. Simulated annealing improves the solution quality for small graph sizes, at the cost of computation capacity. A hybrid algorithm produces slightly better partitions than KL in an intermediate execution time. We compared our KL-based algorithm to METIS for a wide range of graphs and found 10%–40% better partitions. We also showed how our algorithm can be used to adapt the deployment to take into account wireless link quality in a mobile cloud computing use case.

Future work consists of integrating these algorithms in an actual software deployment framework to automatically distribute software components on a cloud infrastructure. The algorithms can also be adapted to optimize alternative objectives, for example energy consumption.

References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (6) (2009) 599–616.
[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, M. Zaharia, Above the clouds: a Berkeley view of cloud computing, Tech. Rep., 2009.
[3] Noesis solutions, optimus. http://www.noesissolutions.com/Noesis/.
[4] T. Verbelen, T. Stevens, P. Simoens, F. De Turck, B. Dhoedt, Dynamic deployment and quality adaptation for mobile augmented reality applications, Journal of Systems and Software 84 (2011) 1871–1882.
[5] S. Ou, K. Yang, J. Zhang, An effective offloading middleware for pervasive services on mobile devices, Pervasive and Mobile Computing 3 (4) (2007) 362–385.
[6] M. Kristensen, Scavenger: transparent development of efficient cyber foraging applications, in: 2010 IEEE International Conference on Pervasive Computing and Communications, PerCom, 2010, pp. 217–226.
[7] T. Verbelen, P. Simoens, F. De Turck, B. Dhoedt, Cloudlets: bringing the cloud to the mobile user, in: Proc. of the 3rd ACM Workshop on Mobile Cloud Computing & Services, MCS'12, 2012.
[8] S. Han, S. Zhang, J. Cao, Y. Wen, Y. Zhang, A resource aware software partitioning algorithm based on mobility constraints in pervasive grid environments, Future Generation Computer Systems 24 (6) (2008) 512–529.
[9] C. Alpert, Recent directions in netlist partitioning: a survey, Integration, The VLSI Journal 19 (1–2) (1995) 1–81.
[10] B. Hendrickson, Graph partitioning models for parallel computing, Parallel Computing 26 (12) (2000) 1519–1534.
[11] H. Meyerhenke, B. Monien, S. Schamberger, Graph partitioning and disturbed diffusion, Parallel Computing 35 (10–11) (2009) 544–569.
[12] T.N. Bui, C. Jones, Finding good approximate vertex and edge partitions is NP-hard, Information Processing Letters 42 (3) (1992) 153–159.
[13] P.-O. Fjallstrom, Algorithms for graph partitioning: a survey, Computer and Information Science 3 (10) (1998).
[14] K. Schloegel, G. Karypis, V. Kumar, The Sourcebook of Parallel Computing, Morgan Kaufmann, 2003, pp. 491–541.
[15] B. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Technical Journal 49 (2) (1970) 291–307.
[16] C. Fiduccia, R. Mattheyses, A linear-time heuristic for improving network partitions, in: Papers on Twenty-Five Years of Electronic Design Automation, ACM, 1988, p. 247.

[17] L. Sun, M. Leng, An effective multi-level algorithm based on simulated annealing for bisecting graph, Lecture Notes in Computer Science 4679 (2007) 1.
[18] L. Sun, M. Leng, S. Yu, A new multi-level algorithm based on particle swarm optimization for bisecting graph, Lecture Notes in Computer Science 4632 (2007) 69.
[19] M. Leng, S. Yu, An effective multi-level algorithm based on ant colony optimization for bisecting graph, Lecture Notes in Computer Science 4426 (2007) 138.
[20] B. Hendrickson, R. Leland, A multilevel algorithm for partitioning graphs, in: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing, CDROM-Supercomputing'95, 1995, 28-es.
[21] Y. Saab, An effective multilevel algorithm for bisecting graphs and hypergraphs, IEEE Transactions on Computers 53 (6) (2004) 641–652.
[22] C. Aykanat, B. Cambazoglu, B. Ucar, Multi-level direct K-way hypergraph partitioning with multiple constraints and fixed vertices, Journal of Parallel and Distributed Computing 68 (5) (2007) 609–625.
[23] A. Pothen, H.D. Simon, K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs, SIAM Journal on Matrix Analysis and Applications 11 (3) (1990) 430–452.
[24] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1) (1999) 359–392.
[25] F. Pellegrini, Scotch and libScotch 5.1 user's guide, LaBRI, Université Bordeaux I, August 2008. http://www.labri.fr/pelegrin/scotch/.
[26] C. Walshaw, M. Cross, JOSTLE: parallel multilevel graph-partitioning software an overview, in: Mesh Partitioning Techniques and Domain Decomposition Techniques, 2007, pp. 27–58.
[27] R. Khandekar, S. Rao, U. Vazirani, Graph partitioning using single commodity flows, Journal of the ACM 56 (4) (2009) 1–15.
[28] P. Chardaire, M. Barake, G.P. McKeown, A PROBE-based heuristic for graph partitioning, IEEE Transactions on Computers 56 (12) (2007) 1707–1720.
[29] R. Loureiro, A. Amaral, An efficient approach for large scale graph partitioning, Journal of Combinatorial Optimization 13 (4) (2007) 289–320.
[30] J.G. Martin, Spectral techniques for graph bisection in genetic algorithms, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation—GECCO'06, 2006, p. 1249.
[31] S. Kumar, S. Das, R. Biswas, Graph partitioning for parallel applications in heterogeneous grid environments, in: Proceedings of the 16th International Parallel and Distributed Processing Symposium, vol. 00, 2002, p. 167.
[32] B. Arafeh, K. Day, A. Touzene, A paradigm for allocating parallel application tasks to heterogeneous computing resources on the grid, in: Proceedings of the International Conference ParCo'05, 2005.
[33] P. Phinjaroenphan, S. Bevinakoppa, P. Zeephongsekul, A heuristic algorithm for mapping parallel applications on computational grids, Advances in Grid Computing 3470 (2005) 1086–1096.
[34] S. Huang, E. Aubanel, V.C. Bhavsar, PaGrid: a mesh partitioner for computational grids, Journal of Grid Computing 4 (1) (2006) 71–88.
[35] D.J. Harvey, S.K. Das, R. Biswas, Design and performance of a heterogeneous grid partitioner, Algorithmica 45 (3) (2006) 509–530.
[36] J. Chen, V. Taylor, Mesh partitioning for efficient use of distributed systems, IEEE Transactions on Parallel and Distributed Systems 13 (1) (2002) 67–78.
[37] IBM ILOG CPLEX. http://www.ilog.com/products/cplex/.
[38] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680. (New York, NY).
[39] V. Cerny, Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, Journal of Optimization Theory and Applications 45 (1) (1985) 41–51.
[40] D.S. Johnson, C.R. Aragon, L.a. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning, Operations Research 37 (6) (1989) 865–892.
[41] M.-W. Park, Y.-D. Kim, A systematic procedure for setting parameters in simulated annealing algorithms, Computers & Operations Research 25 (3) (1998) 207–217.
[42] D. Eppstein, J. Wang, A steady state model for graph power laws, in: 2nd International Workshop on Web Dynamics, 2002.



Tim Verbelen received his M.Sc. degree in Computer Science from Ghent University, Belgium in June 2009. Since then he has been working as a Ph.D. researcher at the Department of Information Technology (INTEC) of the Faculty of Engineering at Ghent University. His main research interests include mobile computing, context-aware computing and adaptive software.

Tim Stevens received his M.Sc. degree in Computer Science from Ghent University, Belgium, in June 2001. In January 2009, he obtained his Ph.D. degree in Computer Science Engineering from the same university. Before joining the Department of Information Technology (INTEC) of Ghent University in August 2003, Tim was a database and system administrator for the Flemish public broadcasting company (VRT). At present, he is a postdoctoral researcher affiliated with INTEC.

His main research interests include future access network architectures, IPv6, Quality of Service (QoS) and traffic engineering in IP networks, anycast-based services, and cloud and grid computing.

Filip De Turck received his M.Sc. degree in Electronic Engineering from the Ghent University, Belgium, in June 1997. In May 2002, he obtained the Ph.D. degree in Electronic Engineering from the same university. During his Ph.D. research he was funded by the F.W.O.-V., the Fund for Scientific Research Flanders. From October 2002 until September 2008, he was a post-doctoral fellow of the F.W.O.-V. and part time professor, affiliated with the Department of Information Technology of the Ghent University. At the moment, he is a full-time professor affiliated with the Department of Information Technology of the Ghent University and the IBBT (Interdisciplinary Institute of Broadband Technology Flanders) in the area of telecommunication and software engineering. Filip De Turck is author or co-author of approximately 250 papers published in international journals or in the proceedings of international conferences. His main research interests include scalable software architectures for telecommunication network and service management, cloud computing, performance evaluation and design of new telecommunication and eHealth services.

Bart Dhoedt received a Masters degree in Electro-technical Engineering (1990) from Ghent University. His research, addressing the use of micro-optics to realize parallel free space optical interconnects, resulted in a Ph.D. degree in 1995. After a 2-year post-doc in opto-electronics, he became Professor at the Department of Information Technology.

Bart Dhoedt is responsible for various courses on algorithms, advanced programming, software development and distributed systems. His research interests include software engineering, distributed systems, mobile and ubiquitous computing, smart clients, middleware, cloud computing and autonomic systems. He is author or co-author of more than 300 publications in international journals or conference proceedings.

A Knowledge Plane for the Internet

David D. Clark*, Craig Partridge♦, J. Christopher Ramming† and John T. Wroclawski*

*M.I.T Lab for Computer Science
200 Technology Square
Cambridge, MA 02139
{ddc,jtw}@lcs.mit.edu

♦BBN Technologies
10 Moulton St
Cambridge, MA 02138
craig@bbn.com

†SRI International
333 Ravenswood Avenue
Menlo Park, CA 94205 USA
chrisramming@yahoo.com

ABSTRACT

We propose a new objective for network research: to build a fundamentally different sort of network that can assemble itself given high level instructions, reassemble itself as requirements change, automatically discover when something goes wrong, and automatically fix a detected problem or explain why it cannot do so.

We further argue that to achieve this goal, it is not sufficient to improve incrementally on the techniques and algorithms we know today. Instead, we propose a new construct, the Knowledge Plane, a pervasive system within the network that builds and maintains high-level models of what the network is supposed to do, in order to provide services and advice to other elements of the network. The knowledge plane is novel in its reliance on the tools of AI and cognitive systems. We argue that cognitive techniques, rather than traditional algorithmic approaches, are best suited to meeting the uncertainties and complexity of our objective.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *network communications.* C.2.3 [**Computer-Communication Networks**]: Network Operations – *network management, network monitoring.* C.2.6 [**Computer-Communication Networks**]: Internetworking.

General Terms

Management, Measurement, Design, Experimentation.

Keywords

Cognition; network applications; network configuration; knowledge plane.

1. INTRODUCTION

The Internet of today is a wonderful success. But success should not blind us to the Internet's limitations. Its emphasis on generality and heterogeneity, the 'narrow-hourglass' combination of a simple, transparent network with rich end-system functionality, and the deeply embedded assumption of a decentralized, multi-administrative structure are critical strengths, but lead to frustrated users when something fails, and high management overhead with much manual configuration, diagnosis and design.

Both user and operator frustrations arise from the same fundamental design principle of the Internet—the simple and transparent core with intelligence at the edges [1,2]. The network carries data without knowing what that data is, or what its purpose is. If some combination of events is keeping data from getting through, the edge may recognize that there is a problem, but the core cannot tell that something is wrong, because the core has no idea what *should* be happening. The edge understands applications, and what their expected behavior is; the core only deals with packets. Similarly, a network operator interacts with the core in very low-level terms such as per-router configuration of routes and policies. There is no way for the operator to express, or the network to model, what the high level goal of the operator is, and how the low-level decisions relate to that high level goal.

As we design a new sort of network, we must not lose the features of the Internet that have made it a success—its openness to new applications, the adaptability of its protocols, and the essential plasticity basic to its nature. Yet we must devise a technique that marries these virtues to a new goal: the ability of the network to know what it is being asked to do, so that it can more and more take care of itself, rather than depending on people to attend to it. If the network had a high-level view of its design goals and the constraints on acceptable configurations, then it could make many low-level decisions on its own. It could communicate with the network designer in terms of how well it met the goals, rather than by displaying a mass of router configuration tables. And it could deal with changes in the high level requirements by reconfiguring itself.

We argue that traditional, algorithmic approaches to adaptivity are unlikely to provide the required sophistication of behavior. The approach we take must offer the ability to abstract and isolate high level goals from low level actions, to integrate and act on imperfect and conflicting information, and to learn from past actions to improve future performance. These properties are precisely those required to function effectively in the Internet's environment of diverse and competing objectives, decentralized control, complexity, and dynamic change.

This paper proposes an approach to network design based on tools from AI and cognitive systems. Specifically, we propose a construct,

SIGCOMM'03, August 25–29, 2003, Karlsruhe, Germany.

a distributed cognitive system that permeates the network, that we call the **knowledge plane**.

The rest of this paper is organized as follows. Section 2 describes the concept of the knowledge plane. It contrasts this concept with alternatives, and argues for the cognitive approach. Section 3 is a discussion of what this construct might do for us—examples of how it can make networking better. Section 4 discusses some important design constraints and considerations for a knowledge plane architecture. Section 5 outlines the key challenges in our path.

2. A Proposal: the Knowledge Plane

The discussion above hints at a solution in which the network has a high-level view of what its purpose is—the goals of its designers, of the applications running on it, and of its users. In an application-specific network, one approach might be to utilize and embed such domain-specific knowledge in the core design of the network, as was done in the telephone network. But this defeats a fundamental objective of the Internet – its ability to host a broad and changing array of applications. Rather than pleasing no one by adding "just a little" application knowledge to the Internet's simple and transparent data transport plane, a better alternative is to devise a separate construct that creates, reconciles and maintains the many aspects of a high-level view, and then provides services and advice as needed to other elements of the network. This is the knowledge plane, or KP.

Understanding the precise best path towards this goal is a matter of significant research, and this paper neither can nor does propose a complete technical description of the knowledge plane. As a start, however, we sketch certain attributes potentially central to the success of a knowledge plane, and consider how this perspective differs from today's practice. These include:

Edge involvement: The end-to-end principle suggests that much valuable information about network performance originates not in the network, but in the devices and applications that use it. This is an inevitable and desirable consequence of the Internet's general-purpose data plane. It implies, however, that much of the "knowledge" in the knowledge plane may be produced, managed, and consumed at *or beyond* the "traditional" edge of the network. The reach of the knowledge plane is broader than that of traditional network management.

Global perspective: Most management systems are regional — each operator manages the part he owns. But truly useful problem identification may depend on correlation of observations from different parts of the network. Not only must data from the edges be combined with data from "inside" the network, but data from different parts of the network may be needed to fully comprehend a sequence of events. The knowledge plane would, ideally, be able to extend its perspective to the entire global network as required.

Compositional structure: If the reach of the KP is global, at the same time it must be designed to take account of what we may loosely call "compositional" considerations. A most basic example is that the KPs of two unconnected networks should be capable of merging their perspective and activities if the networks become connected.

A corollary of the composition problem is the need to operate in the presence of imperfect and conflicting information: some regions will desire to keep date private. Mutual distrust among some network operators and service providers, and indeed, among any parties that jockey for economic advantage, leads directly to today's need for highly skilled human reasoning to deduce and model network behavior. The KP faces a similar problem: it cannot assume a homogeneous network of shared objectives and shared information.

Unified approach: One might speculate that the various problems we aim to address could most easily be solved by distinct mechanisms, working bottom up, perhaps loosely tied together at the top. In contrast, the KP as we conceive it is a single, unified system, with common standards and a common framework for "knowledge". This unified approach is needed because real world knowledge is not strictly partitioned by task. We suggest that the knowledge plane should be structured similarly, based on the *knowledge*, not the task. We believe that while point solutions may be easier to develop, an integrated approach will be substantially more effective in the long run.

Cognitive framework: The knowledge plane needs to make judgments in the presence of partial or conflicting information; to recognize and mediate conflicts in policies and goals; to respond to problems and attacks in better-than-human time frames; to perform optimizations in high-dimensional environments that are too complicated to be addressed by humans or analytical solutions; and to automate functions that today must be carried out by rare and highly skilled network technicians. We therefore expect cognitive techniques to serve as the foundation of the knowledge plane: representation, learning, and reasoning that allow the knowledge plane to be "aware" of the network and its actions in the network.

We turn now to further discussion of three ideas key to our position: the necessity of a new construct, the desirability of a unified knowledge plane, and the value of cognitive tools.

2.1 Why a New Construct?

Most discussions of network architecture recognize two architectural divisions, or planes: a data plane, over which content is forwarded, and a control or management plane, which is used to direct, measure, and repair the data plane. By talking of a "knowledge plane" we are saying a fundamentally new construct is required, rather than fitting knowledge into an existing plane (presumably the management plane). Why do we believe a new construct is required?

If we look at the two existing planes, we find two radically different structures. The data plane (in almost any notable data transport architecture) uses some form of layering to hide complexity and to enable extensibility, interoperability and scalability. In contrast the control and management system is invariably designed to cut across the layering, giving visibility and access to all the aspects of the network, which must be monitored and managed. And, indeed, because the management plane is all-seeing, it tends to scale poorly and to be hard to change.

The knowledge plane clearly sits in a different place. Since it doesn't move data directly, it is not the data plane. And unlike the management plane it tends to break down boundaries to provide a unified view, rather than partition the world into managed enclaves. It is functionally unlike the management plane as well – it is hard to envision the KP managing accounting records (reading them occasionally, perhaps, but collecting, storing and processing them, no).

2.2 Why a Unified Approach?

Consider the example of a user trying to install a new application and discovering that it does not work. One reason might be that the ISP of the user has blocked that class of traffic. For the KP to give

the most effective feedback to each party, it needs access to the configuration constraints set by the ISP, so it can determine the rules behind the blocking and tell the user what this implies. So it is necessary that the information about network configuration and about user-observed problems be in the same framework.

A related example concerns overlay network such as CDNs. It is easy to imagine that one component of the KP is topology and performance information that a CDN could use to position its delivery nodes "close" to users. This information could come from a diversity of sources such as "network weather" services, user-reported experience, and ISPs, and would include not just traffic measurements, but information about administrative traffic restrictions and local firewall restrictions (perhaps the "users" can't receive certain types of content). The interested parties (users, CDNs) benefit from having this information integrated and presented in a consolidated form.

There are some cases where the KP may be able to resolve a problem on its own. If it discovers that the reason for a problem is a low-level decision that is not material to the high-level goals of the operators, it might change the decision. But to determine if a change is appropriate, KP needs access to the reasoning behind the setting. So the knowledge about planning needs to be in the same context as the repair problem.

When a component of the network makes a low-level observation about a possible anomaly, it has no idea what the relevance actually is. This observation might trigger a repair, a reconfiguration, a notification to a network operator in a distant part of the network, a security alert, or something else quite different. So observations on network conditions cannot be thought of as being a part of one problem space, but instead as being a part of the KP.

We recognize that point solutions to specific problems may get part of the way more rapidly that the general solution postulated here. But the core of our hypothesis is that to get to the final goal: a network that can configure itself, that can explain itself, that can repair itself, and does not confound the user with mysteries, the approach based on the combination of point solutions will not succeed.

2.3 Why a Cognitive System?

Our objectives for the Knowledge Plane require it to meet a number of significant challenges:

- It must function usefully in the presence of incomplete, inconsistent, and possibly misleading or malicious information. System failures, information filtering for privacy or competitive reasons, and finite network resources are just some of the forces conspiring to create this requirement.
- It must perform appropriately in the presence of conflicting or inconsistent higher-level goals among the Internet's different stakeholders. This is a manifestation of the *tussle* dilemma discussed in [12].
- It must operate effectively in the face of generality, including the introduction of new technologies and applications not conceived of at the time of its design, and in the face of a highly dynamic environment, including both short-term and long-term changes in the structure and complexity of the underlying network.

We hypothesize that these challenges cannot be met by analytical solutions, because analytical solutions generally require complete information, precise problem formulations, and a relatively static operating environment. Instead, we suggest that "cognitive" techniques will be needed. The key benefit of these techniques is their potential to perform effectively, and to evaluate and improve their own performance, in the presence of complex, inconsistent, dynamic, and evolving environments. We discuss two defining characteristics of a cognitive knowledge plane.

First, the KP must eventually "close the loop" on the network as does an ordinary control system. As we gain experience and trust, the knowledge plane will first enable a *recognize-explain* cycle, then a *recognize-explain-suggest* cycle, and ultimately a *recognize-act* cycle for many management tasks. Because the knowledge plane must be more general and flexible than standard control systems, we look elsewhere for additional inspiration. Architectures inspired by theories of human cognition [18] have achieved some successes and hint at one approach. In the knowledge plane context, a cognitive architecture would of course be distributed and decentralized, and the partitioning would be effected in part to support divergent interests of network stakeholders.

Second, the KP must be able to learn and reason. Learning is the principled accumulation of knowledge, and can take place through many means: by trial and error, by instruction, by generalization, by analogy, through problem solving and mental search, and more. Some learning approaches require human involvement, and some do not. In a static problem environment, one simple enough to admit of analytic solution, learning is irrelevant. But IP networks, by design and intent, are constantly evolving in many dimensions, and are infinite in potential configurations. To the extent possible, when new situations are recognized or new actions performed and evaluated, the knowledge plane should improve: its knowledge base should grow in useful ways. The first and most immediate challenge of learning is to model the behavior, dependencies, and requirements of applications through the obscuring veil of our existing transparent data plane.

Reasoning involves the composition of existing knowledge to draw new inferences and beliefs. Reasoning processes can translate declarative knowledge (whether handcrafted or learned) into interpretations of observations and decisions about actions. If we wish the network of the future to support high-level goals and constraints, we will need reasoning methods that can operate on these abstractions.

In the long run, an interesting and important function of reasoning in the knowledge plane will be to support mediation between users and operators whose goals may conflict with each other and/or with fixed design constraints. The inevitability of such conflicts suggests that we must develop new techniques for representing and reasoning about constraints and policies. Initially, these representations will need to be inferred from low-level configurations and actions, but the ultimate goal is to express goals and policies at a high level and use those to generate low-level configurations.

Even in the short run we can bring to bear a great deal of existing research on the design and construction of a knowledge plane. Experience with cognitive architectures [18], recent work in multi-agent systems [22], and the emerging field of algorithmic game theory may prove directly useful. However, the networking context also raises many challenges that will stretch the current state of cognitive systems and redirect research in new and intriguing ways [19,20].

3. What is the Knowledge Plane Good For?

At a high level, we proposed a unified goal for the KP: build a new generation of network by allowing it to have a view of what it supposed to be, and what it is supposed to be doing. To achieve this goal, there are more specific problem domains to be supported. Here we discuss in more detail some of them.

Fault diagnosis and mitigation: Today, when some part of the Internet fails, it is almost impossible for the end user to tell what has happened, to figure out who should be notified, or what to do to correct the fault. If we take the Internet of today as the starting point, it is appealing to imagine a command that a user can run to demand an explanation when something seems to be broken. This is the WHY(problem-x) command: why is x broken? So, for instance, the user might ask, "Why can't I get to www.acm.org?"

However, the WHY formulation is not bold enough. An over-bold alternative would be that if the KP is smart enough, the network should never fail. In this case, there is no need for WHY. But this ambition is fundamentally flawed. In some cases, only a human knows enough to determine if what is happening is actually a fault. When Dave unplugs his laptop and puts it in his briefcase, there may be some applications that suddenly stop working, but this is not a fault. It is what Dave intended, and if some semi-smart KP wakes up each time he disconnects his laptop and asks if he wants to reconnect it, this is a nightmare, not a success. So there will be times when only a person can give the KP guidance. Instead of WHY(problem-x), this is FIX(problem-x). The user is saying that something is broken, and make it right.

Is this enough guidance that the KP can correct what is wrong? In fact, the interesting examples are when the "problem" is caused by conflicting specifications or constraints that come from different people. One may say FIX(this game I just installed that does not work), and the reason it is failing is that the ISP has blocked that game. One may say FIX(lousy bandwidth) but the problem is that one has exceeded one's usage quota and the ISP is rate-limiting. These are cases where the KP may not be able to resolve the matter. What we might strive for, however, is a KP that can either resolve a problem or say why not. So one answer to FIX(problem-x) may be CANNOT(reason-y). And if the system does fix something, it may want to tell a person that this happened, in case there is a further action that only a person can take.

This example suggests that the interaction between the user and the KP is bi-directional and expressive. And of course, the KP may communicate with many entities about a problem. The demand from a user FIX(broken-game) might trigger a message back to the user that the game is blocked, but might also trigger a message to the ISP that it has an unhappy user.

A further extension of this story is that the KP can provide an assistant for user and managers, an agent that watches what the people do, and learns over time what is normal and what is not. So a KP agent on Dave's laptop might learn that Dave unplugs it all the time, while an agent on Dave's desktop machine might realize that he never disconnects it, and risk bothering Dave to ask if he meant to do that. In this way, the problem of fault diagnosis and mitigation has a learning component.

Once the FIX(problem-x) function has been implemented in the KP, programs as well as people can use it. As the user's agent learns, it should more and more often give this signal on its own. And other programs, such as application code, may detect and signal that something is wrong. The KP will have to decide how much credence to give these signals, depending on where they come from.

Behind the scenes, the FIX command will trigger a range of activities in the KP. The FIX command would start with a local component that runs on the user's machine, and then exchanges information with the KP to figure out what is wrong. The diagnostics can check out functions at all levels, from packet forwarding to application function. There are several current research projects that this application could build on [13,14].

Once the end node has performed what diagnosis it can, the next stage is for the tool to add assertions to the shared knowledge plane about what it has discovered, and ask the KP for relevant information. This contribution to the knowledge plane allows all the users on the network collectively to build a global view of network and service status. This data can be combined with information derived from measurement efforts now going on across the Internet that attempt to build an overall model of network status [9,15]. Such aggregation is important if the failure is one that affects lot of users.

Automatic (re)configuration: The dynamic routing of the original Internet did not take into account administrative and policy constraints, so routing today is more and more defined by static policy tables. This means that devices such as routers are increasingly manually configured and managed. Static tables and manual configuration make the network brittle to failure, hard to change, and even harder to reason about globally. Imagine, as part of the KP, a configuration manager for a region of the Internet, which would accept high-level assertions about how the components of a network are supposed to arrange themselves, and guide the actual detailed configuration accordingly. Examples include controlling the deployment of a consumer network in the home, an ad hoc network in support of a rapid deployment force, or a network for a small business. Successful accomplishment of this project could lead to substantial reductions in manpower needed to configure and operate networks.

The KP configuration manager should have enough understanding of low-level structure to detect if the network is properly configured according to the high-level constraints, to detect if a better configuration alternative is available, and to detect if the system appears to be corrupted. The reasoning must go in both directions. That is, the manager must be able to derive low-level settings given high-level goals, priorities and constraints, and it must be able to look at existing low-level settings and describe the resulting behavior in the high-level terms.

Again, the interesting problem (once we get the basic idea to work) is when the system encounters conflicting assertions made by different parties. The network manager might say ROUTING_PREFERENCE(low-cost links), and an end-user's machine might say FIX(low bandwidth). Again, the KP may be able to resolve some of these problems, and might learn over time when it is safe for it to act on its own, and when it must kick the problem back to the relevant humans in meaningful terms. (This example, by the way, illustrates why the KP must be seen as a unified system, not as separate systems for fault management and for configuration.)

The configuration task is not something that happens once at the turn-up of the network. It should be something that is happening constantly, looking at changing network conditions, application demands, and changing constraints. It is also a task that can run "recursively". A global network is not built top down. It is built bottom up, region by region. Each region will first configure itself

using its locally specified goals and constraints. But when two regions then connect, there may be further constraints that one imposes on the other. So a provider network might say to a subscriber network: NO_MULTICAST. This might cause the subscriber network to change some of its internal organization, disable some end-user applications, and so on.

Support for overlay networks: If the KP has enough information to configure the network itself, that information can also be useful to applications that are configuring themselves. For example, we are increasingly seeing the development of application-specific overlay networks on the Internet. Each overlay network uses edge-based mechanisms to evaluate the performance of different possible paths through the Internet, and seeks to build a set of transport paths that effectively route application packets through what appears to be the part of the Internet best suited to the application's needs. Currently, application networks must probe the Internet, because there is no mechanism for them to learn about the capabilities of the network core. The KP would be in a position to aggregate application- and network-derived knowledge about network performance, offer applications better information about the network than they could learn by probing, and access to control points whose behavior could be modified to help better meet the applications' needs. The KP thus enables per-application control over traffic without the need to build per-application infrastructure throughout the network.

Knowledge-enhanced intrusion detection: There are a number of projects (and a number of products) that perform some sort of analysis to detect network intrusions. In general they look for patterns in data observed somewhere in the network. The current generation of these tools trigger both false positives and false negatives. It has been hypothesized that a next generation of tools for intrusion detection will require that observations from several points in the network will have to be correlated, in order to get a more robust and useful signal. The development of the knowledge plane provides a basis to implement this data gathering and correlation.

4. Knowledge Plane Architecture

Previous sections of this paper have outlined the goals we set for a knowledge plane. In this section, we consider aspects of its system structure. Our discussion is speculative: any successful KP architecture will be shaped by a number of requirements and constraints, not all of which are apparent today. At its highest level the architecture of the KP will be shaped heavily by two broad forces: its distributed, compositional structure, and its multi-scale, potentially global knowledge perspective.

Our ultimate objective is that networked systems should organize themselves, under the constraints and guidance of external inputs, to meet the goals of their stakeholders. Even in the near term, the KP must respect and build on the fact that networks have internal structure and dynamics -- large networks are composed by interconnecting smaller ones, participants come and go, and relationships between the owners, operators and users of different networks may change even when the physical structure does not.

This implies that the knowledge plane serving a network is not a globally engineered entity, but is instead an autonomously created structure that is recursively, dynamically, and continuously composing and decomposing itself from smaller sub-planes. This requirement argues that the KP:

- Is distributed - KP functionality for different regions of the network is physically and logically decentralized.
- Is bottom up - simple entities (e.g. web servers) can compose into larger, more complex entities (e.g. a web farm) as needed, and decompose themselves from the system as appropriate. This is a recursive process, that may proceed on many levels.
- Is constraint driven - the basic principle is that the system can, and may, adopt (or not) any behavior that is not specifically constrained.
- Moves from simple to complex. Speaking generally, the act of composing a set of networks to form a larger one places more requirements or constraints on the behavior of each network. A trivial example would be that a standalone IP network can use a wide range of addresses, but connecting it to a larger network constrains the range of options in this regard.

Our first objective for the KP system architecture is that it support this distributed, compositional perspective, providing the necessary enabling abstractions and capabilities.

In contrast with the distributed *organization* of the KP, we have argued in previous sections of this paper that KP may often benefit from taking a global *perspective* - integrating observations and conclusions from many points in the network. Key implications of this are that:

- Data and knowledge integration is a central function of the KP. The KP must be able to collect, filter, reduce, and route observations, assertions, and conclusions from different parts of the network to points where they are useful.
- The KP must operate successfully in the presence of imperfect information. Because this global perspective is both physically large and spans multiple administrative entities, the cognitive algorithms of the KP must be prepared to operate in the presence of limited and uncertain inputs.
- The KP must reason about information tradeoffs. Sometimes, a global perspective may be critical. Other times, it may be unimportant, or merely somewhat useful. The KP must be prepared to reason about the tradeoffs involved in using data of differing scope. For instance, diagnosing a web server failure may, or more likely, may not require polling for user experience from locations far away. A KP may need to employ introspective meta-reasoning to act most effectively in these circumstances.

Our second objective for the KP system architecture is that to the extent possible it develop, utilize, and reason about information at whatever scope is appropriate for the problem it is addressing.

4.1 Functional and Structural Requirements

The above objectives, together with the core goals of the knowledge plane, lead us to several top-level functional and structural architectural requirements. We discuss four of these below.

4.1.1 Core Foundation

The heart of the knowledge plane is its ability to integrate behavioral models and reasoning processes into a distributed, networked environment. The first component of this ability is support for the creation, storage, propagation and discovery of a variety of information, likely including *observations*, which describe current conditions; *assertions*, which capture high-level goals, intentions and constraints on network operations; and *explanations*, which are

an example of how knowledge itself is embodied—explanations take observations and assertions and map them to conclusions.

To learn about and alter its environment, the knowledge plane must access, and manage, what the cognitive community calls *sensors* and *actuators*. Sensors are entities that produce observations. Actuators are entities that change behavior (e.g., change routing tables or bring links up or down). So, for instance, a knowledge application that sought to operate a network according to certain policies might use sensors to collect observations on the network, use assertions to determine if the network's behavior complies with policy, and, if necessary, use actuators to change the network's behavior.

The most central aspect of the knowledge plane is its support for cognitive computations. This is a challenging problem because the dynamic and distributed KP environment is not well matched to the classical knowledge level algorithms and agent architectures that underpin much of current AI. Most AI algorithms are not designed to work in a highly distributed context, and direct experience in building a large distributed data management system with embedded cognitive abilities is limited.[1] What is needed are robust, tractable and on-line algorithms for environments that are highly dynamic, partially observable, stochastic and error prone. The field of Multi-Agent Systems [22] has had some initial success in solving these problems, although those addressed to date typically lack the dynamicity required for the knowledge plane. Thus refinement of this portion of the knowledge plane architecture, its infrastructure support for a range of appropriate cognitive algorithms, is likely to progress in conjunction with further research in cognitive systems themselves.

4.1.2 Cross-Domain and Multi-Domain Reasoning

Where does the KP "run"? The composed, regional structure of the KP might suggest that a specific server would support the part of the KP that "reasons about" a region, for example an Internet AS. One possibility is that the administrator of the AS would run the KP that oversaw that AS. At a more abstract level, one might state this structuring strategy as "each region is responsible for reasoning about itself."

This is a bad idea, for several reasons. If the AS is down, this could render the relevant KP information unreachable at exactly the wrong time. The administrator of an AS might wish to limit the conclusions that the KP reached about it, perhaps to remove knowledge that seems unflattering, while others may choose to reach those conclusions anyway. These examples show that reasoning about an AS occurs independently of the AS; a fact that should be reflected in the system structure. Different parts of the KP might independently reason about an AS, and compare answers, to detect that part of the KP is corrupted. This shows that there should be no specific physical relationship between a region of the network and the KPs reasoning engines related to that region.

A more radical possibility is that multiple entities compete to provide information about a given AS. Each entity collects its own data and sells its observations. The KP could seek information from whichever entity or entities it believes provides the most accurate and timely (or most cost effective) information. This ``knowledge marketplace" creates a host of architectural challenges, ranging from how to reason about information from multiple providers (even if three different companies tell you the same thing about an AS, it may turn out that they're all reselling data from one Internet weather service: if you really want a second opinion, how do you find the second weather service?) to how to design KP protocols to discourage different knowledge companies from subtly "enhancing" the KP protocols or data in ways that make it harder for users to concurrently use the servers of other knowledge providers?

This discussion demonstrates the potential richness of information flow in the KP. Messages need to flow to more than one location so that redundant reasoning can occur – and how a message flows may depend on who asks it. Different parts of the KP may reach different conclusions, and reconciling these is as important as is dealing with incomplete input data.

4.1.3 Data and Knowledge Routing

We have argued that the KP will benefit from gaining a global perspective on the network it serves. It is useful to consider how this perspective might come about. In a very small network, it might theoretically be possible to collect all relevant information, and flood that information to each node in the network (more precisely, in the distributed KP).

This idea is clearly impractical in larger networks. First, the sheer volume of information is technically daunting, requiring a highly scalable solution. Beyond this, forces such as competition and privacy come into play. In a network of any size, it is necessary to limit and optimize the collection and routing of information. More sophistication is needed.

We suggest that the KP architecture should implement a framework for knowledge management and routing characterized by two attributes. It is knowledge-driven - the routing system itself incorporates information about what knowledge is most useful in different circumstances, and uses scalable distributed techniques to filter observations and "attract" the most relevant observations towards potential customers. It understands tradeoffs - it may incorporate the concept of quality - reasoning about producing better or less good answers with correspondingly more or less effort, time, bandwidth, etc., rather than just producing "an" answer.

4.1.4 Reasoning about Trust and Robustness

The KP's combination of compositional structure and global perspective creates challenges to achieving a robust and trustworthy design. Because a functioning KP is formed at any time from the composition of the participating networks, the architecture must reflect the fact that parts of the KP may be corrupted or broken, that some participants may lie or export deliberately flawed reasoning, and that system actions must be based on inputs that may be partial, outdated or wrong.

This suggests that the KP may need to build, maintain, and reason about trust relationships among its components and participants. Portions of the KP that misbehave may be deemed untrustworthy by other portions, and this information may be propagated among portions that have decided to trust each other. In this way, a web of trust can grow that identifies KP elements that seem to be trustworthy and shuns elements that are not. This introspection would likely require the development of trust models, and the use of

[1] One early and related attempt, the DARPA-sponsored Automated Network Management (ANM) project, sought to build a network-wide MIB collector combined with AI tools [7]. The ANM experience was that collecting data was relatively easy, but getting the data to the right place was hard – it was easy to overwhelm links with management traffic if information was circulated too aggressively.

scalable techniques (such as the so-called "small world" models [10]) to search a web of trust.

5. Creating a Knowledge Plane

5.1 Possible Building Blocks

There is substantial basic research that may be relevant to creating the KP. Examples include epidemic algorithms [5] for distributing data, Bayesian networks for learning [4], rank aggregation to enable a web of trust [6], constraint satisfaction algorithms [21], and policy-based management techniques [23,24]. All of these techniques have been developed in other networking contexts and seem likely to be relevant here.

5.2 Challenges

If we are to create a successful knowledge plane, we must grapple with and solve a number of challenging problems. Because one of the goals of the knowledge plane is to give applications the ability to learn and reason about their environment, many of these problems sit at the boundaries of networking and artificial intelligence.[2] This section sketches some of the key themes that run through these problems.

How do we represent and utilize knowledge? We want the knowledge plane to support reasoning (figure out why John can't reach www.example.edu) and learning (the last time www.example.edu was unreachable, there was a DNS problem, so let's check DNS performance). The current state of the art in reasoning and learning tells us that we need to build abstract models of the entities we seek to understand, and then use information to reason about, and potentially update, those models. Current research into schemes for representation, such as the DAML Project[3], may give us some insight into how to represent information about which we can reason. However, we must also work out how to extract and process all the valuable information that presumably is not in DAML (or whatever form we pick) but in SNMP MIBs, system logs, and other disparate places. How do we construct, represent, and distribute the models that drive the reasoning?

How do we achieve scalable utility? The knowledge plane is a building block for a network that is more reliable and more robust. Properly implemented, it should continue to improve the network, even as the network gets bigger and the knowledge plane itself gets bigger. As we add more knowledge and new applications to the knowledge plane, it should become more valuable and useful overall. Those are hard goals: as the volume of data increases, or the number of elements in a system grows, we all too commonly find bottlenecks and algorithms that do not scale. For example, if a network failure triggers a flood of messages into the KP, how are these aggregated and controlled so that parts of the KP are not driven into overload? We will likely find ourselves challenged to abstract data and impose compartments or hierarchy on portions of the knowledge plane to allow it to scale – how do we ensure that the abstraction and compartmentalization adds rather than subtracts value?

How do we route knowledge? Suppose the knowledge plane learns a valuable new fact, or comes to a valuable realization. How is that fact or realization disseminated? Is it pushed out to all interested parties? If so, how do we know who the interested parties are? Is the fact simply labeled and placed into the knowledge plane for interested parties to discover? If so, how do the interested parties know to look for it? Are there ways to intelligently summarize data that make these push-pull tradeoffs easier?

How do we provide the right economic incentives? The networking community has come to learn that the success of a distributed system depends, in large part, on the economic incentives embedded in the system's design [11,12]. The knowledge plane is rife with economic challenges. How do we motivate people to put information into the knowledge plane? Much of the data in the knowledge plane will be valuable – should the knowledge plane provide mechanisms for people to buy and sell information (or better, "knowledge")? How do we avoid making the knowledge plane protocols a point of economic competition (e.g., avoid the vendor-specific enhancements to HTML problem)?

How do we deal with malicious and untrustworthy components? There is no way that we can expect that all nodes in the KP are trustworthy, competent or reliable. Broken nodes may inject malformed observations, some nodes may lie about their behavior, and some players may attempt to disrupt or confuse the KP, either as a way to attack the network as a whole, or to gain some advantage over others. How can the algorithms of the KP protect themselves, filter out bad information, and reach valid conclusions in the presence of uncertainty and misrepresentation? The KP system will have to depend on approaches such as consensus, rating, and cross-checking to detect mal-formed or malicious behavior. A design that is robust to inconsistent inputs is necessary for success.

As proposed above, a model of trust should be a core building block of the KP. Building a model of trust requires that there be some persistent robust expression of identity. There is no requirement that the identity be linked to a actual person (although for some purposes this may be preferred); the minimum requirement is that identity not be forged or stolen, so that one can build up a consistent model of trust based on prior observations of that identity.

6. Summary

This paper proposed to augment a network with a knowledge plane, a new higher-level artifact that addresses issues of "knowing what is going on" in the network. At an abstract level, this is a system for gathering *observations*, *constraints* and *assertions*, and applying rules to these to generate *observations* and *responses*. At the physical level, this is a system built out of parts that run on hosts and servers within the network. It is a loosely coupled distributed system of global scope.

The grander goal is to build a new generation of network, a network that can drive its own deployment and configuration, that can diagnose its own problems, and make defensible decisions about how to resolve them.

Previous attempts to do "high-level network management" have not been very successful; one possible reason is that previous projects have not been able to find the correct high-level abstractions. The hypothesis behind the KP is that there exist suitable ways to abstract detailed behavior, and to talk about goals, plans, constraints and methods at a high level. The knowledge plane is much more than a

[2] For a general overview of knowledge representation issues, the reader may wish to read [16].

[3] The DARPA Agent Markup Language is a set of extensions to Extensible Markup Language (XML) and the Resource Description Framework to support ontologies (statements of relationships between objects) for web objects. See www.daml.org.

data-base of facts—it is a construct that embodies cognitive tools and learning

7. Acknowledgements

This research was supported in part by the U.S. Defense Advanced Research Projects Agency under contracts F30602-00-2-0553 and F30602-00-C-0087 (this document is approved for public release, distribution unlimited). The authors thank the SIGCOMM reviewers and the many participants in the DARPA Knowledge Plane Study for discussions and comments that have contributed greatly to the development of our perspective.

8. REFERENCES

[1] D.D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *Proc. ACM SIGCOMM '88*, pp. 102-111.

[2] D.S. Isenberg, "The Rise of the Stupid Network," *Computer Telephony*, Aug 1997, pp. 16-26.

[3] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[4] T. Bayes, "An Essay towards solving a Problem in the Doctrine of Chances," *Philosophical Trans. Royal Society of London* 53 (1763), pp. 370-418.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart and D. Terry, "Epidemic Algorithms for Replicated Database Management," *Proc. ACM PODC '87*, pp. 1-12.

[6] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank aggregation methods for the Web," *Proc. 10th Intl. Conference on World Wide Web*, pp. 613-622 (2001).

[7] J. Wescott, *Automated Network Management*, BBN Report No. 5641. BBN Technologies (1984).

[8] L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*. Stanford Digital Library Project (1998).

[9] V. Paxson, J. Mahdavi, A. Adams and M. Mathis, "An Architecture for Large Scale Internet Measurement," *IEEE Communications Magazine* 36 (1998), pp. 48-54.

[10] J. Kleinberg, "The small-world phenomena: an algorithmic perspective," *Proc. 32nd ACM Symp. Theory of Computing* (2000), pp. 163-170.

[11] L. McKnight and J. Bailey, ed. *Internet Economics*. MIT Press (1997).

[12] D.D. Clark, J. Wroclawski, K.R. Sollins, and R. Braden, "Tussle in Cyberspace: Defining Tomorrow's Internet," *Proc. ACM SIGCOMM 2002*, pp. 347-356.

[13] M. Mathis, "Diagnosing Internet Congestion with a Transport Layer Performance Tool," *Proc. INET '96*,

[14] J. Padhye and S. Floyd, "Identifying the TCP Behavior of Web Servers," *Proc. ACM SIGCOMM 2001*.

[15] V.N. Padmanabhan, L. Qiu and H.J. Wang, "Passive Network Tomography Using Bayesian Inference", *Proc. Internet Measurement Workshop 2002*.

[16] R. Davis, H. Shrobe, and P. Szolovits, "What is a Knowledge Representation?" *AI Magazine*, 14(1):17-33 (1993).

[17] S. Hangal and M. Lam, "Tracking down software bugs using automatic anomaly detection," *Proc. International Conference on Software Engineering '02*.

[18] P. Langley and J. E. Laird, "Cognitive Architectures: Research Issues and Challenges". Draft of October 31, 2002.

[19] T. Dietterich and P. Langley, "Machine Learning for Cognitive Networks: Technology Assessment and Research Challenges". Draft of May 11, 2003.

[20] T. Dietterich, "Learning and Reasoning". Unpublished article of May 26, 2003.

[21] V. Kumar, "Algorithms for Constraint Satisfaction Problems: A Survey". *The AI Magazine*, 13, pp. 32-44 (1992).

[22] P. Stone and M. Veloso, "Multiagent Systems: A Survey from a Machine Learning Perspective", A*utonomous Robots*, 8(3):345-383 (2000).

[23] M. Sloman, "Policy Driven Management for Distributed Systems," *Jour. Network and Systems Management*, vol 2, no 4, Dec 1994, pp. 333-360.

[24] R. Chadha, G. Lapiotis, S. Wright, guest eds., "Policy-Based Networking", *IEEE Network* special issue, March/April 2002, Vol. 16 Issue 2.

Doing more with less: Meta-reasoning and meta-learning in humans and machines

Article *in* Current Opinion in Behavioral Sciences · January 2019

DOI: 10.1016/j.cobeha.2019.01.005

6 authors, including:

Falk Lieder
Max Planck Institute for Intelligent Systems
87 PUBLICATIONS **1,038** CITATIONS

SEE PROFILE

Doing more with less: Meta-reasoning and meta-learning in humans and machines

Thomas L. Griffiths[a,*], Frederick Callaway[a], Michael B. Chang[b], Erin Grant[b], Paul M. Krueger[a], Falk Lieder[c]

[a] *Princeton University, USA*
[b] *University of California at Berkeley, USA*
[c] *Max Planck Institute for Intelligent Systems, Tübingen, Germany*

Abstract

Artificial intelligence systems use an increasing amount of computation and data to solve very specific problems. By contrast, human minds solve a wide range of problems using a fixed amount of computation and limited experience. We identify two abilities that we see as crucial to this kind of general intelligence: meta-reasoning (deciding how to allocate computational resources) and meta-learning (modeling the learning environment to make better use of limited data). We summarize the relevant AI literature and relate the resulting ideas to recent work in psychology.

Keywords: artificial intelligence, meta-reasoning, meta-learning

The current trend in artificial intelligence research is to focus on solving more ambitious problems using more computational power and more training data. One recent analysis, presented in a blog post by OpenAI [1], charted breakthroughs in AI as a function of time and computational resources, finding that the path from the first deep networks producing high performance on image classification to recent successes in playing games like Go involved an exponential increase in computational resources. No such trend applies to human cognition: humans have finite computational resources – those that can be carried around inside our heads – and are limited to the data that can be obtained in the course of a lifetime.

*Corresponding author. This research was supported by contract number FA9550-18-1-0077 from the Air Force Office of Scientific Research and by contract number FA8650-18-2-7832 from the Defence Advanced Research Projects Agency.

In this paper, we will argue that the constraints that characterize human cognition are also intrinsic to developing key components of what we recognize as intelligence [2]. Since the human mind has only a limited amount of computational resources, it has to allocate them in an adaptive manner to solve complex problems efficiently. Having finite time in which to learn means that humans have to be able to make the most of each piece of data available to them, building and then drawing upon a rich model of the world in which learning takes place. As a consequence of developing these abilities, humans become efficient, general-purpose learners – a strong contrast to current AI systems that require huge amounts of training data and are highly specialized to particular tasks.

These two components of human intelligence – efficient use of computational resources [3, 4] and efficient use of data – relate to two problems that have been studied in the AI literature: meta-reasoning and meta-learning. In the next two sections of this paper we review recent progress in these two areas, highlighting how it relates to human cognition. In the final section we consider how this approach can lead to more human-like AI systems: systems that do more with less.

Meta-reasoning

The term "meta-reasoning" contains within it the solution to the problem of how to efficiently deploy computational resources: meta-reasoning is reasoning about reasoning, which means making intelligent decisions about how to think [5]. In research on artificial intelligence, meta-reasoning plays a central role in the definition and design of rational agents that can operate on performance-limited hardware and interact with their environment in real-time [2, 6, 7, 8, 4]. Considering this problem led to the notion of "metalevel rationality" [7], under which the rationality of an agent is not assessed by the expected utility of the actions they take, but by how well the algorithm they follow in order to select those actions trades off expected utility with the costs of taking more time and expending more computation before acting. From this perspective, rationality is not just about making good decisions and drawing good inferences, but also about employing efficient cognitive strategies. Subsequent work refined the notion of metalevel rationality into the concept of "bounded optimality" that also takes into account the computational constraints on metareasoning itself [9, 7]. Bounded optimality defines an optimal program for an agent with performance-limited

hardware that has to interact with its environment in real time as the solution to a constrained optimization problem over the space of programs that the hardware can execute. Here, the program's performance is measured by the utility of the sequence of world states that would result from letting the program run on the agent's hardware as it interacts with its environment.

Having defined this criterion for assessing the rationality of resource-bounded agents, the question arises of how agents can achieve it. Rational meta-reasoning [6] provides one solution: formulated in these terms, the problem of deciding how to think can itself be expressed using the language of statistical decision theory, and many of the familiar tools of AI can be applied to it [10, 11]. Formally, rational meta-reasoning reduces to the problem of computing the value of computation (VOC) for each computation c that could be executed. The VOC is the difference between the increase in expected utility that would be gained by executing computation c and the cost that would be incurred by doing so. In the simplest case, when at most one step of computation can be performed, this can be expressed as

$$\mathrm{VOC}(c, b) = \mathbb{E}_{p(b'|b,c)}\left[\max_{a'} \mathbb{E}\left[U(a')|b'\right] - \max_{a} \mathbb{E}\left[U(a)|b\right]\right] - \mathrm{cost}(c)$$

where b is the agent's current belief, b' is the refined belief resulting from executing computation c, and $\mathbb{E}\left[U(a)|b\right]$ is the expected utility of taking action a over the distribution of outcomes corresponding to belief b.

The rational agent should pursue the computation with the highest VOC, or, if no computation has positive VOC, not pursue any computation at all. The challenge here is that computing the VOC itself would be extremely costly if it required executing each computation to determine the resulting action. Consequently, research has focused on how to efficiently approximate the VOC or identify special cases where it can be computed more easily [2, 9, 10, 11, 12, 13, 14, 15, 6, 16, 17, 18, 19].

We see meta-reasoning as a key component of human intelligence, with the potential to explain a wide range of aspects of human cognition and to shed light on the factors that differentiate human minds from current AI systems. Previous work on human metacognition (e.g., [20]) and active learning (e.g., [21, 22]) have explored aspects of human cognition relevant to meta-reasoning – namely awareness of one's own internal states such as the accuracy of a memory or confidence in a judgment, and reasoning intelligently about how to gather information. However, meta-reasoning is a far more general phenomenon, characterizing the process of selecting or discovering

the cognitive processes that will be used to tackle a task.

Part of what makes people smart is their capacity to make decisions about how they make decisions [23, 24]. This makes decision-making a natural starting point for studying human meta-reasoning. An extensive literature has suggested that people make decisions by following heuristics [25]. A heuristic in itself could constitute a bounded optimal strategy, finding a good trade-off between accuracy and the time required to make a decision. However, since human decisions take place in a variety of environments and the relative cost of errors and time vary across those environments, it makes more sense to have a corresponding variety of heuristics and decide intelligently which strategy to follow in which situation [26]. Previous research has explored human strategy selection, identifying how the strategies that people use vary based on the task and suggesting relatively simple schemes for selecting strategies based on ideas from reinforcement learning [27, 28]. Formulated as a meta-reasoning problem, strategy selection becomes a matter of choosing the strategy that has highest VOC. Lieder and Griffiths [26] showed that taking this approach, assuming that people develop simple internal models of the accuracy and time cost associated with applying different strategies to different problems, better captured human performance than previous psychological models of strategy selection.

Being able to select between existing strategies is only one aspect of what makes people able to adapt to different problems. More generally, people need to be able to efficiently deploy their cognitive resources in situations that they have never encountered before – to discover new strategies. This is a much more challenging meta-reasoning problem. However, in many situations it is possible to use a simplified version of the problem to make progress in strategy discovery. The key observation is that the construction of a strategy often reduces to a sequence of decisions about which computation to perform next [15]. Each computation reveals some piece of information or reduces uncertainty to some extent, providing data that can be used in selecting the next computation. In this case, strategy discovery reduces to a sequential decision problem, and can be expressed as a meta-level Markov decision process (MDP; see Figure 1) [10]. The MDP is a standard way of formulating reinforcement learning problems, and as a consequence we can leverage powerful tools from that literature and apply them to meta-reasoning [13, 15].

One application of this approach is deriving optimal strategies for decision-making in environments where just gathering information about the options is costly. For example, a significant amount of research on decision-making

has used the Mouselab task illustrated in Figure 1(b). In this paradigm people are presented with a set of gambles to choose between but can only reveal the properties of those gambles by moving their computer mouse to particular locations on the screen. This task is supposed to externalize the process of attending to different pieces of information when making decisions, providing a way for psychologists to identify the strategies that people are following. Previous work using this approach has typically focused on strategies identified by the researchers, examining whether people change their strategies as the parameters of the task are varied. However, this task can be formulated as a strategy discovery problem and (approximately) solved using methods for solving MDPs. This solution provides a rational explanation for previously proposed heuristics such as Take-The-Best and Satisficing; but it also predicts novel combinations and extensions of these hand-generated heuristics, some of which people were found to actually use [29, 15]. The result is a far richer picture of the kinds of strategies that can and should be used, and a more accurate characterization of the strategies that people actually follow.

The meta-reasoning approach to strategy discovery has potential applications that range far beyond decision-making. For example, developing an adaptive planning strategy – deciding which paths to explore in a multi-step decision or problem-solving task – can be expressed in these terms, allowing us to identify optimal planning algorithms that can be compared against existing cognitive models and human behavior [30]. The computation of the VOC at the heart of rational meta-reasoning also has close parallels with ideas that have been proposed in the neuroscience literature on cognitive control [31, 32]. More generally, the rational meta-reasoning framework provides us with a set of tools for exploring how people identify the strategies or algorithms that guide their behavior in any cognitive task, from the choices that they make about how to manage their memory to choices about where to deploy their attention.

Meta-learning

While meta-reasoning focuses on the efficient use of cognitive resources, a related challenge is the efficient use of data. One of the most striking aspects of human learning is the ability to learn new words or concepts from limited numbers of examples [33]. This ability contrasts strongly with traditional machine learning methods, which typically require many labeled examples in

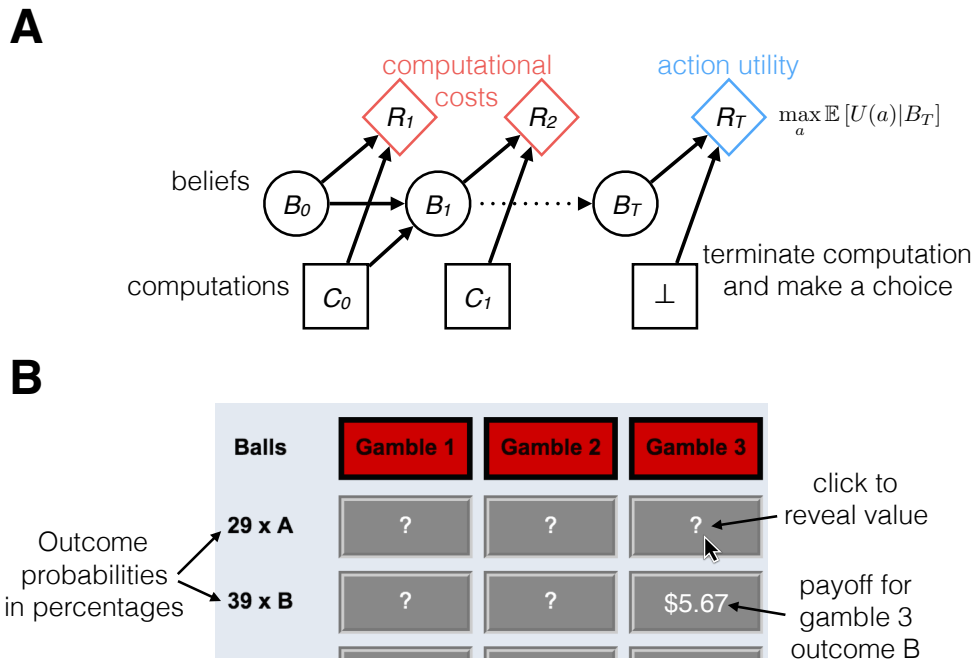


Figure 1: **Meta-reasoning**. (a) Illustration of a meta-level Markov decision process. The initial meta-level rewards capture the cost of computation and the final meta-level reward captures the benefits of computation by the expected object-level reward for choosing an action based on the final belief state. (b) Illustration of the Mouselab paradigm. The Mouselab paradigm externalizes computations by clicks, belief states by revealed information, and the cost of each computation by the fee charged for the corresponding click.

order to even begin to classify stimuli accurately. Recent work in machine learning has aimed to narrow this gap, exploring problems of "few-shot" or "one-shot" learning [34].

One of the frameworks that has been most effective in developing machine learning systems capable of solving these problems is "meta-learning" [35, 36]. In meta-learning, the learner is not presented with a single task – such as learning a particular concept – but with many tasks that all have a similar character. The system aims to leverage commonalities across these tasks in order not only to become better at solving each individual task but also to solve future tasks better and more quickly, effectively "learning to learn."

Recent approaches to meta-learning operate by estimating a single set of hyperparameters that parameterize a task-general component, such as a metric space [34], a memory-augmented neural network [39], or a recurrent neural network [40, 41], that is shared across all tasks. In one particular approach, gradient-based meta-learning, learners are adapted for better performance on each specific task using an optimization algorithm such as gradient descent [42, 43]. Meta-learning is implemented by also learning the parameterization of the learners that adapt to each task. The goal is to find a set of parameters that work well across all of the different tasks so that the learners start with a bias that allows them to perform well despite receiving only a small amount of task-specific data.

Gradient-based meta-learning is an approach with a long history [35, 44, 45] that has flourished due to the recent success of deep learning systems, which make use of large artificial neural networks with parameters trained by gradient descent. In cognitive science, models of cognition based on artificial neural networks are often presented as an alternative to Bayesian models, which explain human judgments as the result of rational statistical inference. It may thus come as a surprise that gradient-based meta-learning has a close relationship with one of the key components of Bayesian models of cognition, hierarchical Bayesian inference.

Hierarchical Bayesian inference captures the idea that learning should take place at multiple levels of abstraction. This idea has been widely used in Bayesian models of cognition. For example, hierarchical Bayesian inference can be used to learn about the properties of objects that words tend to label (such as shape) at the same time as learning the meaning of individual words [46], and to learn about the kinds of causal relationships that exist at the same time as learning those relationships [47]. While Bayesian inference generically indicates how a learner should combine data with a prior

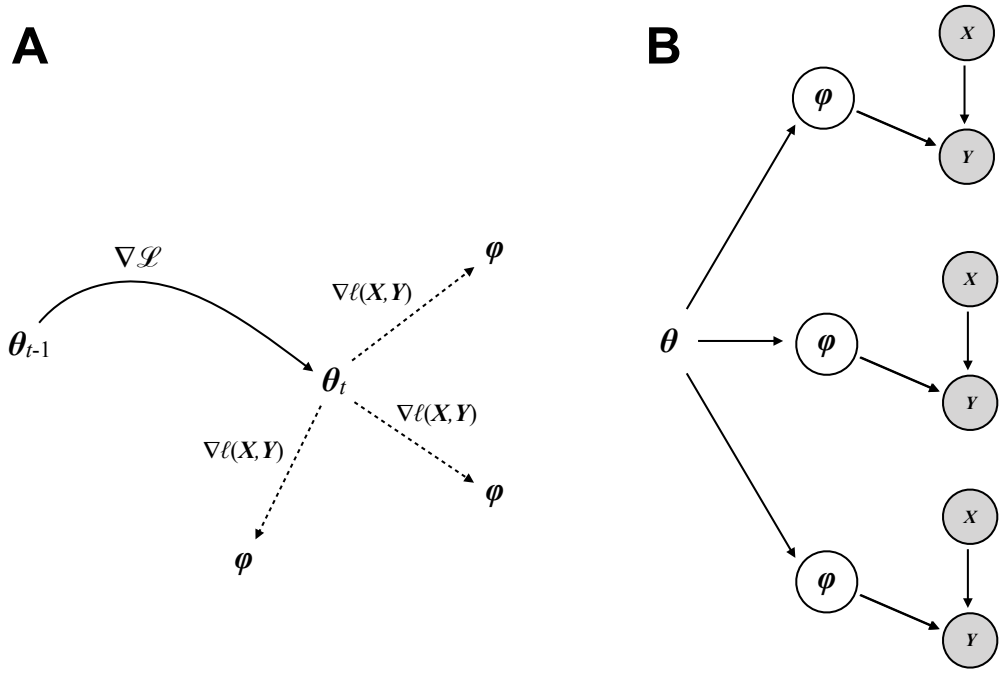


Figure 2: **Gradient-based meta-learning**. (a) The model-agnostic meta-learning (MAML) [37] algorithm optimizes the parameters θ of a set of models so that when one or a few gradient descent steps are taken from the initialization at θ using a small sample of task data (X, Y) to compute the negative log-likelihood $\ell(X, Y)$, each model obtains new parameters ϕ that result in good generalization performance on another sample of data from the same task as measured by the marginal negative log-likelihood $\mathscr{L}$. (b) The probabilistic graphical model for which MAML provides a parameter estimation procedure [38]. Each task-specific parameter ϕ is distinct from but influences the estimation of the others through the parameters of a prior θ shared across all ϕ.

distribution over hypotheses, a hierarchical Bayesian model learns that prior distribution through experience.

There is clearly a loose analogy between hierarchical Bayesian inference and any approach to meta-learning: a meta-learning algorithm tries to establish an inductive bias for new tasks from old tasks, equivalent to "learning a prior." Grant et al. [38] showed that this analogy is, in fact, exact for a prominent form of gradient-based meta-learning, model-agnostic meta-learning (MAML; see Figure 2) [37]. The key idea is that the few steps of gradient descent performed by the task-specific learners results in an approximation to the Bayesian estimate of the parameter values for that task with a prior that depends on the initial parameterization, so learning the initial parameterization is equivalent to learning a prior.

Since much of the previous literature on learning-to-learn in cognitive science has focused on hierarchical Bayesian models, this connection provides a way to translate insights from modeling human learning into contemporary machine learning systems. It also opens the door to defining models using the rich and expressive language of probabilistic generative models that are able to take advantage of the large-scale learning algorithms used in deep learning. As a consequence, it may be possible to develop models that are able to capture the process of learning to learn at a resolution and scale that comes closer to that of human cognition.

Towards general intelligence

Meta-reasoning and meta-learning individually capture key components of human intelligence, but it may be by combining them that we come closest to identifying what is necessary to achieve the kind of general intelligence that currently eludes AI systems. While meta-learning has begun to explore problems that involve learning to perform many different tasks in parallel, those tasks remain relatively similar to one another. By contrast, what characterizes human environments is the need to perform many different tasks that are genuinely different from one another: the same system drives cars, plays chess, loads the dishwasher, looks after small children, creates art, and writes scientific papers. In order to make AI systems that demonstrate the same kind of general intelligence, we need to think about how to train a single system on sets of tasks that display the same diversity.

Part of what allows humans to solve this broad range of problems is the capacity to intelligently reuse elements of their cognitive and motor skills

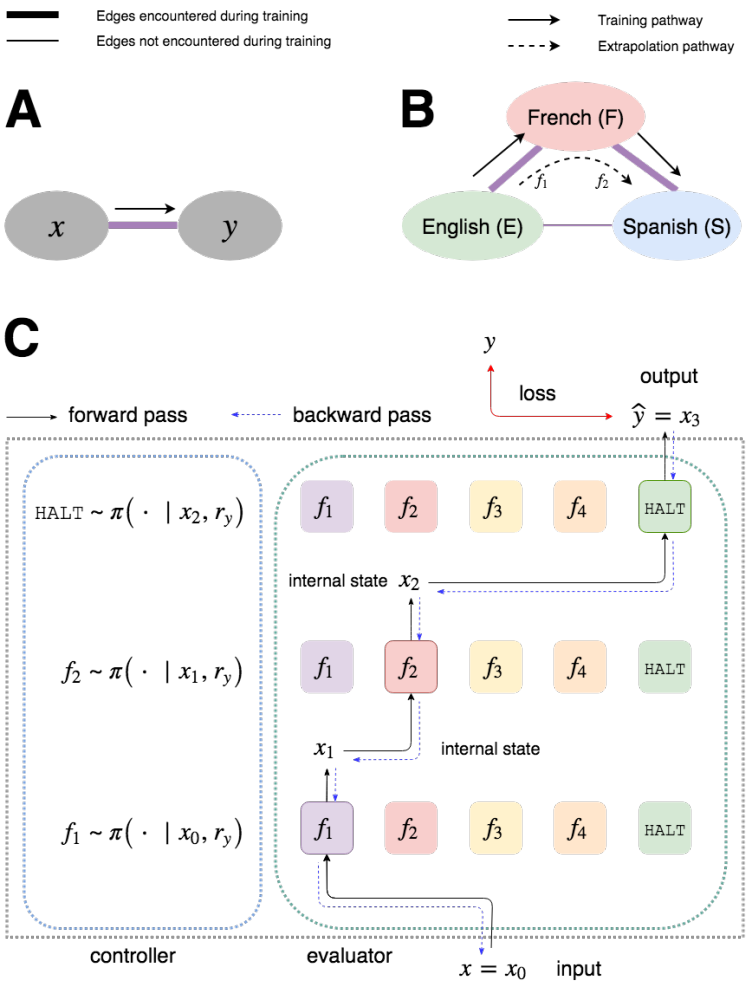


Figure 3: **Compositional Recursive Learner**. (a) In the standard supervised learning problem, the learner receives supervision for mapping an instance x from one distribution r_x to its corresponding instance y in another distribution r_y. (b) In the extrapolation problem, the learner receives only supervision for certain mappings but not for others. For example, suppose that during training the learner receives supervision for learning to translate English to French and to translate French to Spanish. A compositional recursive learner attempts to distill out primitive modules that transform English to French (f_1) and French to Spanish (f_2) and compose them to extrapolate to English-to-Spanish problems. (c) The compositional recursive learner consists of a controller π, an evaluator, and a set of reusable computational modules f_k. Its goal is to transform its input x_0 into a target representation r_y by composing together learned modules. At step j, the controller observes the internal state x_j and the target representation r_y and selects a module f_k. The evaluator applies f_k to transform x_j into a new internal state x_{j+1}. When π selects `HALT`, a loss is computed by comparing the current internal state with the desired output. The loss is backpropagated through the modules, and the controller is trained with policy optimization algorithms.

when they encounter a new problem [48]. To translate this into the language of deep learning, we are making intelligent decisions about how to deploy learned neural modules dynamically, fluidly constructing new network architectures out of old components. This approach has elements of both meta-reasoning and meta-learning: we learn to perform well across different tasks by efficiently allocating cognitive resources. Recent work exploring this approach has shown that it can be used to automatically construct neural network architectures for novel problems in a way that supports far broader generalization than traditional neural network learning algorithms [49].

Figure 3 shows how reasoning about what modules to deploy to solve a problem supports a distinctive form of generalization. The key idea is that such a system can solve problems that are *composed* of previously-solved subproblems, in the formal sense of function composition. For example, a system that has learned to translate English to French and French to Spanish can automatically translate English to Spanish by first translating to French. This supports long-range generalization beyond the tasks that the system has been trained upon – much like what we see in human behavior.

We anticipate that learning how to do more with less will become an increasingly important aspect of AI as researchers begin to hit the limits of their computational and data resources, and that hitting those limits will paradoxically result in systems that more closely resemble human cognition. This convergence could be accelerated by including computational constraints into the definition of the benchmark problems that drive the development of machine learning and artificial intelligence.

References

[1] D. Amodei, D. Hernandez, AI and compute, 2018. http://blog.openai.com/ai-and-compute/.

[2] S. J. Russell, E. Wefald, Do the right thing: studies in limited rationality, MIT Press, Cambridge, MA, 1991.

[3] T. L. Griffiths, F. Lieder, N. D. Goodman, Rational use of cognitive resources: Levels of analysis between the computational and the algorithmic, Topics in cognitive science 7 (2015) 217–229.

[4] S. J. Gershman, E. J. Horvitz, J. B. Tenenbaum, Computational ra-

tionality: A converging paradigm for intelligence in brains, minds, and machines, Science 349 (2015) 273–278.

[5] M. T. Cox, A. Raja, Metareasoning: Thinking about thinking, MIT Press, Cambridge, MA, 2011.

[6] S. J. Russell, E. Wefald, Principles of metareasoning, Artificial intelligence 49 (1991) 361–395.

[7] S. J. Russell, Rationality and intelligence, Artificial intelligence 94 (1997) 57–77.

[8] E. J. Horvitz, S. Zilberstein, Computational tradeoffs under bounded resources, Artificial Intelligence 126 (2001) 1–4.

[9] S. J. Russell, D. Subramanian, Provably bounded-optimal agents, Journal of Artificial Intelligence Research 2 (1994) 575–609.

[10] N. Hay, S. J. Russell, D. Tolpin, S. Shimony, Selecting computations: Theory and applications, in: N. de Freitas, K. Murphy (Eds.), Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence, AUAI Press, Corvallis, OR, 2012.

[11] F. Lieder, D. Plunkett, J. B. Hamrick, S. J. Russell, N. Hay, T. L. Griffiths, Algorithm selection by rational metareasoning as a model of human strategy selection, in: Advances in neural information processing systems, 2014, pp. 2870–2878.

[12] S. Milli, F. Lieder, T. L. Griffiths, When does bounded-optimal metareasoning favor few cognitive systems?, in: S. P. Singh, S. Markovitch (Eds.), Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, volume 31, AAAI Press, Palo Alto, CA, 2017.

[13] F. Callaway, S. Gul, P. Kruegera, T. L. Griffithsa, F. Lieder, Learning to select computations, in: Uncertainty in Artificial Intelligence: Proceedings of the Thirty-Fourth Conference, 2018.

[14] C. H. Lin, A. Kolobov, E. Kamar, E. Horvitz, Metareasoning for planning under uncertainty, in: Proceedings of the 24th International Conference on Artificial Intelligence, AAAI Press, 2015, pp. 1601–1609.

[15] F. Lieder, P. M. Krueger, T. L. Griffiths, An automatic method for discovering rational heuristics for risky choice., in: Proceedings of the 39th Annual Meeting of the Cognitive Science Society (CogSci), 2017.

* This paper formalizes the problem of discovering rational heuristics as solving a meta-level Markov decision process.

[16] E. Vul, N. Goodman, T. L. Griffiths, J. B. Tenenbaum, One and done? Optimal decisions from very few samples, Cognitive science 38 (2014) 599–637.

[17] S. J. Russell, Rationality and intelligence: A brief update, in: V. C. Müller (Ed.), Fundamental issues of artificial intelligence, Springer, 2016, pp. 7–28.

** A survey of recent progress in artificial intelligence research on meta-reasoning and bounded optimality that clarifies the differences between perfect rationality, calculative rationality, meta-level rationality, and bounded optimality.

[18] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: European conference on machine learning, Springer, 2006, pp. 282–293.

[19] F. Lieder, T. Griffiths, N. Goodman, Burn-in, bias, and the rationality of anchoring, in: Advances in neural information processing systems 25, 2012, pp. 2690–2798.

[20] J. Dunlosky, J. Metcalfe, Metacognition, Sage Publications, Thousand Oaks, CA, 2008.

[21] M. Oaksford, N. Chater, A rational analysis of the selection task as optimal data selection., Psychological Review 101 (1994) 608–631.

[22] T. M. Gureckis, D. B. Markant, Self-directed learning: A cognitive and computational perspective, Perspectives on Psychological Science 7 (2012) 464–481.

[23] J. W. Payne, J. R. Bettman, E. J. Johnson, The adaptive decision maker, Cambridge University Press, 1993.

[24] Y.-L. Boureau, P. Sokol-Hessner, N. D. Daw, Deciding how to decide: Self-control and meta-decision making, Trends in cognitive sciences 19 (2015) 700–710.

[25] G. Gigerenzer, W. Gaissmaier, Heuristic decision making, Annual review of psychology 62 (2011) 451–482.

[26] F. Lieder, T. L. Griffiths, Strategy selection as rational metareasoning, Psychological Review 124 (2017) 762–794.

** An application of rational reasoning to modeling people's capacity for adaptive strategy selection and how it is acquired through learning.

[27] J. Rieskamp, P. E. Otto, SSL: a theory of how people learn to select strategies., Journal of Experimental Psychology: General 135 (2006) 207.

[28] I. Erev, G. Barron, On adaptation, maximization, and reinforcement learning among cognitive strategies., Psychological review 112 (2005) 912.

[29] S. Gul, P. M. Krueger, F. Callaway, T. L. Griffiths, F. Lieder, Discovering rational heuristics for risky choice, in: The 14th biannual conference of the German Society for Cognitive Science, GK, 2018.

[30] F. Callaway, F. Lieder, P. Das, S. Gul, P. M. Krueger, T. L. Griffiths, A resource-rational analysis of human planning, in: Proceedings of the 40th Annual Conference of the Cognitive Science Society, Cognitive Science Society, Austin, TX, 2018.

* Rational metareasoning is applied to discover optimal planning strategies and the resulting predictions are tested experimentally.

[31] F. Lieder, A. Shenhav, S. Musslick, T. L. Griffiths, Rational metareasoning and the plasticity of cognitive control, PLoS Computational Biology 14 (2018) e1006043.

* This paper links the idea of "value of computation" from the AI literature to an independently proposed idea of "value of control" in cognitive neuroscience.

[32] A. Shenhav, S. Musslick, F. Lieder, W. Kool, T. L. Griffiths, J. Cohen, M. Botvinick, Toward a rational and mechanistic account of mental effort, Annual Review of Neuroscience 40 (2017) 99–124.

[33] B. M. Lake, R. Salakhutdinov, J. B. Tenenbaum, Human-level concept learning through probabilistic program induction, Science 350 (2015) 1332–1338.

[34] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al., Matching networks for one shot learning, in: Advances in Neural Information Processing Systems (NIPS), 2016, pp. 3630–3638.

[35] J. Schmidhuber, Evolutionary principles in self-referential learning, Ph.D. thesis, Institut für Informatik, Technische Universität München, 1987.

[36] S. Thrun, L. Pratt, Learning to learn, Springer Science & Business Media, 2012.

[37] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: Proceeedings of the 34th International Conference on Machine Learning (ICML), 2017.

* The model-agnostic meta-learning (MAML) algorithm learns a parameter initialization for gradient-based optimization of function approximators such as artificial neural networks; the result is a model that is easy to fine-tune on a variety of tasks.

[38] E. Grant, C. Finn, S. Levine, T. Darrell, T. L. Griffiths, Recasting gradient-based meta-learning as hierarchical Bayes, in: Proceeedings of the 6th International Conference on Learning Representations (ICLR), 2018.

** A particular gradient-based approach to meta-learning can be reformulated as a classic method for parameter estimation in a hierarchical Bayesian model (HBM). The connection provides a means for research domains that traditionally make use of HBMs, such as cognitive science, to scale to high-dimensional models such as neural networks and therefore naturalistic stimuli such as images.

[39] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, T. Lillicrap, Meta-learning with memory-augmented neural networks, in: Proceedings of

the 33rd International Conference on Machine Learning (ICML), 2016, pp. 1842–1850.

[40] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, P. Abbeel, RL^2: Fast reinforcement learning via slow reinforcement learning, arXiv preprint arXiv:1611.02779 (2016).

[41] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, M. Botvinick, Learning to reinforcement learn, arXiv preprint arXiv:1611.05763 (2016).

[42] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, N. de Freitas, Learning to learn by gradient descent by gradient descent, in: Advances in Neural Information Processing Systems (NIPS) 29, 2016, pp. 3981–3989.

[43] S. Ravi, H. Larochelle, Optimization as a model for few-shot learning, in: Proceeedings of the 4th International Conference on Learning Representations (ICLR), 2017.

[44] S. Bengio, Y. Bengio, J. Cloutier, J. Gecsei, On the optimization of a synaptic learning rule, in: Proceedings of the Conference on Optimality in Artificial and Biological Neural Networks, 1992, pp. 6–8.

[45] S. Hochreiter, A. Younger, P. Conwell, Learning to learn using gradient descent, Proceedings of the International Conference on Artificial Neural Networks (ICANN) (2001) 87–94.

[46] C. Kemp, A. Perfors, J. B. Tenenbaum, Learning overhypotheses with hierarchical bayesian models, Developmental science 10 (2007) 307–321.

[47] V. K. Mansinghka, C. Kemp, J. B. Tenenbaum, T. L. Griffiths, Structured priors for structure learning, in: Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence, UAI'06, AUAI Press, Arlington, Virginia, United States, 2006, pp. 324–331.

[48] N. A. Taatgen, The nature and transfer of cognitive skills., Psychological review 120 (2013) 439.

[49] M. B. Chang, A. Gupta, S. Levine, T. L. Griffiths, Automatically composing representation transformations as a means for generalization, arXiv preprint arXiv:1807.04640 (2018).

* This paper reframes the extrapolation problem as a problem of learning algorithmic procedures over learned transformations between representations.

[50] R. Ackerman, V. A. Thompson, Meta-reasoning: Monitoring and control of thinking and reasoning, Trends in Cognitive Sciences 21 (2017) 607–617.

** Provides an overview of recent results on human metacognition.

[51] P. M. Krueger, F. Lieder, T. L. Griffiths, Enhancing metacognitive reinforcement learning using reward structures and feedback, in: G. Gunzelmann, A. Howes, T. Tenbrink, E. Davelaar (Eds.), Proceedings of the 39th Annual Meeting of the Cognitive Science Society, Cognitive Science Society, Austin, TX, 2017.

[52] J. X. Wang, Z. Kurth-Nelson, D. Kumaran, D. Tirumala, H. Soyer, J. Z. Leibo, D. Hassabis, M. Botvinick, Prefrontal cortex as a meta-reinforcement learning system, Nature neuroscience 21 (2018) 860.

Definition and Complexity of Some Basic Metareasoning Problems*

Vincent Conitzer and **Tuomas Sandholm**
Carnegie Mellon University
Computer Science Department
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
{conitzer,sandholm}@cs.cmu.edu

Abstract

In most real-world settings, due to limited time or other resources, an agent cannot perform all potentially useful deliberation and information gathering actions. This leads to the metareasoning problem of selecting such actions. Decision-theoretic methods for metareasoning have been studied in AI, but there are few theoretical results on the complexity of metareasoning. We derive hardness results for three settings which most real metareasoning systems would have to encompass as special cases. In the first, the agent has to decide how to allocate its deliberation time across anytime algorithms running on different problem instances. We show this to be $\mathcal{NP}$-complete. In the second, the agent has to (dynamically) allocate its deliberation or information gathering resources across multiple actions that it has to choose among. We show this to be $\mathcal{NP}$-hard even when evaluating each individual action is extremely simple. In the third, the agent has to (dynamically) choose a limited number of deliberation or information gathering actions to disambiguate the state of the world. We show that this is $\mathcal{NP}$-hard under a natural restriction, and $\mathcal{PSPACE}$-hard in general.

1 Introduction

In most real-world settings, due to limited time, an agent cannot perform all potentially useful deliberation actions. As a result it will generally be unable to act rationally in the world. This phenomenon, known as bounded rationality, has been a long-standing research topic (e.g., [3, 17]). Most of that research has been *descriptive*: the goal has been to characterize how agents—in particular, humans—deal with this constraint. Another strand of bounded rationality research has the *normative (prescriptive)* goal of characterizing how agents *should* deal with this constraint. This is particularly important when building artificial agents.

Characterizing how an agent should deal with bounded rationality entails determining how the agent should deliberate. Because limited time (or other resources) prevent the agent from performing all potentially useful deliberation (or information gathering) actions, it has to select among such actions. Reasoning about which deliberation actions to take is called *metareasoning*. Decision theory [7, 10] provides a normative basis for metareasoning under uncertainty, and decision-theoretic deliberation control has been widely studied in AI (e.g., [2, 4–6, 8, 9, 12–15, 18–20]).

However, the approach of using metareasoning to control reasoning is impractical if the metareasoning problem itself is prohibitively complex. While this issue is widely acknowledged (e.g., [8, 12–14]), there are few theoretical results on the complexity of metareasoning.

We derive hardness results for three central metareasoning problems. In the first (Section 2), the agent has to decide how to allocate its deliberation time across anytime algorithms running on different problem instances. We show this to be $\mathcal{NP}$-complete. In the second metareasoning problem (Section 3), the agent has to (dynamically) allocate its deliberation or information gathering resources across multiple actions that it has to choose among. We show this to be $\mathcal{NP}$-hard even when evaluating each individual action is extremely simple. In the third metareasoning problem (Section 4), the agent has to (dynamically) choose a limited number of deliberation or information gathering actions to disambiguate the state of the world. We show that this is $\mathcal{NP}$-hard under a natural restriction, and $\mathcal{PSPACE}$-hard in general.

These results have general applicability in that most metareasoning systems must somehow deal with one or more of these problems (in addition to dealing with other issues). We also believe that these results give a good basic overview of the space of high-complexity issues in metareasoning.

2 Allocating anytime algorithm time across problems

In this section we study the setting where an agent has to allocate its deliberation time across different problems—each of which the agent can solve using an anytime algorithm. We show that this is hard even if the agent can perfectly predict the performance of the anytime algorithms.

2.1 Motivating example

Consider a newspaper company that has, by midnight, received the next day's orders from newspaper stands in the 3

*The material in this paper is based upon work supported by the National Science Foundation under CAREER Award IRI-9703122, Grant IIS-9800994, ITR IIS-0081246, and ITR IIS-0121678.

cities where the newpaper is read. The company owns a fleet of delivery trucks in each of the cities. Each fleet needs its vehicle routing solution by 5am. The company has a default routing solution for each fleet, but can save costs by improving (tailoring to the day's particular orders) the routing solution of any individual fleet using an anytime algorithm. In this setting, the "solution quality" that the anytime algorithm provides on a fleet's problem instance is the amount of savings compared to the default routing solution.

We assume that the company can perfectly predict the savings made on a given fleet's problem instance as a function of deliberation time spent on it (we will prove hardness of metareasoning even in this deterministic variant). Such functions are called *(deterministic) performance profiles* [2, 6, 8, 9, 20]. Each fleet's problem instance has its own performance profile.[1] Suppose the performance profiles are as shown in Fig. 1.

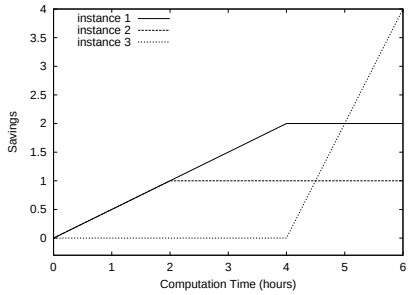


Figure 1: Performance profiles for the routing problems.

Then the maximum savings we can obtain with 5 hours of deliberation time is 2.5, for instance by spending 3 hours on instance 1 and 2 on instance 2. On the other hand, if we had until 6am to deliberate (6 hours), we could obtain a savings of 4 by spending 6 hours on instance 3.

2.2 Definitions and results

We now define the metareasoning problem of allocating deliberation across problems according to performance profiles.

Definition 1 (PERFORMANCE-PROFILES) *We are given a list of performance profiles $(f_1, f_2, \ldots, f_m)$ (where each f_i is a nondecreasing function of deliberation time, mapping to nonnegative real numbers), a number of deliberation steps N, and a target value K. We are asked whether we can distribute the deliberation steps across the problem instances to get a total performance of at least K; that is, whether there exists a vector $(N_1, N_2, \ldots, N_m)$ with $\sum_{1 \le i \le m} N_i \le N$ and $\sum_{1 \le i \le m} f_i(N_i) \ge K$.*

A reasonable approach to representing the performance profiles is to use piecewise linear performance profiles. They can model any performance profile arbitrarily closely, and have been used in the resource-bounded reasoning literature to characterize the performance of anytime algorithms (e.g. [2]). We now show that the metareasoning problem is $\mathcal{NP}$-complete even under this restriction. We will reduce from the KNAPSACK problem.[2]

Definition 2 (KNAPSACK) *We are given a set S of m pairs of positive integers (c_i, v_i), a constraint $C > 0$ and a target value $V > 0$. We are asked whether there exists a set $I \subseteq S$ such that $\sum_{j \in I} C_j \le C$ and $\sum_{j \in I} v_j \ge V$.*

Theorem 1 *PERFORMANCE-PROFILES is $\mathcal{NP}$-complete even if each performance profile is continuous and piecewise linear.*[3]

Proof: The problem is in $\mathcal{NP}$ because we can nondeterministically generate the N_i in polynomial time (since we do not need to bother trying numbers greater than N), and given the N_i, we can verify if the target value is reached in polynomial time. To show $\mathcal{NP}$-hardness, we reduce an arbitrary KNAPSACK instance to the following PERFORMANCE-PROFILES instance. Let there be m performance profiles, given by $f_i(n) = 0$ for $n \le c_i$, $f_i(n) = n - c_i$ for $c_i < n \le c_i + v_i$, and $f_i(n) = v_i$ for $n > c_i + v_i$. Let $N = C + V$ and let $K = V$. We claim the two problem instances are equivalent. First suppose there is a solution to the KNAPSACK instance, that is, a set $I \subseteq S$ such that $\sum_{j \in I} C_j \le C$ and $\sum_{j \in I} v_j \ge V$. Then set the N_i as follows: $N_i = 0$ for $i \notin I$, and $N_i = c_i + \frac{V}{\sum_{j \in I} v_j} v_i$ for $i \in I$. Then, $\sum_{1 \le i \le m} N_i = \sum_{i \in I} N_i = \sum_{i \in I} (c_i + \frac{V}{\sum_{j \in I} v_j} v_i) = \sum_{i \in I} c_i + \frac{V}{\sum_{j \in I} v_j} \sum_{i \in I} v_i = \sum_{i \in I} c_i + V \le C + V = N$ since $\sum_{j \in I} C_j \le C$. Also, since $\sum_{j \in I} v_j \ge V$, it follows that for every $i \in I$, we have $c_i \le N_i \le c_i + v_i$, and hence $\sum_{1 \le i \le m} f_i(N_i) = \sum_{i \in I} f_i(N_i) = \sum_{i \in I} (\frac{V}{\sum_{j \in I} v_j} v_i) =$

[1] Because the anytime algorithm's performance differs across instances, each instance has its own performance profile (in the setting of deterministic performance profiles). In reality, an anytime algorithm's performance on an instance cannot be predicted perfectly. Rather, usually statistical performance profiles are kept that aggregate across instances. In that light one might question the assumption that different instances have different performance profiles. However, sophisticated deliberation control systems can condition the performance prediction on features of the instance—and this is necessary if the deliberation control is to be fully normative. (Research has already been conducted on conditioning performance profiles on instance features [8, 9, 15] or results of deliberation on the instance so far [4, 8, 9, 15, 18–20].)

[2] This only demonstrates *weak* NP-completeness, as KNAPSACK is weakly NP-complete; thus, perhaps pseudopolynomial time algorithms exist.

[3] If one additionally assumes that each performance profile is concave, then the metareasoning problem is solvable in polynomial time [2]. While returns to deliberation indeed tend to be diminishing, usually this is not the case throughout the performance profile. Algorithms often have a setup phase in the beginning during which there is no improvement. Also, iterative improvement algorithms can switch to using different local search operators once progress has ceased using one operator (for example, once 2-swap has reached a local optimum in TSP, one can switch to 3-swap and obtain gains from deliberation again) [16].

$\frac{V}{\sum_{j\in I} v_j} \sum_{i\in I} v_i = V = K$. So we have found a solution to the PERFORMANCE-PROFILES instance. On the other hand, suppose there is a solution to the PERFORMANCE-PROFILES instance, that is, a vector $(N_1, N_2, \ldots, N_m)$ with $\sum_{1\le i\le m} N_i \le N$ and $\sum_{1\le i\le m} f_i(N_i) \ge K$. Since spending more than $c_i + v_i$ deliberation steps on profile i is useless, we may assume that $N_i \le c_i + v_i$ for all i. We now claim that $I = \{i : N_i \ge c_i\}$ is a solution to the KNAPSACK instance. First, using the fact that $f_j(N_j) = 0$ for all $j \notin I$, we have $\sum_{i\in I} v_i \ge \sum_{i\in I} f_i(N_i) = \sum_{1\le i\le m} f_i(N_i) \ge K = V$. Also, $\sum_{i\in I} c_i = \sum_{i\in I} N_i - \sum_{i\in I}(N_i - c_i) = \sum_{i\in I} N_i - \sum_{i\in I} f_i(N_i) \le \sum_{1\le i\le m} N_i - \sum_{1\le i\le m} f_i(N_i) \le N - K = C + V - V = C$. So we have found a solution to the KNAPSACK instance. ∎

The PERFORMANCE-PROFILES problem occurs naturally as a subproblem within many metareasoning problems, and thus its complexity leads to significant difficulties for metareasoning. This is the case even under the (unrealistic) assumption of perfect predictability of the efficacy of deliberation. On the other hand, in the remaining two metareasoning problems that we analyze, the complexity stems from uncertainty about the results that deliberation will provide.

3 Dynamically allocating evaluation effort across options (actions)

In this section we study the setting where an agent is faced with multiple options (actions) from which it eventually has to choose one. The agent can use deliberation (or information gathering) to evaluate each action. Given limited time, it has to decide which ones to evaluate. We show that this is hard even in very restricted cases.

3.1 Motivating example

Consider an autonomous robot looking for precious metals. It can choose between three sites for digging (it can dig at most one site). At site A it may find gold; at site B, silver; at site C, copper. If the robot chooses not to dig anywhere, it gets utility 1 (for saving digging costs). If the robot chooses to dig somewhere, the utility of finding nothing is 0; finding gold, 5; finding silver, 3; finding copper, 2. The prior probability of there being gold at site A is $\frac{1}{8}$, that of finding silver at site B is $\frac{1}{2}$, and that of finding copper at site C is $\frac{1}{2}$.

In general, the robot could perform deliberation or information gathering actions to evaluate the alternative (digging) actions. The metareasoning problem would be the same for both, so for simplicity of exposition, we will focus on information gathering only. Specifically, the robot can perform tests to better evaluate the likelihood of there being a precious metal at each site, but it has only limited time for such tests. The tests are the following: (1) Test for gold at A. If there is gold, the test will be positive with probability $\frac{14}{15}$; if there is no gold, the test will be positive with probability $\frac{1}{15}$. This test takes 2 units of time. (2) Test for silver at B. If there is silver, the test will be positive with probability 1; if there is no silver, the test will be positive with probability 0. This test takes 3 units of time. (3) Test for copper at C. If there is copper, the test will be positive with probability 1;if there is no copper, the test will be positive with probability 0. This test takes 2 units of time.

Given the probabilities of the tests turning out positive under various circumstances, one can use Bayes' rule to compute the expected utility of each digging option given any (lack of) test result. For instance, letting a be the event that there is gold at A, and t_A be the event that the test at A is positive, we observe that $P(t_A) = P(t_A|a)P(a) + P(t_A| - a)P(-a) = \frac{14}{15}\frac{1}{8} + \frac{1}{15}\frac{7}{8} = \frac{7}{40}$. Then, the expected utility of digging at A given that the test at A was positive is $5\,P(a|t_A)$, where $P(a|t_A) = \frac{P(t_A|a)P(a)}{P(t_A)} = \frac{\frac{14}{15}\frac{1}{8}}{\frac{7}{40}} = \frac{2}{3}$, so the expected utility is $\frac{10}{3}$. Doing a similar analysis everywhere, we can represent the problem by trees shown in Fig. 2. In these trees, being at the root represents not having done a test yet, whereas being at a left (right) leaf represents the test having turned out positive (negative); the value at each node is the expected value of digging at this site given the information corresponding to that node. The values on the edges are the probabilities of the test turning out positive or negative. We can subsequently use these trees for analyzing how we should gather information. For instance, if we have 5 units of time, the optimal information gathering policy is to test at B first; if the result is positive, test at A; otherwise test at C. (We omit the proof because of space constraint.)

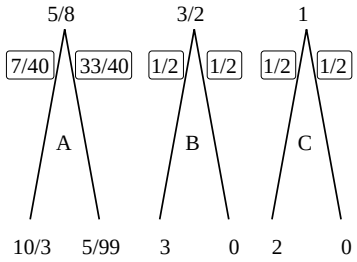


Figure 2: Tree representation of the action evaluation instance.

3.2 Definitions

In the example, there were four *actions* that we could evaluate: digging for a precious metal at one of three locations, or not digging at all. Given the results of all the tests that we might undertake on a given action, executing it has some expected value. If, on the other hand, we do not (yet) know all the results of these tests, we can still associate an expected value with the action by taking an additional expectation over the outcomes of the tests. In what follows, we will drop the word "expected" in its former meaning (that is, when talking about the expected value given the outcomes of all the tests), because the probabilistic process regarding this expectation has no relevance to how the agent should choose to test. Hence, all expectations are over the outcomes of the tests.

While we have presented this as a model for information gathering planning, we can use this as a model for planning

(computational) deliberation over multiple actions as well. In this case, we regard the tests as computational steps that the agent can take toward evaluating an action.[4]

To proceed, we need a formal model of how evaluation effort (information gathering or deliberation) invested on a given action changes the agent's beliefs about that action. For this model, we generalize the above example to the case where we can take multiple evaluation steps on a certain action (although we will later show hardness even when we can take at most one evaluation step per action).

Definition 3 *An* action evaluation tree *is a tree with*

- *A root r, representing the start of the evaluation;*
- *For each nonleaf node w, a cost k_w for investing another step of evaluation effort at this point;*
- *For each edge e between parent node p and child node c, a probability $p_e = p_{(p,c)}$ of transitioning from p to c upon taking a step of evaluation effort at p;*
- *For each leaf node l, a value u_l.*

According to this definition, at each point in the evaluation of a single action, the agent's only choice is *whether* to invest further evaluation effort, but not *how* to continue the evaluation. This is a reasonable model when the agent does evaluation through deliberation and has one algorithm at its disposal. However, in general the agent may have different information gathering actions to choose from at a given point in the evaluation, or may be able to choose from among several deliberation actions (e.g., via search control [1, 14]). In Section 4, we will discuss how being able to choose between tests may introduce drastic complexity even when evaluating a single thing. In this section, however, our focus is on the complexities introduced by having to choose between different actions on which to invest evaluation effort next.

The agent can determine its expected value of an action, given its evaluation so far, using the subtree of the action evaluation tree that is rooted at the node where evaluation has brought us so far. This value can be determined in that subtree by propagating upwards from the leafs: for parent p with a set of children C, we have $u_p = \sum_{c \in C} (p_{(p,c)} u_c)$.

We now present the metareasoning problem. In general, the agent could use an *online* evaluation control policy where the choices of how to invest future evaluation effort can depend on evaluation results obtained so far. However, to avoid trivial complexity issues introduced by the fact that such a *contingency* strategy for evaluation can be exponential in size, we merely ask what action the agent should invest its first evaluation step on.

Definition 4 (ACTION-EVALUATION) *We are given l action evaluation trees, indexed 1 through l, corresponding to l different actions. (The transition processes of the trees are independent.) Additionally, we are given an integer N. We are asked whether, among the online evaluation control policies that spend at most N units of effort, there exists one that takes its first evaluation step on action 1, and gives maximal expected utility among online evaluation control policies that spend at most N units of effort. (If at the end of the deliberation process, we are at node w_i for tree i, then our utility is $\max_{1 \le i \le m} u_{w_i}$, because we will choose the action with the highest expected value.)*

[4]For this to be a useful model, it is necessary that updating beliefs about the value of an action (after taking a deliberation step) is computationally easy relative to the evaluation problem itself.

3.3 Results

We now show that even a severely restricted version of this problem is $\mathcal{NP}$-hard.[5]

Theorem 2 *ACTION-EVALUATION is $\mathcal{NP}$-hard, even when all trees have depth either 0 or 1, branching factor 2, and all leaf values are -1, 0, or 1.*

Proof: If action evaluation tree j has depth 1 and branching factor 2, we represent it by $(p_1^j, p_2^j, u_1^j, u_2^j, k^j)$ where p_i^j is the transition probability to leaf i, u_i^j is the value at leaf i, and k^j is the cost of taking the (only) step of evaluation. We reduce an arbitrary KNAPSACK instance to the following ACTION-EVALUATION instance. Let $l = m + 3$, let $\delta = \frac{1}{16m(\sum_{1 \le i \le m} v_i)^2}$, and let $\epsilon = 2\delta \sum_{1 \le i \le m} v_i = \frac{1}{8m \sum_{1 \le i \le m} v_i}$. We set tree 1 $(p_1^1, p_2^1, u_1^1, u_2^1, k^1) = (\epsilon, 1-\epsilon, 1, -1, 1)$, and tree 2 $(p_1^2, p_2^2, u_1^2, u_2^2, k^1) = ((1-\epsilon)^m(\epsilon + \delta V), 1 - (1-\epsilon)^m(\epsilon + \delta V), 1, -1, C+1)$[6]. Tree 3 has depth 0 and a value of 0. Finally, for each pair (c_i, v_i) in the KNAPSACK instance, there is an evaluation tree $(p_1^{i+3}, p_2^{i+3}, u_1^{i+3}, u_2^{i+3}, k^{i+3}) = (\delta v_i, 1 - \delta v_i, 1, -1, c_i)$. We set $N = C + 1$. We claim the instances are equivalent. First we make some observations about the constructed ACTION-EVALUATION instance. First, once we determine the value of a action to be 1, choosing this action is certainly optimal regardless of the rest of the deliberation process. Second, if at the end of the deliberation process we have not discovered the value of any action to be 1, then for any of the trees of depth 1, either we have discovered the corresponding action's value to be -1, or we have done no deliberation on it at all. In the latter case, the expected value of the action is always below 0 (δ is carefully set to achieve this). Hence, we will pick action 3 for value 0. It follows that an optimal deliberation policy is one that maximizes the probability of discovering that a action has value 1. Now, consider the *test set* of a policy, which is the set of actions that the policy would evaluate if no action turned out to have value 1. Then, the probability of discovering that a action has value 1 is simply equal to the probability that at least one of the actions in this set has value 1. So, in this case, the quality of a policy is determined by its test set. Now we observe that any optimal action is either the one that only evaluates action 2 (and then runs out of deliberation time), or one that has action 1 in its

[5]ACTION-EVALUATION is trivial for $l = 1$: the answer is "yes" if it is possible to take a step of evaluation. The same is true if there is no uncertainty with regard to the value of any action; in that case any evaluation is irrelevant.

[6]Note that using m in the exponent does not make the reduction exponential in size, because the length of the binary representation of numbers with m in the exponent is linear in m.

test set. (For consider any other policy; since evaluating action 1 has minimal cost, and gives strictly higher probability of discovering a action with value 1 than evaluating on any other action besides 2, simply replacing any other action in the test set with action 1 is possible and improves the policy.) Now suppose there is a solution to the KNAPSACK instance, that is, a set $I \subseteq S$ such that $\sum_{i \in I} C_i \leq C$ and $\sum_{i \in I} v_i \geq V$. Then we can construct a policy which has as test set $J = \{1\} \bigcup \{j : j - 3 \in I\}$. (Evaluating all these actions costs at most $C + 1$ deliberation units.) The probability of at least one of these actions having value 1 is at least the probability that exactly one of them has value 1, which is $\sum_{j \in J} p_1^j \prod_{k \in J, k \neq j} p_2^k \geq \sum_{j \in J} p_1^j (1-\epsilon)^m = (1-\epsilon)^m (\epsilon + \sum_{i \in I} \delta v_i) \geq (1-\epsilon)^m (\epsilon + \delta V) = p_1^2$. Using our previous observation we can conclude that there is an optimal action that has action 1 in its test set, and since the order in which we evaluate actions in the test set does not matter, there is an optimal policy which evaluates action 1 first. On the other hand, suppose there is no solution to the KNAPSACK instance. Consider a policy which has 1 in its test set, that is, the test set can be expressed as $J = \{1\} \bigcup \{j : j - 3 \in I\}$ for some set I. Then we must have $\sum_{i \in I} C_i \leq C$, and since there is no solution to the KNAPSACK instance, it follows that $\sum_{i \in I} v_i \leq V - 1$. But the probability that at least one of the actions in the test set has value 1 is at most $\sum_{j \in J} p_1^j = \epsilon + \sum_{i \in I} \delta v_i \leq \epsilon + \delta(V-1) = \epsilon + \delta V - \delta$. On the other hand, $p_1^2 = (1-\epsilon)^m (\epsilon + \delta V) \geq (1 - m\epsilon)(\epsilon + \delta V) \geq \epsilon + \delta V - 2m\epsilon^2$. If we now observe that $2m\epsilon^2 = \frac{1}{32m(\sum_{1 \leq i \leq m} v_i)^2} < \frac{1}{16m(\sum_{1 \leq i \leq m} v_i)^2} = \delta$, it follows that the policy of just evaluating action 2 is strictly better. So, there is no optimal policy which evaluates action 1 first. ∎

We have no proof that the general problem is in $\mathcal{NP}$. It is an interesting open question whether stronger hardness results can be obtained for it. For instance, perhaps the general problem is $\mathcal{PSPACE}$-complete.

4 Dynamically choosing how to disambiguate state

We now move to the setting where the agent has only one thing to evaluate, but can choose the order of deliberation (or information gathering) actions for doing so. In other words, the agent has to decide how to disambiguate its state. We show that this is hard. (We consider this to be the most significant result in the paper.)

4.1 Motivating example

Consider an autonomous robot that has discovered it is on the edge of the floor; there is a gap in front of it. It knows this gap can only be one of three things: a staircase (S), a hole (H), or a canyon (C) (assume a uniform prior distribution over these). The robot would like to continue its exploration beyond the gap. There are three courses of physical action available to the robot: attempt a descent down a staircase, attempt to jump over a hole, or simply walk away. If the gap turns out to be a staircase and the robot descends down it, this gives utility 2. If it turns out to be a hole and the robot jumps over it, this gives utility 1 (discovering new floors is more interesting). If the robot walks away, this gives utility 0 no matter what the gap was. Unfortunately, attempting to jump over a staircase or canyon, or trying to descend into a hole or canyon, has the disastrous consequence of destroying the robot (utility $-\infty$). It follows that if the agent cannot determine with certainty what the gap is, it should walk away.

In order to determine the nature of the gap, the robot can conduct various tests (or *queries*). The tests can determine the answers to the following questions: (1) Am I inside a building? A *yes* answer is consistent only with S; a *no* answer is consistent with S, H, C. (2) If I drop a small item into the gap, do I hear it hit the ground? A *yes* answer is consistent with S, H; a *no* answer is consistent with H, C. (3) Can I walk around the gap? A *yes* answer is consistent with S, H; a *no* answer is consistent with S, H, C.

Assume that if multiple answers to a query are consistent with the true state of the gap, the distributions over such answers are uniform and independent. Note that after a few queries, the set of states consistent with all the answers is the intersection of the sets consistent with the individual answers; once this set has been reduced to one element, the robot knows the state of the gap.

Suppose the agent only has time to run one test. Then, to maximize expected utility, the robot should run test 1, because the other tests give it no chance of learning the state of the gap for certain. Now suppose that the agent has time for two tests. Then the optimal test policy is as follows: run test 2 first; if the answer is *yes*, run test 1 second; otherwise, run test 3 second. (If the true state is S, this is discovered with probability $\frac{1}{2}$; if it is H, this is discovered with probability $\frac{1}{4}$, so total expected utility is $\frac{5}{12}$. Starting with test 1 or test 3 can only give expected utility $\frac{1}{3}$.)

4.2 Definitions

We now define the metareasoning problem of how the agent should dynamically choose queries to ask (deliberation or information gathering actions to take) so as to disambiguate the state of the world. While the illustrative example above was for information gathering actions, the same model applies to deliberation actions for state disambiguation (such as image processing, auditory scene analysis, sensor fusing, etc.).

Definition 5 (STATE-DISAMBIGUATION) *We are given*

- *A set $\Theta = \{\theta_1, \theta_2, \ldots, \theta_r\}$ of possible world states;*[7]
- *A probability function p over Θ;*

[7] If there are two situations that are equivalent from the agent's point of view (the agent's optimal course of action is the same and the utility is the same), then we consider those situations to be one state. Note that two such situations may lead to different answers to the queries. For example, one situation may be that the gap is an indoor staircase, and another situation may be that the gap is an outdoor staircase. These situations are considered to be the same state, but will give different answers to the query "Am I inside?".

- *A utility function $u : \Theta \rightarrow \Re^{\geq 0}$ where $u(\theta_i)$ gives the utility of knowing for certain that the world is in state θ_i at the end of the metareasoning process; (not knowing the state of the world for certain always gives utility 0);*
- *A query set Q, where each $q \in Q$ is a list of subsets of Θ. Each such subset corresponds to an answer to the query, and indicates the states that are consistent with that answer. We require that for each state, at least one of the answers is consistent with it: that is, for any $q = (a_1, a_2, \ldots, a_m)$, we have $\bigcup_{1 \leq j \leq m} a_j = \Theta$. When a query is asked, the answer is chosen (uniformly) randomly by nature from the answers to that query that are consistent with the world's true state (these drawings are independent);*
- *An integer N; A target value G.*

We are asked whether there exists a policy for asking at most N queries that gives expected utility at least G. (Letting $\pi(\theta_t)$ be the probability of identifying the state when it is θ_t, the expected utility is given by $\sum_{1 \leq t \leq r} p(\theta_t)\pi(\theta_t)u(\theta_t)$.) [8]

4.3 Results

Before presenting our $\mathcal{PSPACE}$-hardness result, we will first present a relatively straightforward $\mathcal{NP}$-hardness result for the case where for each query, only one answer is consistent with the state of the world. This situation occurs when the states are so specific as to provide enough information to answer every query. Our reduction is from SET-COVER.

Definition 6 (SET-COVER) *We are given a set S, a collection of subsets $\mathcal{T} = \{T_i \subseteq S\}$, and a positive integer M. We are asked whether any M of these subsets cover S, that is, whether there is a subcollection $\mathcal{U} \subseteq \mathcal{T}$ such that $|\mathcal{U}| = M$ and $\bigcup_{T_i \in \mathcal{U}} = S$.*

Theorem 3 *STATE-DISAMBIGUATION is $\mathcal{NP}$-hard, even when for each state-query pair there is only one consistent answer.*

Proof: We reduce an arbitrary SET-COVER instance to the following STATE-DISAMBIGUATION instance. Let $\Theta = S \bigcup \{b\}$. Let p be uniform. Let $u(b) = 1$ and for any $s \in S$, let $u(s) = 0$. Let $Q = \{(\Theta - T_i, T_i) : T_i \in \mathcal{T}\}$. Let $M = N$ and let $G = \frac{1}{|S|+1}$. We claim the instances are equivalent.

First suppose there is a solution to the SET-COVER instance, that is, a subcollection $\mathcal{U} \subseteq \mathcal{T}$ such that $|\mathcal{U}| = M$ and $\bigcup_{T_i \in \mathcal{U}} = S$. Then our policy for the STATE-DISAMBIGUATION instance is simply to ask the queries corresponding to the elements of U, in whichever order and unconditionally on the answers of the query. If the true state is in S, we will get utility 0 regardless. If the true state is b, each query will eliminate the elements of the corresponding T_i from consideration. Since $\mathcal{U}$ is a set cover, it follows that after all the queries have been asked, all elements of S have been eliminated, and we know that the true state of the world is b, to get utility 1. So the expected utility is $\frac{1}{|S|+1}$, so there is a solution to the STATE-DISAMBIGUATION instance.

On the other hand, suppose there is a solution to the STATE-DISAMBIGUATION instance, that is, a policy for asking at most N queries that gives expected utility at least G. Because given the true state of the world, there is only one answer consistent with it for each query, it follows that the queries that will be asked, and the answers given, follow deterministically from the true state of the world. Since we cannot derive any utility from cases where the true state of the world is not b, it follows that when it is b, we must be able to conclude that this is so in order to get positive expected utility. Consider the queries that the policy will ask in this latter case. Each of these queries will eliminate precisely the corresponding T_i. Since at the end of the deliberation, all the elements of S must have been eliminated, it follows that these T_i in fact cover S. Hence, if we let $\mathcal{U}$ be the collection of these T_i, this is a solution to the SET-COVER instance. ■

We are now ready to present our $\mathcal{PSPACE}$-hardness result. The reduction is from stochastic satisfiability, which is $\mathcal{PSPACE}$-complete [11].

Definition 7 (STOCHASTIC-SAT (SSAT)) *We are given a Boolean formula in conjunctive normal form (with a set of clauses C over variables $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n$). We play the following game with nature: we pick a value for x_1, subsequently nature (randomly) picks a value for y_1, whereupon we pick a value for x_2, after which nature picks a value for y_2, etc., until all variables have a value. We are asked whether there is a policy (contingency plan) for playing this game such that the probability of the formula being eventually satisfied is at least $\frac{1}{2}$.*

Now we can present our $\mathcal{PSPACE}$-hardness result.

Theorem 4 *STATE-DISAMBIGUATION is $\mathcal{PSPACE}$-hard.*

Proof: Let $\Theta = C \bigcup \{b\} \bigcup V$ where V consists of the elements of an upper triangular matrix, that is, $V = \{v_{11}, v_{12}, \ldots, v_{1n}, v_{22}, v_{23}, \ldots, v_{2n}, v_{33}, \ldots\ldots, v_{nn}\}$. p is uniform over this set. u is defined as follows: $u(c) = 0$ for all $c \in C$, $u(b) = 1$, and $u(v_{ij}) = H = 2 \prod_{q \in Q} N_{ans}(q)$ for all $v_{ij} \in V$, where $N_{ans}(q)$ is the number of possible answers to q. The queries are as follows. For every $v_{ij} \in V$, there is a query $q_{ij} = (\{v_{ij}\}, \Theta - \{v_{ij}\})$. Additionally, for each variable x_i there are the following two queries: letting $V_i = \{v_{ij} : j \geq i\}$ (that is, row i in the matrix), and letting $C_z = \{c \in C : z \in c\}$, we have

- $q_{x_i} = (V_i, C, \Theta - V_i - C_{x_i} - C_{y_i}, \Theta - V_i - C_{x_i} - C_{-y_i})$;

[8] There are several natural generalizations of this metareasoning problem, each of which is at least as hard as the basic variant. One allows for positive utilities even if there remains some uncertainty about the state at the end of the disambiguation process. In this more general case, the utility function would have *subsets* of Θ as its domain (or perhaps even probability distributions over such subsets). In general, specifying such utility functions would require space exponential in the number of states, so some restriction of the utility function is likely to be necessary; nevertheless, there are utility functions specifiable in polynomial space that are more general than the one given here. Another generalization is to allow for different distributions for the query answers given. One could also attribute different execution costs to different queries. Finally, it is possible to drop the assumption that queries completely rule out certain states, and rather take a probabilistic approach.

- $q_{-x_i} = (V_i, C, \Theta - V_i - C_{-x_i} - C_{y_i}, \Theta - V_i - C_{-x_i} - C_{-y_i})$.

We have n steps of deliberation. Finally, the goal is $G = \frac{2|V|H+1}{2|\Theta|}$. First suppose there is a solution to the SSAT instance, that is, there exists a contingency plan for setting the x_i such that the probability that the formula is eventually satisfied is at least $\frac{1}{2}$. Now, if we ask query q_{x_i} (q_{-x_i}), we say this *corresponds* to us selecting x_i ($-x_i$); if the answer to query q_{z_i} is $\Theta - V_i - C_{z_i} - C_{y_i}$ ($\Theta - V_i - C_{z_i} - C_{y_i}$), we say this corresponds to nature selecting y_i ($-y_i$). Then, consider the following contingency plan for asking queries:

- Start by asking the query corresponding to how the first variable is set in the SSAT instance (that is, q_{x_1} if x_1 is set to *true*, q_{-x_1} if x_1 is set to *false*);
- So long as all the queries and answers correspond to variables being selected, we follow the SSAT contingency plan; that is, whenever we have to ask a query, we ask the query that corresponds to the variable that would be selected in the SSAT contingency plan if variables so far had been selected in a manner corresponding to the queries and answers we have seen;
- If, on the other hand, we get V_i as an answer, we proceed to ask $v_{ii}, v_{i(i+1)}, \ldots, v_{i(n-1)}$ in that order;
- Finally, if we get C as an answer, we simply stop.

We make two observations about this policy. First, if the true state of the world is one of the v_{ij}, we will certainly discover this. (Upon asking query i, which is q_{x_i} or $-q_{x_i}$, we will receive answer V_i and switch to q_{ik} queries; then if $j < n$, query $j+1$ will be q_{ij}, we will receive answer $\{v_{ij}\}$, and know the state; whereas if $j = n$, we will eliminate all the other elements of V_{ij} with queries $i+1$ through n, and know the state.) Second, if the true state is b, for any i ($1 \leq i \leq n$), query i will be either q_{x_i} or $-q_{x_i}$. This will certainly eliminate all the v_{ij}, so we will know the state at the end if and only if we also manage to eliminate all the clauses. But now notice that each query-answer pair eliminates exactly the same clauses as the corresponding variable selections satisfy. It follows that we will know the state in the end if and only if these corresponding variable selections satisfy all the literals. But the process by which the queries and answers are selected is exactly the same as in the SSAT instance with the solution policy. It follows we discover the true state with probability at least $\frac{1}{2}$. Hence, our total expected utility is at least $\frac{|V|}{|\Theta|}H + \frac{1}{|\Theta|}\frac{1}{2} = G$. So there is a policy that achieves the goal.

Now suppose there is a policy that achieves the goal. We first claim that such a policy will always discover the true state if it is one of the v_{ij}. For if a policy does not manage this, then there is some v_{ij} such that for some combination of answers consistent with v_{ij} , the policy will not discover the state. Suppose this is indeed the true state. Since each consistent answer to query q occurs with probability at least $\frac{1}{N_{ans}(q)}$, it follows that the unfavorable combination of answers occurs with probability at least $\prod_{q \in Q} \frac{1}{N_{ans}(q)}$. It follows that even if we discover the true state in every other scenario, our expected utility is at most $(\frac{|V|}{|\Theta|} - \frac{1}{|\Theta|}\prod_{q \in Q} \frac{1}{N_{ans}(q)})H + \frac{1}{|\Theta|} = G + \frac{1}{|\Theta|}(-2 + \frac{1}{2}) < G$. Now, it is straightfoward to show that this implies that so long as no answer has been one of the V_i or C, query i ($1 \leq i < n$) is either q_{x_i} or $-q_{x_i}$. Query n may still be q_{nn} under these conditions, but since queries q_{x_n} and q_{-x_n} are both more informative than q_{nn}, we may assume that the policy that achieves the target value asks one of the former two in this case as well. It follows that the part of this policy that handles the cases where no answers have been either one of the V_i or C corresponds exactly to a valid SSAT policy, according to the correspondence between queries/answers and variable selections outlined earlier in the proof. But now we observe, as before, that if the true state is b, the probability that we discover this with the STATE-DISAMBIGUATION policy is precisely the probability that this SSAT policy satisfies all the clauses. This probability must be at least $\frac{1}{2}$ in order for the STATE-DISAMBIGUATION policy to reach the target expected utility value. So there is a solution to the SSAT instance. ∎

The following theorem allows us to make any hardness result on STATE-DISAMBIGUATION go through even when restricting ourselves to a uniform prior over states, or to a constant utility function over the states.

Theorem 5 *Every STATE-DISAMBIGUATION instance is equivalent to another STATE-DISAMBIGUATION instance with a uniform prior p, and to another with a constant utility function u ($u(\theta_t) = 1$ for all $\theta_t \in \Theta$). Moreover, these equivalent instances can be constructed in linear time.*

Proof: The only relevance of p and u in STATE-DISAMBIGUATION is to the policy's expected utility $\sum_{1 \leq t \leq r} p(\theta_t)\pi(\theta_t)u(\theta_t)$. So, only the products $p(\theta_t)u(\theta_t)$ matter; adding a constant factor to them also makes no difference if we correct G accordingly. Hence, any instance is equivalent to one where we replace p and u by $p'(\theta_t) = \frac{1}{|\Theta|}$ and $u'(\theta_t) = |\Theta|p(\theta_t)u(\theta_t)$. It is also equivalent to one where we replace p, u and G by $p''(\theta_t) = \frac{p(\theta_t)u(\theta_t)}{\sum_{1 \leq i \leq r}(p(\theta_i)u(\theta_i))}, u''(\theta_t) = 1, G'' = \frac{G}{\sum_{1 \leq i \leq r}(p(\theta_i)u(\theta_i))}$. ∎

5 Conclusion and future research

In most real-world settings, due to limited time or other resources, an agent cannot perform all potentially useful deliberation and information gathering actions. This leads to the metareasoning problem of selecting such actions carefully. Decision-theoretic methods for metareasoning have been studied in AI for the last 15 years, but there are few theoretical results on the complexity of metareasoning.

We derived hardness results for three metareasoning problems. In the first, the agent has to decide how to allocate its deliberation time across anytime algorithms running on different problem instances. We showed this to be $\mathcal{NP}$-complete. In the second, the agent has to (dynamically) allocate its deliberation or information gathering resources across

multiple actions that it has to choose among. We showed this to be $\mathcal{NP}$-hard even when evaluating each individual action is very simple. In the third, the agent has to (dynamically) choose a limited number of deliberation or information gathering actions to disambiguate the state of the world. We showed that this is $\mathcal{NP}$-hard under a natural restriction, and $\mathcal{PSPACE}$-hard in general.

Our results have general applicability in that most metareasoning systems must somehow deal with one or more of these problems (in addition to dealing with other issues). The results are not intended as an argument against metareasoning or decision-theoretic deliberation control. However, they do show that the metareasoning policies directly suggested by decision theory are not always feasible. This leaves several interesting avenues for future research: 1) investigating the complexity of metareasoning when deliberation (and information gathering) is costly rather than limited, 2) developing optimal metareasoning algorithms that usually run fast (albeit, per our results, not always), 3) developing fast optimal metareasoning algorithms for special cases, 4) developing approximately optimal metareasoning algorithms that are always fast, and 5) developing meta-metareasoning algorithms to control the meta-reasoning, etc.

References

[1] Eric B Baum and Warren D Smith. A Bayesian approach to relevance in game playing. *Artificial Intelligence*, 97(1–2):195–242, 1997.

[2] Mark Boddy and Thomas Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–285, 1994.

[3] Irving Good. Twenty-seven principles of rationality. In V Godambe and D Sprott, eds, *Foundations of Statistical Inference*. Toronto: Holt, Rinehart, Winston, 1971.

[4] E Hansen and S Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126:139-157, 2001.

[5] Eric Horvitz. Principles and applications of continual computation. *Artificial Intelligence*, 126:159–196, 2001.

[6] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. *Workshop on Uncertainty in AI*, pp. 429–444, Seattle, Washington, 1987. American Assoc. for AI. Also in L. Kanal, T. Levitt, and J. Lemmer, eds, *Uncertainty in AI 3*, Elsevier, 1989, pp. 301-324.

[7] Ronald Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966.

[8] Kate Larson and Tuomas Sandholm. Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, 132(2):183–217, 2001. Early version: AAAI, pp. 48–55, Austin, TX, 2000.

[9] Kate Larson and Tuomas Sandholm. Costly valuation computation in auctions. *Theoretical Aspects of Rationality and Knowledge (TARK)*, pp. 169-182, 2001.

[10] James E Matheson. The economic value of analysis and computation. *IEEE Transactions on Systems Science and Cybernetics*, 4(3):325–332, 1968.

[11] C Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31:288–301, 1985.

[12] David Parkes and Lloyd Greenwald. Approximate and compensate: A method for risk-sensitive meta-deliberation and continual computation. In *AAAI Fall Symposium on Using Uncertainty within Computation*, 2001.

[13] S Russell and D Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 1:1–36, 1995.

[14] S Russell and E Wefald. *Do the right thing: Studies in Limited Rationality*. 1991.

[15] Tuomas Sandholm and Victor Lesser. Utility-based termination of anytime algorithms. *ECAI Workshop on Decision Theory for DAI Applications*, pp. 88-99, Amsterdam, 1994. Extended version: UMass CS tech report 94-54.

[16] Tuomas Sandholm and Victor Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94:99-137, 1997. Early version: IJCAI-95,662-669.

[17] Herbert A Simon. *Models of bounded rationality*, volume 2. MIT Press, 1982.

[18] Shlomo Zilberstein, François Charpillet, and Philippe Chassaing. Real-time problem solving with contract algorithms. *IJCAI*, pp. 1008–1013, 1999.

[19] Shlomo Zilberstein and Abdel-Illah Mouaddib. Reactive control of dynamic progressive processing. *IJCAI*, pp. 1268–1273, 1999.

[20] Shlomo Zilberstein and Stuart Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1–2):181–213, 1996.