Optimizing Decision Quality with Contract Algorithms

Shlomo Zilberstein[*]
Computer Science Department
University of Massachusetts
Amherst, MA 01003 U.S.A.
shlomo@cs.umass.edu

Abstract

Contract algorithms offer a tradeoff between output quality and computation time, provided that the amount of computation time is determined prior to their activation. Originally, they were introduced as an intermediate step in the composition of interruptible anytime algorithms. However, for many real-time tasks such as information gathering, game playing, and a large class of planning problems, contract algorithms offer an ideal mechanism to optimize decision quality. This paper extends previous results regarding the meta-level control of contract algorithms by handling a more general type of performance description. The output quality of each contract algorithm is described by a probabilistic (rather than deterministic) *conditional performance profile.* Such profiles map input quality and computation time to a probability distribution of output quality. The composition problem is solved by an efficient off-line compilation technique that simplifies the run-time monitoring task.

1 Decision making with contract algorithms

The wide performance variability of artificial intelligence techniques, most notably in search and knowledge-based systems, has been a major obstacle in applying these techniques to real-time environments. This problem led to the development of a variety of approximation techniques such as *anytime algorithms* [Dean and Boddy, 1988; Horvitz, 1987], *design-to-time* [Garvey and Lesser, 1993] and various *progressive reasoning* methods [Mouaddib and Zilberstein, 1995]. It is by now well understood that a successful system must trade off decision quality for computation time. Anytime algorithms in particular offer a simple means by which a system can monitor and maximize its overall utility.

Contract algorithms are a special type of anytime algorithms that was introduced in order to simplify the anytime algorithm composition problem [Russell and Zilberstein, 1991]. General anytime algorithms are interruptible, but naive composition of anytime algorithms destroys interruptibility since no results are available before the last component is activated. Similar to interruptible algorithms, contract algorithms offer a tradeoff between computation time and quality of results, but the amount of computation time must be determined prior to activation so that time can be allocated optimally to the components. This leads to a two step solution to the composition problem of interruptible algorithms: first the system is compiled into one large contract algorithm and then this algorithm is made interruptible with only a small, constant penalty [Zilberstein, 1993].

Despite the original motivation for their development, contract algorithms may be just the right solution in many real-time problem domains. Such domains as information gathering, database query processing, game playing, and many planning and scheduling tasks are characterized by a *predictable utility function*, that is, the utility of results of quality q at a future time t can be determined by the current state of the domain. In such domains, and in domains that have *near-predictable utility function*, it is advantageous to utilize contract algorithms rather than interruptible ones, given the performance degradation associated with the contract to interruptible conversion.

Many existing programming paradigms can be used to construct useful contract algorithms. Examples include iterative deepening search, iterative improvement algorithms in numerical computation, variable precision logic, relational database query answering, and randomized techniques such as Monte Carlo algorithms and fingerprinting algorithms. For a survey of such programming techniques and examples of their AI applications see [Zilberstein, 1993]. The composition of these algorithms, however, presents a non-trivial meta-level resource allocation problem. For any given contract time allocated to a composite system, the problem is how to allocate this time to the components so as to maximize the *overall* output quality. We refer to this problem as the contract algorithm composition problem.

The rest of this paper presents an efficient solution to the contract algorithm composition problem. Section 2

[*]URL: http://anytime.cs.umass.edu/~shlomo.

defines the problem formally and describes the main factors that determine the complexity of the problem. In Section 3, we show that the resource allocation problem can be mapped to a decision problem represented by an influence diagram. Unfortunately, standard algorithms for evaluating influence diagrams perform poorly on this problem. Section 4 describes an off-line compilation process and a run-time monitoring technique that offer an alternative solution to the resource allocation problem. In section 5, we show that the compilation problem can be solved efficiently for a large class of composite systems. As a result, large systems composed of contract algorithms can be optimally monitored with negligible run-time overhead. We conclude with a summary of the contribution of this work and some open problems.

2 The meta-level resource allocation problem

This section defines the resource allocation problem that arises when a system is composed of a set of contract algorithms. To formally define the problem, one must answer a number of basic questions. These questions are discussed below.

Program structure The first question is what type of programming constructs are used to connect the system's components. In this paper, we will consider the case of functional composition as the only programming construct. The results, however, can be generalized to additional programming constructs as shown in [Zilberstein, 1993].

```
SpeechRecognizer(utterance)
      LinguisticVerification(
         GenerateInterpretation(
            FilterNoise(utterance),
            ClassifySpeaker(utterance)),
         GenerateContext(state))
```

Figure 1: A speech recognition module composed of contract algorithms.

Many systems can be described at the top level as a composition of a set of modules. Consider for example a speech recognition system whose structure is shown in Figure 1. Each elementary function represents a contract algorithm. The lower level modules filter noise from the input and classify the speaker (in terms of geneder, accent, and other features that may be used to calibrate the interpretation module). The results are passed to a function that generates possible interpretations. Finally, candidate interpretations along with the current context are passed to a function that performs linguistic verification and determines the best interpretation. Each one of these functions can be composed of more primitive contract algorithms. The resource allocation problem is the problem of calculating the execution time of each elementary component, so as to maximize the quality of the final interpretation.

Performance profiles The second question is what meta-level knowledge is used to characterize the performance of individual contract algorithms. We use discrete *conditional performance profiles* (CPPs) that map input quality and run-time to a discrete probability distribution of output quality. For an algorithm A with two inputs, for example, the CPP is denoted by $Q_A(i,j,t)$, and $Q_A(i,j,t)[k]$ is the probability of output quality q_k with input qualities (q_i, q_j) and time allocation t. Each quality measure, $0 \leq q_i \leq 1$, can represent the level of certainty, precision, or specificity of the data. These quality measures are "local" to each component. Part of the composition problem is to propagate the effect of quality degradation in lower levels on the overall quality of the system and on its utility. The CPP of an elementary contract algorithm can be determined empirically by running the algorithm over randomly selected problem instances [Dean and Boddy, 1988; Zilberstein, 1993]. In fact, we are currently developing a set of programming tools to mechanize the construction of CPPs and to store them in a library for later use [Grass and Zilberstein, 1995].

Time-dependent utility functions The third question is how the quality of the output of the system will affect the domain, $\mathcal{D}$, in which it operates. As we mentioned earlier, we assume that the environment is characterized by a predictable utility function, $U_{\mathcal{D}}(S,t,q)$. This function represents the utility of a result of quality q in state S at time t. For example, in the speech recognition domain described above, the state S may represent situations with different level of error sensitivity, q may represent the probability of correct interpretation and the overall utility may depend on S and q, as well as on the delay in producing the interpretation.

Monitoring schemes The fourth question is how to determine the total allocation of computation time to the system and how to monitor the execution of the components. In this paper, our goal is to derive the optimal allocation prior to the activation of the system. This approach leads to a simple monitoring scheme by which every module is activated with a fixed, predetermined contract time. Several strategies for adjusting the allocation of residual time based on the *actual* progress in problem solving have been proposed in [Zilberstein, 1993]. These techniques can be used to modify the fixed contract strategy to improve performance when there is a large degree of uncertainty regarding the output of each module.

The answers to the above four questions produce a well-defined meta-level resource allocation problem. Namely, given a system that is a functional composition of contract algorithms, the CPPs of the components, and a time-dependent utility function, what is the best overall contract and how should the time be distributed to the components so as to maximize the overall utility of the system.

3 Optimal resource allocation using influence diagrams

The time allocation problem defined above can be represented as a decision problem using influence diagrams [Howard and Matheson, 1981]. The construction of the influence diagram is a trivial modification of the directed acyclic graph (DAG) representation of the system itself.

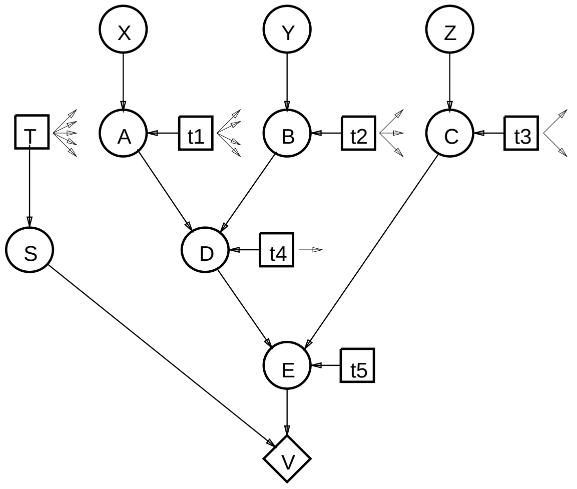

Figure 2: Influence diagram representation of the meta-level resource allocation problem.

For example, the speech recognition system described above can be schematically represented by the following functional expression: E(D(A(x),B(y)),C(z)) where A, B, C, D and E are contract algorithms. Given such an expression, we can map the resource allocation problem to the influence diagram shown in Figure 2. In this diagram, each elementary contract algorithm is represented by a chance node that corresponds to the uncertainty regarding the output quality of the algorithm. In addition, we represent the quality of each input, X, Y and Z, by a chance node. The output quality of each contract algorithm is influenced by the qualities of the inputs and by the time allocation to that node. This dependency is characterized by the CPP of the contract algorithm that forms the conditional probability matrix attached to the node. The problem is then to determine the total allocation of time, T, and the sub-allocations t1, ..., t5 to each component, that would maximize the overall quality of the system. Note that the overall utility is represented by a value node that is influenced by the quality of the system's output and by the state of the environment, represented by a single chance node S. The state of the environment is more likely to be determined be a complex probabilistic model but since we focus in this paper on the behavior of the decision making components, a single node is used to represent that influence.

Note that, in general, the decision diagrams that we get satisfy the following two restrictions:

1. There is a total ordering among the decision nodes that is determined by the order of evaluation of the composite system.
2. Each decision node influences all the successive decisions (since only the remaining time can be allocated at each step). To simplify the diagram, the links representing this influence are shown as short outgoing arrows.

Under these two conditions, Shachter's [1986] transformation approach can be applied to evaluate the diagram. But the complexity of this evaluation technique is high even with a small number of modules. The reason is the exponential complexity of the algorithm combined with the fact that the discrete time variables may range over a large number of values. As a result, solving the time allocation problem by standard evaluation techniques for influence diagrams is not practical.

The question is what properties or reasonable restrictions can be introduced so as to reduce the complexity of the run-time resource allocation problem. An efficient alternative solution will not only simplify the composition of contract algorithms, but will also apply to a general class of resource allocation problems represented by the above influence diagram. These general resource allocation problems are characterized by the following three properties:

1. Each computational element offers a trade off between resource consumption (not necessarily computation time) and output quality;
2. Resource availability is limited; and
3. Conditional performance profiles can be constructed to characterize the dependency of output quality on input quality and resource allocation.

This class of resource allocation problems is, in general, NP-complete [Zilberstein, 1993], but under a set of reasonable assumptions it can be solved efficiently by the compilation process described in the next two sections.

4 Optimal resource allocation using off-line compilation

As an alternative to the influence diagram solution, we developed a two-step solution based on (1) an off-line compilation process that derives the optimal allocation to the components for any *given* contract time, and constructs the best performance profile of the whole system; and (2) a run-time monitoring technique that determines the optimal overall contract time in any given situation.

Calculating the optimal CPP of the system When the computation time of each component of a system is known, we can easily derive the probability distribution of the output. Hence, for any given contract time, we can in principle find the best apportionment of time to the components based on the resulting quality distribution (and the utility function of the system). We refer to this problem as *global compilation* of contract algorithms, since it solves the global optimization problem directly. Global compilation is analogous to evaluating the influence diagram of the previous section. It is obviously an NP-complete problem and cannot be solved even off-line for large programs [Zilberstein, 1993]. The *local compilation* technique presented in Section 5 addresses this complexity issue.

Calculating the optimal contract time Suppose that the system has been compiled into a single contract algorithm, A, whose CPP is $Q_A(i,j,t)$. Let S_0 be the current state of the domain, and let S_t represent the state of the domain at time t. Let q_t represent the quality of the result of the contract algorithm at time t, and let $U_{\mathcal{D}}(S,t,q)$ be the time-dependent utility function. The optimal contract time is calculated as follows.

Due to uncertainty concerning the quality of the result of the algorithm, the expected utility of the result in state S_t at time t is represented by:

$$U'_{\mathcal{D}}(S_t,t) = \sum_k Q_A(i,j,t)[k]U_{\mathcal{D}}(S_t,t,q_k) \tag{1}$$

The probability distribution of future output quality is provided by the CPP of the algorithm. Due to uncertainty regarding the future state of the domain, the expected utility of the results at time t is represented by:

$$U''_{\mathcal{D}}(t) = \sum_{S_t\in\mathcal{S}} Pr(S_t|S_0)U'_{\mathcal{D}}(S_t,t) \tag{2}$$

The probability distribution of the future state is calculated using a probabilistic model of the environment. Now, the optimal contract time, t_c, can be determined before the system is activated by solving the following equation:

$$t_c = arg\max_t\{U''_{\mathcal{D}}(t)\} \tag{3}$$

As we mentioned earlier, once an initial contract time is determined, several monitoring strategies can be used to modify the allocation based on the *actual* progress in problem solving. In particular, we have studied two strategies for contract adjustment [Zilberstein, 1993]. The first strategy re-allocates residual time among the remaining modules once the result of a module becomes available. The second strategy adjusts the original contract each time an elementary component terminates. In the latter case, the monitor considers the output provided by an intermediate computation as input to a smaller residual system composed of the remaining contract algorithms. At that point, a better contract time can be determined that takes into account the actual quality of all the intermediate results generated so far.

To summarize, the compilation and monitoring approach to meta-level resource allocation is a valuable alternative, provided that the off-line compilation process could be performed efficiently. This is achieved by the local compilation technique described below.

5 The optimality of local compilation

In this section we show how the composition of contract algorithms can be solved efficiently by an off-line local compilation process. The goal of the compilation process is to produce the best possible CPP of a composite module based on the CPPs of the components. Instead of solving the global optimization problem directly, local compilation calculates the optimal CPP of an algorithm based on the CPPs of its immediate components. If those components are not elementary, their CPPs are calculated using local compilation as well.

To simplify our analysis and to be able to guarantee both efficiency and optimality, we use the following three assumptions:

1. The input monotonicity assumption that the output quality of a contract algorithm improves as the quality of its input(s) improves. This assumption is not only reasonable but also represents a desired property of every contract algorithm.
2. The bounded degree assumption that the number of inputs to each individual algorithm is bounded. This assumption represents a good software engineering practice.
3. The tree-structured assumption that the structure of the program can be represented by a directed tree. This is the only real restriction since general functional expressions have a DAG representation but not necessarily a tree-structured one. Several techniques to remove this restriction and achieve optimal (or near-optimal) performance are described in [Zilberstein, 1993]. These techniques are analogous to the methods used to evaluate general Bayesian networks [Pearl, 1988].

5.1 Optimality of local compilation of deterministic CPPs

We first prove the optimality of local compilation when the CPP of each contract algorithm is deterministic, that is, when the input quality and run-time determine exactly the output quality. Under the above three assumptions, and without limiting the generality of the discussion, we will consider binary functions only and assume that the composite expression is a complete binary tree.

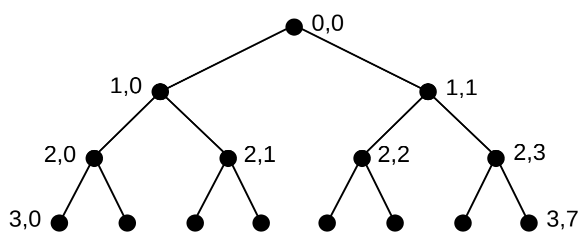

Figure 3: A tree representation of a composite expression.

Let $f_{i,j}$ denote the j^{th} function on the i^{th} level of the tree. The root node is denoted accordingly by $f_{0,0}$. If the tree is of depth n, then the nodes corresponding to $f_{n,0}, ..., f_{n,2^n-1}$ are leaf nodes whose inputs are input variables. For any other node $f_{i,j}$, $0 \le i \le n-1$, $0 \le j \le 2^i - 1$, the inputs are: $f_{i+1,2j}$ and $f_{i+1,2j+1}$ as shown in Figure 3.

Corresponding to each node of the binary tree, there is a deterministic CPP, $Q_{i,j}(q_1, q_2, t)$, which determines the output quality for that node as a function of its input qualities, q_1 and q_2, and time allocation, t. Given a composite expression e of depth n, and a particular input quality, the global compilation problem is the problem of finding the optimal time allocation to all the nodes of the tree that would maximize the output quality of the root node:

$$Q^G_e(t) = arg \max_{t_{i,j}} Q^*_{0,0} \quad : \quad \sum_{0 \le i \le n} \sum_{0 \le j \le 2^i - 1} t_{i,j} = t \qquad (4)$$

where $Q^*_{0,0}$ denotes the result of replacing (in the expression e) every function by its CPP and every input variable by its quality.

Given a particular composite expression e of depth n, a local compilation scheme for e is defined by induction on its structure. For a leaf node, the locally compiled CPP is the elementary CPP of that node:

$$Q^L_{n,j}(t) = Q_{n,j}(q_{n,j,1}, q_{n,j,2}, t), \quad 0 \le j \le 2^n - 1 \qquad (5)$$

where $q_{n,j,1}$ and $q_{n,j,2}$ are the qualities of the two inputs of the particular function. For each internal node, the locally compiled CPP is defined using the CPPs of its immediate inputs as follows:

$$Q^L_{i,j}(t) = arg \max_{t_1, t_2}$$
$$\{Q_{i,j}(Q^L_{i+1,2j}(t_1), Q^L_{i+1,2j+1}(t_2), t - t_1 - t_2)\} \qquad (6)$$

Finally, the CPP of e (as a result of local compilation) is denoted by the following expression:

$$Q^L_e(t) = Q^L_{0,0}(t) \qquad (7)$$

Note that the external input quality was deliberately omitted in this notation since the focus is on the result of local compilation for any *given* input quality.

Using a discrete tabular representation of CPPs with τ time units and assuming the the size of the composite expression is κ, local compilation requires $O(\tau^2)$ work per (internal) node of the tree. Hence the total amount of work is $O(\kappa\tau^2)$ (for each possible input quality). In terms of space requirements, even though local compilation requires $O(\kappa)$ separate CPPs (one for each internal node of the tree), its total space requirement is only a constant factor more than the space requirement of global compilation. The reason is that a compiled global CPP needs to specify the allocation to *each* node of the tree (i.e. κ elements) for each total allocation while a compiled local CPP needs to specify only the allocation to the immediate successors of each node and to the node itself (i.e. three elements). To summarize, local compilation has the same space requirements as global compilation but it reduces the time complexity of the global optimization problem from exponential to polynomial. Moreover, the complexity is linear in the size of the program.

More importantly, the following theorem guarantees that the final result of local compilation is globally optimal.

Theorem 5.1 ***Optimality of local compilation of deterministic CPPs:*** *Let e be a composite expression of an arbitrary depth n whose conditional performance profiles satisfy the input monotonicity assumption, then for any input and contract time t:*

$$Q^L_e(t) = Q^G_e(t)$$

Proof: By induction on the depth of the tree. For trees of depth 1 the claim is trivially true because both compilation schemes solve the same optimization problem. Suppose that the claim is true for trees of depth $n-1$ or less. Let e be an expression of depth n, and let $t_{i,j}$ be the allocations to $f_{i,j}$ based on global compilation and resulting in a global optimum. Let t_l and t_r be respectively the total allocation to the left and right subtrees of the root node:

$$t_l = \sum_{i=1}^{n} \sum_{j=0}^{2^{i-1}-1} t_{i,j} \qquad (8)$$

$$t_r = \sum_{i=1}^{n} \sum_{j=2^{i-1}}^{2^i-1} t_{i,j} \qquad (9)$$

$$t = t_l + t_r + t_{0,0} \qquad (10)$$

Then:

$$Q^G_e(t) \quad =$$

By definition and monotonicity:

$$= \quad Q_{0,0}(Q^G_{1,0}(t_l), Q^G_{1,1}(t_r), t_{0,0}) \qquad (11)$$

By the induction hypothesis:

$$= \quad Q_{0,0}(Q^L_{1,0}(t_l), Q^L_{1,1}(t_r), t_{0,0}) \qquad (12)$$

By local compilation:

$$\le \quad Q^L_{0,0}(t) \qquad (13)$$

By definition:

$$= \quad Q^L_e(t) \qquad (14)$$

By global compilation:

$$\le \quad Q^G_e(t) \qquad (15)$$

Hence $Q^L_e(t) = Q^G_e(t)$ □

Note that the input monotonicity assumption is required to guarantee the optimality of local compilation. The bounded degree assumption, on the other hand, is only used to guarantee that the complexity of local compilation of each internal node is polynomial in t.

5.2 Optimality of local compilation of probabilistic CPPs

We now analyze local compilation of contract algorithm characterized by probabilistic CPPs. This case is more complicated since we have to optimize over probability distributions of quality, rather than over fixed quality. Suppose, again, that each module has exactly two inputs. A general case of local compilation optimizes the CPP of a contract algorithm C whose inputs are produced by the algorithms A and B. Given the CPPs of A, B and C, and a particular time allocation (t_A, t_B, t_C), the probability of output quality q_k is given by:

$$Pr(q_k) = \sum_{k_A} \sum_{k_B} Q_A(i_A, j_A, t_A)[k_A] \\ Q_B(i_B, j_B, t_B)[k_B] Q_C(k_A, k_B, t_C)[k] \tag{16}$$

The expected output quality is:

$$Q^*_C(t_A, t_B, t_C) = \sum_k q_k Pr(q_k) \tag{17}$$

A natural approach to local compilation is to maximize the *expected* quality of the output for any given total allocation, t. This can be done by solving the following local optimization problem:

$$arg \max_{t_A+t_B+t_C=t} \{ \; Q^*_C(t_A, t_B, t_C) \; \} \tag{18}$$

But, without knowing the CPP that a module may affect, one cannot determine which probability distribution is superior since, in general, it is possible that $\overline{x} > \overline{y}$, but $\overline{q(x)} < \overline{q(y)}$ (even when q satisfies the input monotonicity property).

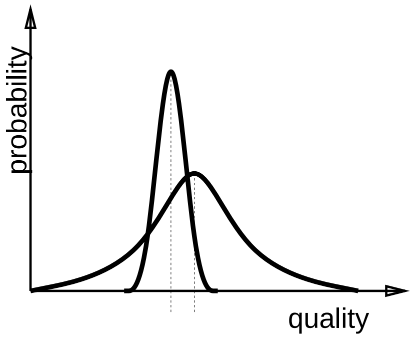

Figure 4: It is not possible to determine which one of the following quality distributions is "better" without knowing the CPP that they affect.

Suppose, for example, that we compare two alternative allocations that produce the quality distributions shown in Figure 4. It is impossible to determine locally which one is better based on expected quality or any other aspect of the distributions. In particular, if low output quality has a "disastrous" effect on the system's performance, then the distribution that has a slightly lower expected value but a higher lower bound on quality may be better.

The question is under what circumstances will the local optimization approach (based on expected quality) guarantee that the result is globally optimal. A sufficient assumption is that the dependency of output quality on input quality is linear. We call this assumption the *input linearity* assumption. Input linearity means that for any given time allocation, the probability of output quality q_k is a linear function of the input qualities. Since we only consider increasing linear functions, input linearity implies input monotonicity. In such case, maximizing expected quality at each compilation step will yield the globally optimal results. Hence we get the following result:

Theorem 5.2 ***Optimality of local compilation of probabilistic CPPs:*** *Let e be a composite expression of an arbitrary depth n whose conditional performance profiles satisfy the input linearity assumption, then for any input and contract time t:*

$$Q^L_e(t) = Q^G_e(t)$$

Proof: An immediate result of Theorem 5.1 and the input linearity assumption, since for any linear function q, $\overline{q(x)} = q(\overline{x})$. Note that input linearity guarantees that the expected output quality of a module is a linear function of the qualities of its inputs, therefore we can simply maximize expected input quality to guarantee maximal expected output quality. □

Obviously, the validity of the input linearity assumption cannot be generally established. However, we suggest that it is a reasonable approximation even in those cases where the dependency is not totally linear. Based on passed experience with contract algorithms, the dependency of output quality on input quality follows a general pattern. With very low input quality, the algorithm tends to fail. Very high input quality does not have a significant marginal effect on output quality. But withing the middle range of possible input quality, the dependence is near-linear. This has been the case with several algorithms that we have developed in the past. We argue that the optimal operation of a system will normally result in a contract time that falls within the middle range boundaries, hence the linearity assumption is a good approximation. Further experimental work is needed to assess the validity of the last argument.

5.3 Applications of the compilation technique

The advantages of compilation of contract algorithms have already been demonstrated by two applications in the area of mobile robot navigation [Zilberstein and Russell, 1993] (using deterministic CPPs) and in the area of model-based diagnosis [Pos, 1993] (using a slightly modified type of performance profiles, called Statistical Performance Profiles). This paper extends those results by analyzing the use of probabilistic CPPs that capture more accurately the performance of contract algorithms.

Another application of this work is to the evaluation of a general class of decision problems that normally arise in resource allocation. When the resulting influence diagram has the structure and properties described in Section 3, it can be evaluated efficiently using the local compilation technique. The idea of compiling CPPs representing conditional probability matrices leads to a powerful tractable evaluation technique for such decision problems. The technique is especially useful in real-time domains where the meta-level resource allocation problem must be solved quickly at run-time.

6 Conclusion

Contract algorithms offer a flexible building block for systems that must trade off decision quality for computational resources. We have formalized the problem of resource allocation that arises in systems composed of contract algorithm and offered two solutions to the problem. The first solution exploits existing algorithms for evaluation of influence diagrams but it is inefficient. The second technique relies on local compilation of CPPs and is both efficient and optimal under certain conditions, most of which are satisfied by any system composed of contract algorithms. We argue that the input linearity assumption, needed to guarantee the optimality of local compilation of probabilistic CPPs, is a reasonable approximation and therefore it leads to near-optimal results.

These results apply to the construction of a large class of AI systems that operate in domains with time-dependent, predictable utility functions. In addition, local compilation is a powerful technique for evaluation of decision models that share the structure of the influence diagram in Section 3. For this type of resource allocation decisions, local compilation offers a significant complexity reduction based on the monotonicity of CPPs.

Further work is needed to analyze the effect of non-linear input-output dependency and to apply the results to larger systems composed of contract algorithms. Our ultimate goal is to provide an analytical foundation for the widespread use of reusable contract algorithms in various decision making applications.

Acknowledgements

Support for this work was provided in part by the National Science Foundation under grant IRI-9409827 (Research Initiation Award) and by a Faculty Research Grant from the University of Massachusetts.

References

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

[Garvey and Lesser, 1993] A. Garvey and V. Lesser. Design-to-time real-time scheduling. In *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1491–1502, 1993.

[Grass and Zilberstein, 1995] J. Grass and S. Zilberstein. Programming with anytime algorithms. In *Proceedings of the IJCAI-95 Workshop on Anytime Algorithms and Deliberation Scheduling*, Montreal, Canada, 1995.

[Hayes-Roth *et al.*, 1991] B. Hayes-Roth *et al.* Guardian: A prototype intelligent agent for intensive-care monitoring. Technical Report KSL-91-42, Stanford Knowledge Systems Laboratory, Stanford, California, 1991.

[Horvitz, 1987] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, Seattle, Washington, 1987.

[Howard and Matheson, 1981] R. A. Howard and J. E. Matheson. Influence diagrams. In *Principles and Applications of Decision Analysis*, vol. 2, Menlo Park, California: Strategic Decision Group, 1984.

[Mouaddib and Zilberstein, 1995] A. I. Mouaddib and S. Zilberstein. Knowledge-based anytime computation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Los Altos, California: Morgan-Kaufmann, 1988.

[Pos, 1993] A. Pos. Time-constrained model-based diagnosis. Master Thesis, Department of Computer Science, University of Twente, The Netherlands, 1993.

[Russell and Wefald, 1991] S. J. Russell and E. H. Wefald. Principles of metareasoning. *Artificial Intelligence*, 49:361–395, 1991.

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

[Shachter, 1986] R. D. Shachter. Evaluating influence diagrams. *Operation Research* 34(6):871–882, 1986.

[von Neumann and Morgenstern, 1947] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* 2nd ed. Princeton, New Jersey: Princeton University Press, 1947.

[Zilberstein, 1993] S. Zilberstein. Operational rationality through compilation of anytime algorithms. Ph.D. dissertation, Computer Science Division, University of California at Berkeley, 1993.

[Zilberstein and Russell, 1993] S. Zilberstein and S. J. Russell. Anytime sensing, planning and action: A practical model for robot control. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1402–1407, Chambery, France, 1993.

[Zilberstein and Russell, 1995] S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, forthcoming, 1995.