??, ??, 1–18 (??)

Resource-Bounded Sensing and Planning in Autonomous Systems

SHLOMO ZILBERSTEIN

shlomo@cs.umass.edu

Computer Science Department, LGRC, Box 34610, University of Massachusetts, Amherst, MA 01003

Received ??. Revised ??.

Abstract. This paper is concerned with the implications of limited computational resources and uncertainty on the design of autonomous systems. To address this problem, we redefine the principal role of sensor interpretation and planning processes. Following Agre and Chapman's plan-as-communication approach, sensing and planning are treated as computational processes that provide *information* to an *execution architecture* and thus improve the overall performance of the system. We argue that autonomous systems must be able to trade off the quality of this information with the computational resources required to produce it. Anytime algorithms, whose quality of results improves gradually as computation time increases, provide useful performance components for time-critical sensing and planning in robotic systems. In our earlier work, we introduced a compilation scheme for optimal composition of anytime algorithms. This paper demonstrates the applicability of the compilation technique to the construction of autonomous systems. The result is a flexible approach to construct systems that can operate robustly in real-time by exploiting the tradeoff between time and quality in planning, sensing and plan execution.

Keywords: Real-time sensing and planning, anytime algorithms, deliberation scheduling, meta-level reasoning, uncertainty

1. Introduction

Resource-bounded reasoning is concerned with the implications of limited computational resources on the design of intelligent systems (Simon, 1982; Dean and Boddy, 1988; D'Ambrosio, 1989; Horvitz, 1989; Russell and Wefald, 1991). Recent work in this field, in particular in the area of anytime computation (Boddy and Dean, 1994; Zilberstein and Russell, 1993), is promising to close an existing gap between the theoretical work on sensing and planning and the constraints of real-world applications.

Early work on such planning systems as STRIPS and NOAH devoted substantial effort to monitoring of planning and plan execution (Fikes, Hart and Nilsson, 1972; Sacerdoti, 1977). However, later work in the AI community has concentrated on systems with perfect sensors and effectors and with unlimited computational power (see (Drummond and Tate, 1989) for a comprehensive survey). These, sometimes hidden, assumptions have restricted the applicability of the results. The first step towards a more successful system architecture is to generalize the role of planning and sensing[1] in autonomous systems. In Section 2, we propose a new approach to sensing and planning as sources of information whose goal is to improve the performance of the system. This approach provides a more realistic theoretical foundation for sensing, planning and control of autonomous agents.

Anytime computation plays a central role in the implementation of our approach. Anytime algorithms (Dean and Boddy, 1988; Horvitz, 1989) introduce a new tradeoff in programming – between computation time and quality of results. This degree of freedom is especially useful when developing the deliberative components and control mechanism of a robotic system. The flexibility offered by anytime algorithms allows accurate sensing and extended planning when time is available, and coarse, fast sensing and planning under time pressure. It is based on the observation that

in order to cope with complex environments in real-time, there is no need to sacrifice the ability to do precise sensing and planning. In Section 3, we show how the sensing and planning components of an agent can be implemented as anytime algorithms. An automated compilation scheme is used in order to optimally compose the components of the system (Zilberstein, 1993). In addition to offering a trade off between quality and time, the new approach separates two important aspects of autonomous agents, namely, the construction of the deliberative components and the optimization of performance. The latter task is achieved by a utility-maximizing run-time monitor.

Run-time monitoring is another important component of our approach. As time allocation to the anytime sensing and planning components becomes a degree of freedom, incremental scheduling and constant monitoring are necessary in order to guarantee the optimal allocation of time. In Section 4, we describe how the run-time monitoring component allocates computation time to the anytime modules based on the performance information produced by the compilation process. In order to reason efficiently at run-time about time allocation, the monitor uses *conditional performance profiles* that give a probabilistic description of the quality of the results of an algorithm as a function of run-time and input quality (or any set of input properties). This is an extension of an earlier notion of performance profile that depends on run-time only (Dean and Boddy, 1988).

To demonstrate our approach we have selected one of the fundamental problems facing any mobile system: the capability to plan its own motion with noisy sensors. Our experiments with a simple, simulated domain are designed to demonstrate our approach. They show that resource-bounded reasoning and anytime computation can simplify the construction and improve the performance of autonomous systems. In Section 5, we show that the advantages of our approach can be applied to the construction of real-world robots. We conclude with a summary of the benefits of our approach and the long-term goals of this research.

2. The role of sensing and planning in autonomous systems

Sensing and planning are computational processes aimed at helping an autonomous robot (or an "agent") to act intelligently in the world. The goal of sensing and planning is to improve the performance of an agent by providing certain types of information. A similar idea, termed *plan-as-communication*, was introduced by Agre and Chapman (1990). Agre and Chapman confronted this approach with the more traditional approaches that they term *plan-as-program*. The plan-as-program view takes plan use to be like program execution. Plans are built from a set of parameterized primitives (such as PUT-ON(x,y)) using a set of composition operators. Executing a plan means carrying out its primitive actions and monitoring conditions specified by the planner, and performing little or no reasoning about the activity in which the agent is engaged. The plan-as-program view has several disadvantages related to its computational complexity, inability to handle imprecise sensory data and uncertainty. It also requires that plans be too detailed and it fails to relate the plan to the concrete situation.

We share Agre and Chapman's view of traditional planning. Our approach is similar to their plan-as-communication idea, but we offer an additional important step. *Since sensing and planning provide information to the execution architecture, their execution should be managed based on the value of this information.* Moreover, since the quality of the information degrades over time, an autonomous system should trade off the quality of the information against the computation time needed to produce it. The rest of this section explains our approach to sensing and planning and describes its implications on the design of autonomous systems.

2.1. Sensing and planning as information sources

At the heart of any robotic system lies an *execution architecture* that interacts with the physical world by performing certain *external actions*. (The term external action refers to the interaction between the robot and the environment. It is used to distinguish external actions from in-

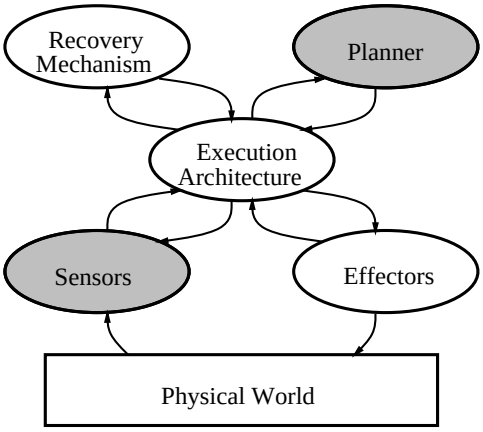


Fig. 1. In practice, the sensing and planning components of a robot interact with the execution architecture, not directly with the physical world.

ternal actions such as sensor data interpretation, planning, and monitoring.) The goal of sensing and planning is to provide the execution architecture with information that can improve the selection of actions. We distinguish between the actions selected by the planner to construct the plan and the actions selected by the execution architecture which are the *actual* actions performed by the robot. The two processes may be totally different and may sometimes disagree. The performance improvement that achieved by the sensing and planning processes is reflected by the system achieving more of its goals in less time.

The execution architecture (or executor) communicates with and coordinates the operation of the main components of the system that are shown in Figure 1. The primary task of the execution architecture is to monitor the execution of external actions that change the physical world. The *recovery mechanism* can suggest ways to overcome certain "simple" failures. Recovery is normally performed using a simple planning mechanism, for example using a greedy approach to quickly restore the necessary conditions rather than searching for an optimal way. Recovery can also be implemented as a reactive component. The planner receives task definitions and information about the current state from the execution architecture and it creates information (in the form of an abstract plan) that helps to achieve the goals of the system. Different representations of states, operators and plans may be used by different systems, but the general role of the execution architecture as described above remains the same.

The sensing and planning modules of a robot should be designed to support a particular *execution architecture*. Unfortunately, much of the work on sensing and planning within the AI community has been carried out with respect to either trivial or ill-specified execution architectures[2]. For example, in STRIPS-style planning and many of its descendants it is assumed that execution is the obvious process of carrying out the actions one by one subject to some ordering constraints. The correctness of a plan is established based on off-line analysis (termed *temporal projection*) that shows that the goal is true after the plan is executed. The state of the domain is totally accessible without any realistic sensing and monitoring.

In reality, planning must be accompanied by an execution architecture that can translate the plan into individual external actions, monitor their execution, recognize various types of failures, recover from "simple" failures, and otherwise rely on re-planning. More recent attempts to design planning systems have recognized the central role of the execution architecture. For example, the Entropy Reduction Engine (ERE) (Bresina and Drummond, 1990) is designed to integrate planning and reaction for control of NASA's autonomous rover. The ERE architecture includes three components, reactor, projector and reducer, that have the capability to operate independently and perform their assigned task. Planning performed by the projector is provided as advice to the reactor in order to improve its performance. Similarly, the reductor provides the projector with search control advice. The same approach has been applied in (Lyons and Hendriks, 1992) to construct a "kitting" robot. But non of these systems has the capability to dynamically vary the quality of planning.

The robotics community has a longer tradition of integrating planning with reaction. The typical approach has been to introduce a planner on top of a reactive execution architecture. For example, SROMA (Xiadong and Bekey, 1988) is an adaptive scheduler for mechanical assembly tasks that utilizes plans that are generated off-line. The run-time scheduler adapts planned actions to the

available resources. The planner in this architecture generates a *complete* plan off-line which is then down loaded into a reactive component. The reactive component is allowed to modify action ordering or introduce minor deviations. A similar approach has been used by (Fox and Kempf, 1985) for robot assembly tasks and by (Fraichard and Laugier, 1991) for robot motion planning. What distinguishes our approach from those systems is its capability to control the quality of planning online in order to respond to resource constraints.

2.2. Implications on the design of the sensing and planning components

Sensing and planning help agents to choose actions intelligently. The ultimate goal of an agent is not to derive accurate domain description, "optimal" plans, or even "correct" (when generated) plans, but rather to transform the physical environment into a desired state and thus perform a certain task. We describe such goals using time-dependent utility functions that define the desirability of certain world configurations. Time-dependent utility functions extend the traditional notion of goals (by allowing partial goal satisfaction) and the traditional notion of deadline (by allowing gradual loss of utility as a function of time). Given such a utility function, sensing and planning should be designed and evaluated within the context of a complete working system. While traditional computational properties such as complexity, correctness, and completeness are very important, they do not fully represent the goal of sensing and planning. The rest of this section analyzes the implications of the above definition of sensing and planning on their design and implementation.

The value of sensing and planning Sensing and planning are computational processes and as such they do not immediately change the external environment or achieve the ultimate goals of an agent. For this reason, the outcome of sensing and planning should be treated as a potentially valuable piece of *information*. The value of this information depends on three factors:

1. The objective quality of the information. In sensing, quality reflects the accuracy of the domain description. In planning, quality reflects the efficiency of the solution to the problem defined by the initial planning conditions.
2. The time at which the sensing and planning information Becomes available to the system and the extent of change that might have occurred in the domain.
3. The capability of the agent to interpret the information produced by the sensing and planning processes and to exploit it effectively.

Standard decision-theoretic techniques can be used to determine the value of planning both analytically and experimentally (Howard, 1966; Zilberstein and Russell, 1993). This approach decouples the value of planning and sensing from their absolute correctness. Moreover, approximate information may be more valuable than precise information, if it can be produced much faster and if the robot has the necessary recovery mechanisms to cope with erroneous information.

Sensing and planning as resource-bounded computation The various factors that determine the value of a plan are not independent. In particular, the deliberation time required to improve the quality of a plan will normally degrade the overall utility of the agent. Similar considerations apply to the quality of the domain description generated by the sensing process. Hence, it is useful to analyze sensing and planning as resource-bounded activities and to develop planning systems that can trade off planning quality for deliberation time. Since in many domains the value of continued planning is context dependent, a utility maximizing approach must rely on constant, real-time monitoring of the planning process.

Dean and Boddy (1988), Horvitz (1989), Russell and Zilberstein (1991) and others have shown that anytime algorithms offer a simple means by which an agent can trade off decision quality for deliberation time. In addition, efficient techniques have been developed for composition and monitoring of systems that are composed of anytime algorithms (Zilberstein, 1993). We show that these techniques are useful for constructing and controlling the sensing and planning components of autonomous systems.

Over the past several years, the AI community has been concerned with the capability of deliberative systems to operate robustly in real-time domains. Some successful system architectures that combine deliberative and reactive components include Guardian (Hayes-Roth, 1992) a medical monitoring application based on a blackboard (BB*) architecture, Phoenix (Howe *et al.*, 1990) a real-time system to control fire fighting, PRS (Ingrand *et al.*, 1992) a general architecture for combining reaction and deliberation and CIRCA (Musliner *et al.* 1995) an architecture that combines real-time operation with AI techniques by cooperation between two separate subsystems. While these systems have totally different structure and characteristics, they share the assumption that deliberative AI techniques are inherently unpredictable in terms of performance. Our work is based on the capability to project (and optimize) performance using performance profiles. In this paper, we concentrate on the problem of maximizing *expected performance* rather than providing a particular performance guarantee. However, performance guarantees can be provided with anytime algorithms by using a more informative type of performance profiles (Zilberstein, 1995).

Sensing and planning as ongoing processes Historically, planning has been closely associated with problem solving and similar search techniques have been widely used in both areas (Hendler *et al.*, 1990). However in many "real-world" environments planning is much more than problem solving. Besides the richer representation of operators and goals in planning, the main difference between planning and problem solving is the temporal scope. In particular, a planning problem can contain much more than a single problem solving episode. Many autonomous systems are designed to operate in an environment over an extended period of time. Their long term goal may be "keep all the machines in this room working" or "keep track of all the targets in region Z". Translating such a high level goal into action may not have a fixed solution that can be derived and implemented. Achieving such goals requires an ongoing situation assessment, planning, and action.

To summarize, sensing and planning may involve more than a single problem solving episode. In order to construct successful autonomous systems we need to modify the basic definitions of sensing and planning and treat them as resource-bounded, reasoning activities that provide useful information to an execution architecture in selecting actions. The overall utility of these processes is determined by the effect that they have on the overall performance of the system.

3. Flexible sensing and planning using anytime computation

The term "anytime algorithm" was coined by Dean in the late 1980's in the context of his work on time-dependent planning. Anytime algorithms are algorithms whose quality of results improves gradually as computation time increases, hence they offer a tradeoff between resource consumption and output quality. Many existing programming techniques produce useful anytime algorithms. Examples include iterative deepening search, variable precision logic, and randomized techniques such as Monte Carlo algorithms or fingerprinting algorithms. For a survey of such programming techniques and examples of algorithms see (Zilberstein, 1993).

Various metrics can be used to measure the quality of a result produced by an anytime algorithm. In particular, the following three metrics have been proved useful in anytime algorithm construction: *certainty* – reflects the degree of certainty that the result is correct, *accuracy* – reflects the degree of accuracy or how close is the approximate result to the exact answer, and *specificity* – reflects the level of detail of the result. In the third case, the anytime algorithm always produces *correct* results, but the level of *detail* is increased over time. The anytime planning algorithm presented in this section offers a good example of using specificity as a quality measure.

It should be emphasized that the notion of interrupted computation is almost as old as computation itself. However, traditionally, interruption was used primarily for two purposes: aborting the execution of an algorithm whose results are no longer necessary, or suspending the execution of an algorithm for a short time because a computation of higher priority must be performed. Anytime algorithms offer a third type of interruption: interruption of the execution of an algorithm

whose results are considered "good enough" by their consumer. We show that this type of interruption has many benefits for autonomous systems. The rest of this section describes a particular implementation of anytime sensing and planning, the conditional performance profiles of these algorithms, and the way they were composed.

The application described in this section is a simulated navigation system that is composed of an anytime sensing module, an anytime planning module, a plan execution mechanism and a run-time monitor. The sensing and planning processes may seem oversimplified, but they capture enough of the problem complexity to demonstrate our approach. The primary purpose of this demonstration is to show how the conditional performance profiles of the components are constructed and represented, how the overall performance profile of the system is derived, and how the execution of the anytime components is monitored.

In our demonstration, a robot is situated in a two dimensional environment with random obstacles. The robot does not have an exact map of the environment but it has a vision capability that allows it to create an approximate map. The accuracy of the domain description depends on the time allocated to the vision module. The environment is represented by a matrix of elementary positions. The robot can move between adjacent cells of the matrix at a varying speed which affects the execution time of the plan as well as the energy consumption. When the simulation starts, the robot is presented with a certain task that requires it to move to a particular position and perform a certain job. For example, the robot may need to carry a box from one location to another. Associated with each task is a reward function that determines the value of the task as a function of completion time. The system is designed to control the movement of the robot, that is, determine its direction and speed at each point of time, while maximizing the overall utility. The overall utility depends on the value of the task (a time dependent function), and on the amount of energy consumed in order to complete it.

A run-time monitor has to determine at each point how much time to allocate to the anytime sensing and path planning modules based on such factors as the current location of the robot, the estimated distance to the goal position, the urgency of the task, and the quality of the plan produced so far.

3.1. Anytime sensing

A primary goal of this work has been to examine the applicability of anytime algorithms in sensor interpretation problems. Instead of assuming that sensors produce a perfect domain description or that they have a fixed error, we assume that the data produced by sensors may have variable quality that depends on the time allocated to the process. Moreover, our approach uses a probabilistic description of the effect of sensory error on the other components of the system in order to optimally control the quality of sensing.

One technique that allows sensing quality to be traded for computation time is *variable sampling resolution*. It is based on a simple principle that by increasing the sampling resolution, one increases the amount of data and hence the quality of the information that the sensors provide. At the same time, a higher sampling resolution requires an increase in computational resources to process the data. This principle applies to many different types of sensors, for example, when using a CCD camera, the sampling resolution relates to the overall number of pixels and to the number of bits used per pixel to record image brightness values. Another technique for anytime sensing is based on varying the frequency of sensing. This approach applies in particular to situations in which a sequence of periodic sensor readings are integrated using certain filters (e.g. Kalman filters). High sensing frequency reduces the uncertainty and thus increases the quality of the information produced by the sensor. But, again, higher sensing frequency requires more computational resources.

In our application, sensing is performed by a simulated vision system whose goal is produce a two dimensional map of the local environment of the robot. The domain description identifies certain regions as "obstacles" and others as "free space". The quality of the sensing process measures the probability that an elementary (base-level) position would be identified correctly as an obstacle or free space. For example, when sens-

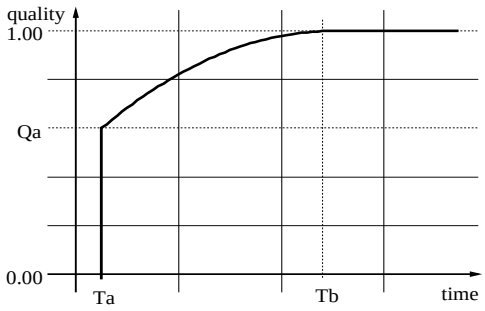


Fig. 2. The performance profile of the vision module

ing quality is 0.98, there is a probability of 0.02 that an elementary position blocked by an obstacle would be labeled as free space or vice versa. We assume that the quality of sensing is uniform within the area in which the sensors are effective. In other words, the quality of sensing is not affected by the robot's position. This assumption, however, is not required by our general model. The assumption only simplifies the simulation program itself.

Figure 2 shows the performance profile of the vision system. It is characterized by four parameters that can be controlled: T_a, T_b, Q_a, and Q_b. T_a is the minimal amount of time needed for the sensing process to produce an initial domain description with quality Q_a. Given a shorter run-time, the sensor does not produce any description of the domain. For a run-time t, $T_a \leq t \leq T_b$, the quality of sensing improves from Q_a to the maximal quality Q_b, which is 1.00 in this example. With real sensors, the maximal possible quality may be less than 1.00 since some small level of noise is normally unavoidable.

In this paper, anytime sensing is treated in a similar way to the *computational* components of the system. However, certain type of sensors, in particular active sensors, require a more complex treatment. The reason is the fact that active sensors may have a significant effect on the state of the environment and thus may have additional influence on the planning process (Zilberstein, 1993). For example, in medical diagnosis, sensing may involve various tests that affect the condition of the patient. The anytime sensing processes that we describe in this paper are used for information gathering only and have no effect on the state of the environment.

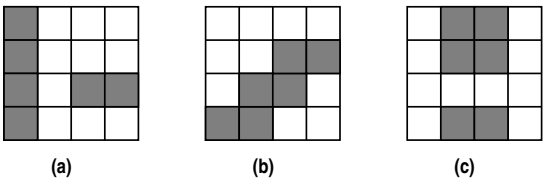


Fig. 3. The "obstacleness" of high-level domain positions represents the proportion of the region covered by obstacles. The obstacleness of the above three regions is 3/8.

3.2. *Anytime abstract planning*

Path planning is performed using an iterative refinement algorithm, similar to the coarse-to-fine search algorithm (Lozano-Pérez and Brooks, 1984). We make the algorithm an anytime algorithm by by varying the abstraction level of the domain description and by allowing for unresolved path segments. As a result, the algorithm can find quickly a low quality plan and then repeatedly refine it by re-planning a segment of the plan in more detail. The rest of this section describes the algorithm and its performance profile.

Abstract description of the domain In our hierarchical (quad trees) representation, the n^{th} level of abstraction corresponds to a certain coarse grid in which every position, (i, j), is an abstraction of a $2^n \times 2^n$ matrix of base-level positions. For example, in the first level of abstraction each position corresponds to a 2×2 matrix of base-level positions. The size of the base-level positions is determined by the minimum size of an obstacle.

Each high level position may include some base-level positions that are blocked by an obstacle and others that are clear. To represent this information, we associate with each high level position a degree of *obstacleness* which is defined by the proportion between the number of blocked base-level positions and the total number of base-level positions. For example, the obstacleness of the three regions shown in Figure 3 is 3/8. Note that obstacleness alone does not provide complete information on the possibility of crossing the region,

```
ATP(start, goal, domain-description)
1       multi-path ← [SEGMENTIZE(start),
            PATH-FINDER(PROJECT(start, L_max),
                PROJECT(goal, L_max),
                domain-description),
            SEGMENTIZE(goal)]
2       REGISTER-RESULT(multi-path)
3       while REFINABLE(multi-path) do
4           REFINE(WORST-SEGMENT(multi-path),
                domain-description)
5           REGISTER-RESULT(multi-path)
6       SIGNAL(TERMINATION)
```

Fig. 4. The anytime planning algorithm

but the lower the degree of obstacleness the easier it is to cross the region.

The anytime planning algorithm The interruptible anytime planner (ATP), shown in Figure 4, constructs a series of plans from *start* to *goal*, whose quality improves over time. It starts with a plan generated by performing best-first search at the highest level of abstraction (L_{max}). Then, it repeatedly refines the plan created so far by selecting the worst segment of the plan, dividing it into two segments (of identical length), and replacing each one of those segments by more detailed plans at a lower abstraction level. The worst segment of the plan is selected according to the degree to which the segment is blocked by obstacles and according to its abstraction level[3]. A special data structure, called a multi-path, is used in order to keep intermediate results. It is a list of successive path segments of arbitrary abstraction level. Note that the length of each segment of an intermediate plan is invariant, since the refinement step cuts the segment into two parts while increasing the resolution. As a result, the run-time of the refinement step is approximately the same for any segment of the plan regardless of its level of abstraction. The algorithm does not include a fixed mechanism (such as a threshold) to terminate the refinement process. This decision is made by the monitor taking into account the particular state of the environment.

The PATH-FINDER is a search procedure that returns the best path between any two positions in the same abstraction level. The path is represented as a list of positions at the same abstraction level. A base-level path must be obstacle-free and hence is a route that the robot can follow. A path at a higher level of abstraction, on the other hand, is the result of best-first search that minimizes the length as well as the obstacleness of the result. The algorithm is similar to A* except that its heuristic evaluation is,

$$f(p) = (1-\alpha)(g(p) + h(p)) + \alpha\ o(p) \quad (1)$$

where $g(p)$ is an estimate of the distance from the initial position, $h(p)$ is an optimistic estimate of the distance to the goal, $o(p)$ is a measure of the obstacleness of the path, and α is the weight of the obstacleness factor. Note that the notion of admissibility, typically a property of A*, is not generally relevant to abstract search techniques since there is no guarantee of getting the shortest base-level path even when the abstract paths are optimal. But the experimental results show that this function is a good heuristic for our domain. Obviously, different planning tasks would require a different evaluation function for finding bad segments of the plan.

Plan execution Abstract plans do not provide a specific path that the robot can follow. In order to follow an abstract plan, the robot must use:

1. A mechanism to map a abstract action into a sequence of base-level actions.
2. A mechanism to repair simple problems that may arise in constructing the sequence of base-level actions.

In our case, an obstacle avoidance procedure was used to perform the two tasks. As long as there exists a path that connects the start and goal positions, the obstacle avoidance procedure can bring the robot to its destination. Therefore, any abstract plan is completable and executable – even when blocked by obstacles. Obstacle avoidance is not a smart navigation method, but it can always substitute for missing details in an abstract plan.

The quality of an abstract plan is determined by the length of the *actual* path that the robot follows when guided by the plan. More specifically, the quality of a plan $\mathcal{P}$ is defined as follows:

$$Quality(\mathcal{P}) = \frac{length(P_{min})}{length(Path(\mathcal{P}))} \quad (2)$$

(a) Level 3 plan

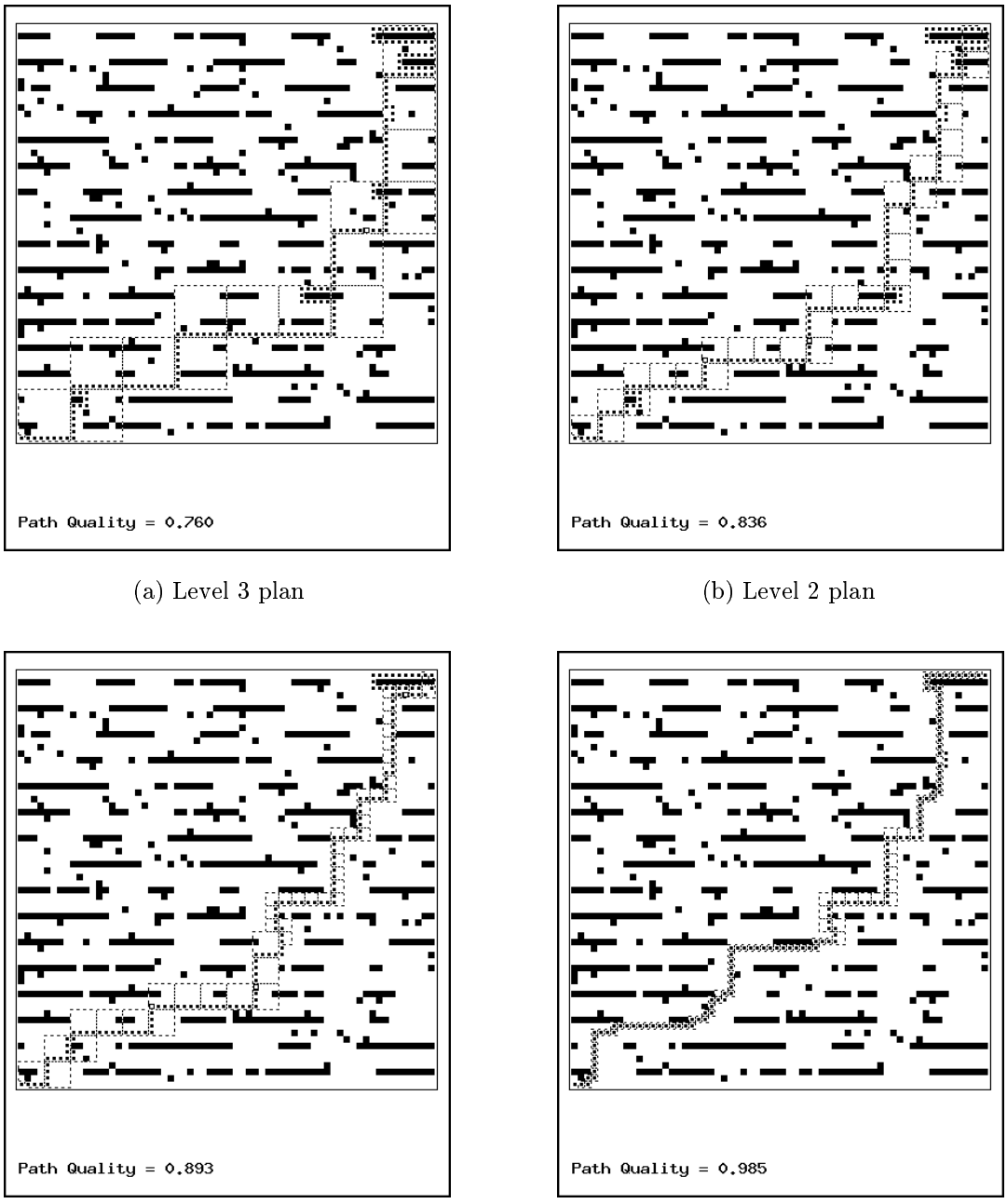
(b) Level 2 plan

(c) Level 2/1 plan

(d) Level 1/0 plan

Fig. 5. Abstract plans with perfect vision

where $length(Path(\mathcal{P}))$ is the length of the actual path that the robot follows when guided by the abstract plan $\mathcal{P}$, and P_{min} is a shortest path at the base-level. Using off-line simulation, we can determine the exact quality of any given plan. When the robot is performing a real-time task, the exact quality of an intermediate plan cannot be determined (since the length of the *actual* path the robot will follow is unknown). But, the quality of plans can be estimated using the performance



(a) Level 3 plan

(b) Level 2 plan

(c) Level 2/1 plan

(d) Level 1/0 plan

Fig. 6. Abstract plans with imperfect vision

profile of the planner and the amount of planning time.

The notion of executable abstract plans – regardless of their arbitrary level of detail – is made possible by using plans as advice that direct the base-level execution mechanism but does not impel a particular behavior. This idea was originated by (Agre and Chapman, 1990) and was experimentally supported by (Gat, 1992). In practice,

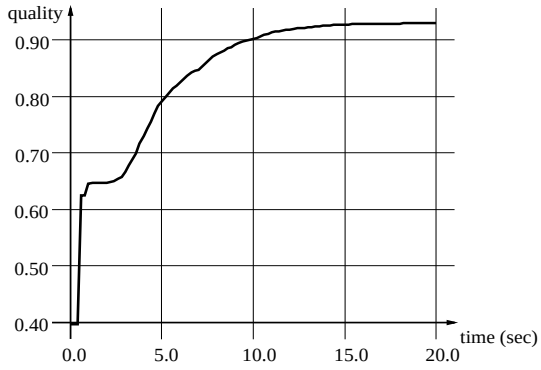


Fig. 7. The performance profile of the planner with perfect vision.

uncertainty makes it hard to use plans except as a guidance mechanism.

Performance with perfect vision We now examine the performance of the abstract planner under the assumption that it is provided with a perfect domain description. Figure 5 shows several intermediate paths generated by the path finder when activated with the start and goal positions being the lower left and upper right corners respectively. Frame (a) shows (by the large squares drawn in broken line) an abstract plan at level 3. The quality of this plan, 0.826, is determined by the length of the route the robot would have followed if guided by this plan (shown in the figure by a heavy broken line) compared to the length of the shortest route. Frames (b), (c) and (d) show how the quality of the plan is gradually improving. Frame (d) shows an abstract plan with segments at levels 0 and 1. Notice that in this example the algorithm generates a plan whose quality is 0.985, almost as good as the quality of the shortest path.

The typical performance of the planner is summarized by its performance profile in Figure 7. The graph shows the expected quality of the plan as a function of run-time. Performance profiles are constructed by collecting statistics from 200 runs of the planner with randomly generated instances of the environment.

When run until completion[4], the anytime planner produces a plan whose expected quality is 0.93. At the same time, its expected completion time is only 27% of the the expected run-time needed to compute the optimal path using the standard A^* algorithm. These figures show that that even without sophisticated monitoring, anytime algorithms offer a much better cost/performance ratio. In other words, the flexibility of the algorithm does not mean that it takes more time to compute a base-level plan. It only means that the algorithm is not *guaranteed* to find the optimal solution. In many autonomous systems, it is undesirable to spend the time needed to compute optimal plans before action can be performed.

Performance with imperfect vision We now turn to examine the effect of vision errors on the quality of planning. We have tested the planning algorithm with vision quality in the range 0.88–1.00 (corresponding to 0–12% error). The following demonstration is based on vision quality of 0.96. This figure is a measure of the sensor's noise level as described in the previous section. The physical domain is identical to the one used in the previous example, however, the map constructed by the vision module is erroneous.

Figure 6 shows several intermediate plans generated by the algorithm and their qualities. Notice that as a result of lower quality of sensing, the quality of the initial plan is only 0.760 compared to 0.826 with perfect vision. Looking at a large set of problem instances, this experiment confirms the basic intuition that as a result of sensory noise (and error in the domain description), the planner produces plans of lower quality.

Based on statistics gathered by running the planning algorithm 200 times (per given vision quality) on randomly generated domains, we derived its conditional performance profile. It describes the expected quality of a plan based on the quality of the domain description and run-time. Figure 8 shows the conditional performance profile. Each curve shows the expected plan quality as a function of run-time for a particular sensing quality. Note that the graph shows consistent and gradual degradation of plan quality as a function of sensing quality.

3.3. Compilation of sensing and planning

Compilation of anytime algorithms is an important technique to build real-time system with anytime algorithms. It allows the programmer to

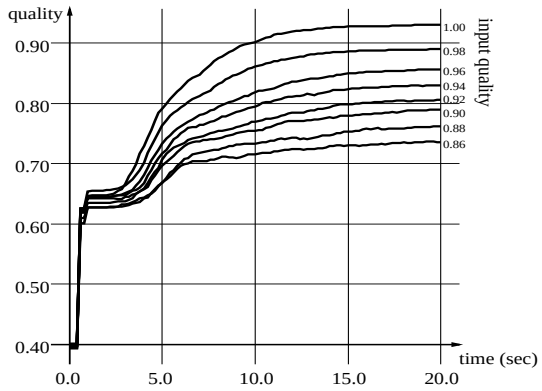


Fig. 8. The conditional performance profile of the planner

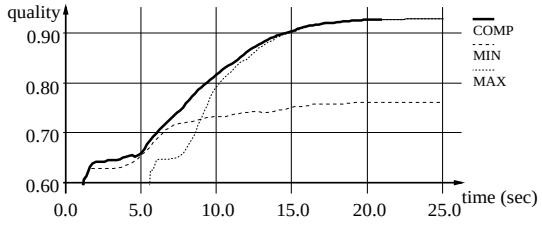


Fig. 9. Compilation of the sensing and planning modules yields an optimal performance profile for the overall system.

compose a system using anytime algorithms as components without dealing directly with the time allocation problem. To explain the compilation process we must first make a distinction between *interruptible* algorithms and *contract* algorithms. Interruptible algorithms produce results of the quality "advertised" by their performance profiles even when interrupted unexpectedly; whereas contract algorithms, although capable of producing results whose quality varies with time allocation, must be given a particular time allocation in advance. The greater freedom of design makes it easier to construct contract algorithms than interruptible ones.

The compilation process is designed to compute off-line the best possible performance profile of a system composed of anytime algorithms. The result of the compilation is a contract algorithm. In those applications where it is necessary to use an interruptible algorithm, the contract algorithm can be transformed into an interruptible one using a construction method presented in (Russell and Zilberstein, 1991). Having made this distinction, we can define compilation as follows:

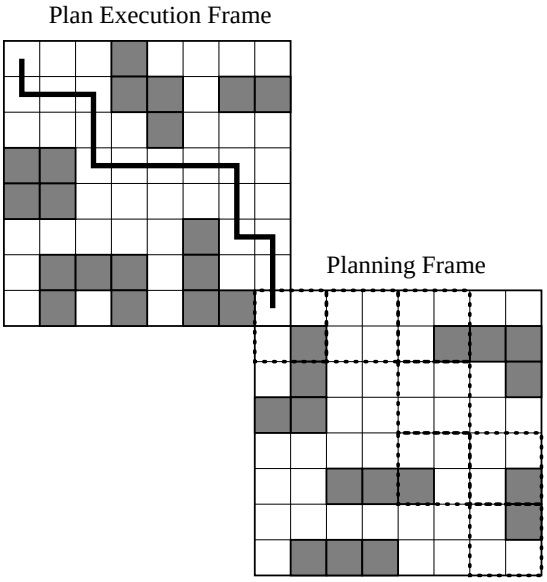


Fig. 10. The navigation system executes an abstract plan while performing anytime sensing and planning for the next execution cycle.

Definition: *Compilation of anytime algorithms is the process of deriving a contract algorithm with an optimal performance profile from a program composed of several anytime algorithms whose conditional performance profiles are given.*

The input to the compiler includes a user defined program composed of anytime algorithms. The compiler also gets a set of *conditional performance profiles* stored in a library. The task of the compiler is to calculate the best performance profile of the program based on the performance profiles of the components. In addition, the compiler generate some information that is used by the run-time monitor to allocate time to the components. The composition of planning and sensing is a simple example of a program composed of anytime algorithms. It is represented by the following program segment:

(find-path Start Goal
(get-domain-description Sensor))

Let $Q_S(t)$ be the performance profile of the sensing module and $Q_P(q,t)$ be the conditional performance profile of the planning module. The compiler calculates the optimal performance profile $Q_C(t)$ by solving the following equation:

$$Q_C(t) = \max_{t_P+t_S=t} \{Q_P(Q_S(t_S), t_P)\} \qquad (3)$$

Figure 9 shows the resulting performance profile. Also shown in that figure are the performance profiles of two other modules: MIN, that allocates to sensing a minimal amount of time, T_a, and MAX, that allocates to sensing a maximal amount of time, T_b. The compiled performance profile is superior to both. It shows that a commitment to a fixed quality of sensory data at design time is a major disadvantage in autonomous systems that operate under variable time pressure.

Further analysis of the above compilation technique shows that when the composed anytime system includes n modules, the compilation problem is NP-complete in the strong sense (Zilberstein, 1993; Zilberstein and Russell, 1995). However, a local compilation technique, that works on a single program structure at a time, significantly improves the efficiency of compilation and is proved to yield optimal performance for a large set of program structures. The local compilation technique applies to situations where the planning and sensing modules themselves are composed of several anytime algorithms.

4. Meta-level control of anytime sensing and planning

To take advantage of their flexibility, systems composed of anytime algorithms require run-time monitoring to determine the optimal time allocation to each component. The compilation process provides the necessary meta-level information to enable efficient run-time monitoring. Various monitoring strategies can be implemented depending on the nature of the task and its time-dependent utility function. The most basic monitoring strategy is to calculate the optimal allocation to the components prior to their activation based on the compiled performance profile of the system. In this section we present an improvement of this approach that involves monitoring of the actual progress made by the anytime components and adjusting the initial contract time accordingly.

4.1. Breaking a large task into small frames

Many autonomous systems do not have global sensing capability that covers the entire domain. Instead, the sensors provide local information with respect to the current position of the system. Therefore, we must distinguish between two levels of planning. Global (coarse) planning is performed using some prior (or accumulated) knowledge about the entire domain, while local planning is performed based on sensory information. The optimization of the long-term behavior of the robot can be performed by dividing a complex task into a series of small sensing, planning and plan execution episodes. We call such episodes *frames*. In mobile robot navigation, the frames can be generated by a global planning process using any combination of the following techniques:

1. A coarse map representing the entire domain.
2. A simple trajectory towards the destination.
3. A partial map of the domain that is based on exploration.

In our application, no prior knowledge of the domain is assumed, so the robot can only use a trajectory towards the destination in order to generate the frames. Figure 10 illustrates this idea.

4.2. Inter-frame optimization

For each frame, the meta-level control has to determine the optimal initial contract time. This decision, referred to as *inter-frame optimization*, is made in the following way:

let

t be the current time (real-time since the beginning of the execution of the task),

f_t be the (estimated) number of frames left at time t for planning and execution, and

e_t be the energy used so far by the robot for plan execution.

Then for any contract time, $t-c$, we can calculate the following domain dependent functions:

$VOT(t, f_t, t_c)$ is the expected value of the task calculated based on the expected completion time. The expected completion time is the current time plus the expected execution time of the plan (whose quality depends on the contract time t_c), plus the expected execution time of the remaining frames.

$COE(e_t, f_t, t_c)$ is the expected cost of energy. It is the sum of the amount of energy used so far, the expected amount of energy to cross the next frame (given that execution time should be t_c), and the estimated amount of energy for the remaining frames.

The best initial contract time, t_c^*, is determined by the following equation:

$$t_c^* = arg \max_{t_c} \{VOT(t, f_t, t_c) - COE(e_t, f_t, t_c)\} \quad (4)$$

Notice that both VOT and COE depend on the particular model of the environment as well as on the performance profile of the system.

4.3. *Intra-frame optimization*

Once an initial contract time is determined, the system starts allocating resources to sensing, planning and plan execution. At the same time it continues to monitor the performance of the anytime modules. This constant monitoring is necessary because of the uncertainty concerning the *actual* quality of plans and the *actual* time necessary to execute them. The purpose of the meta-level control in this phase is to reach an optimal plan quality for the next frame while executing the current plan. This goal is achieved by an adaptive modification of the original contract time by a process referred to as *intra-frame optimization*.

The monitor determines at each point whether planning is ahead of or behind expectations by comparing the estimated plan quality to the quality advertised by the performance profile. It also determines whether plan execution is ahead of or behind expectations by comparing the estimated execution time to the frame contract time. If planning is ahead of expectations and plan execution is behind, the monitor accelerates plan execution by allocating more resources (energy) to plan execution. If planning is behind and plan execution is ahead, the monitor slows down plan execution by reducing resource consumption.

This monitoring strategy can be modified in various ways. For example, one can consider planning more than one frame ahead, when plan execution is slow. Another possibility in this case is to improve parts of the plan that is being executed. The better the monitoring strategy the better it responds to unpredictable situations and the higher the overall utility it achieves. One good measure of the performance of the monitoring component is how close is the overall utility to the best expected result projected initially by the contract time calculation. The results in our domain show that the monitoring strategy described above is sufficient in order to achieve within 4% error the optimal task value that the system computes initially. Obviously, this figure may be different in more realistic implementations. However, we strongly believe that the capability presented in this paper to adaptively modify the resources allocated to sensing and planning processes carry great benefits for real-world applications. We are currently working on the development of monitoring policies that are sensitive to such factors as the variance of the performance of the anytime algorithms, the time-dependent utility of the system, how well the run-time monitor can estimate the quality of the currently available solution, and the cost of monitoring (Hansen and Zilberstein, 1995).

5. The path to real-world applications

The experimental part of this work successfully demonstrates our approach, but the simplicity of the simulated domain may raise some questions regarding the applicability of the technique to real-world applications. Some questions relate to such simplifying assumptions as uniform sensing quality, horizontal rectangular obstacles and a navigation environment with no dead-ends. These assumptions are not likely to hold in many real-world applications. The key question however is not whether the *particular* sensing and planning algorithms that we used will generalize to other domains, but whether *different* anytime algorithms, some of which are readily available, could be used instead in conjunction with our compilation and monitoring techniques. The rest of this section describes several anytime sensor interpretation and planning techniques that can handle more complex domains. Possible violation of our simplifying assumptions is also discussed.

5.1. Anytime sensing in the real-world

A large number of programming techniques can be used to construct anytime algorithms for realistic sensing tasks. RESUN (Carver and Lesser, 1991) is a blackboard-based sensor interpretation system that gathers evidence in an incremental manner to resolve particular sources of uncertainty. Reduction of uncertainty improves the quality of the interpretation over time. Another technique that is widely used in sensor data interpretation is Bayesian reasoning (Pearl, 1988). Several useful algorithms for evaluation of Bayesian networks and influence diagrams are based on incremental accumulation of evidence and offer a time/quality tradeoff. One example called *bounded conditioning* has been developed by (Horvitz *et al.*, 1989). Bounded conditioning monotonically refines the bounds on posterior probabilities in a Bayesian network and converges on the exact probabilities of interest. The approach allows a reasoner to exchange computational resources for incremental reduction of uncertainty. The algorithm solves a probabilistic inference problem in complex Bayesian networks by breaking the problem into a set of mutually exclusive, tractable subproblems and ordering their solutions by the expected effect that each subproblem will have on the final answer. Another technique for anytime evaluation of belief networks using state-space abstraction is presented in (Wellman and Liu, 1994).

Another class of sensing techniques is based on integration, (or *fusion*), of multiple sensor readings using certain filters (e.g. Kalman filters) or certain reasoning techniques (e.g Dempster-Shafer). Such sensing methods can be converted into anytime algorithms by varying the number of sensor readings. For example, Hutchinson and Kak (1989) present a technique for gradually reducing the ambiguity in the world description over time using the Dempster-Shafer theory of evidence. Hager and Mintz (1987) developed a decision-theoretic technique for selecting optimal sensing strategies. They treat sensors as noisy information sources and associate a certain risk function with each sensing operation.

The variable resolution technique that we used in simulation can be used in practice in conjunction with different sensor interpretation techniques. Both processing time and the quality of the domain description (in terms of detail and correctness) will normally increase as the resolution of the raw-data is increased.

5.2. Anytime planning in complex domains

Planning in complex domains is normally based on search and reasoning, both of which can be performed by incremental improvement algorithms. For example, Elkan (1990) present a rule-based, abductive strategy for discovering and revising plausible plans. In his approach, candidate plans are found quickly by allowing them to depend on assumptions. His formalism makes explicit which antecedents of rules have the status of default conditions. Candidate plans are refined incrementally by trying to justify the assumptions on which they depend. This model was implemented by replacing the standard depth-first exploration strategy of Prolog with an iterative-deepening version. The result is an anytime algorithm for incremental approximate planning.

Another approach, *variable precision logic* (Michalski and Winston, 1986), is concerned with problems of reasoning with incomplete information and resource constraints. Variable precision logic offers mechanisms for handling trade-offs between the precision of inferences and the computational efficiency of deriving them. Michalski and Winston address primarily the issue of variable certainty level and employ *censored production rules* as an underlying representational and computational mechanism. These censored production rules are created by augmenting ordinary production rules with an exception condition and are written in the form "if A then B unless C," where C is the exception condition. Systems using censored production rules are free to ignore the exception conditions when resources are tight. Given more time, the exception conditions are examined, lending credibility to high-speed answers or changing them.

A large class of planning and scheduling techniques operate on fully formed plans, gradually improving an existing plan rather than generating one from scratch (Alterman, 1988). Our abstract planner falls under this category. Unfortunately, there are no guarantees that gradual refinement

will *always* improve the quality of a plan. In domains that include dead-ends (which cannot always be detected at the higher abstraction levels), the planner will have to backtrack in some situations. But our technique does not rely on monotonicity of quality improvement with respect to *every* problem instance. It relies on the improvement of the *expected* quality over many problem instances – a property that is easier to maintain.

Finally, abstraction techniques have been used in several AI planning systems. State abstraction was introduced by ABSTRIPS (Sacerdoti, 1974). More recent systems such as ALPINE (Knoblock, 1990) can automatically generate abstraction hierarchies for robot planning. PABLO (Christensen, 1990) is a nonlinear planner that reasons hierarchically by generating abstract predictions. It can produce a sequence of executable actions should it be interrupted before the final plan has been completed.

5.3. *Increasing the level of parallelism in the system*

In our demonstration, sensor interpretation and planning are performed by a single processor in parallel to plan execution. In principle, however, sensor interpretation could be performed by a separate processor. This additional degree of parallelism will require a different monitoring scheme, but it does not contradict the basic premises of our approach: that sensing should be implemented as an anytime process and that it should be controlled based on its expected effect on the system's performance.

The compilation technique remains essential in systems that perform sensing and planning in parallel, since both the sensing and planning components could be composed of several anytime algorithms. For example, a speech recognition module can first classify the speaker (in terms of gender, accent, and other features that help calibrate the interpretation module), then generate possible interpretations, and finally perform linguistic verification and determine the best interpretation. Each one of these activities can be implemented using one or more anytime algorithms.

6. Conclusion

We have presented a new approach to sensing and planning in autonomous systems which improves their capability to operate in real-time with limited computational resources. Our approach is based on developing the sensing and planning components of the system as anytime algorithms and representing their performance using conditional performance profiles. The control of the anytime components of the system is implemented using off-line compilation and run-time monitoring. An implementation of the model was presented in the area of mobile robot navigation. Our approach offers several advantages in addressing computational costs in autonomous systems: it is an optimizing rather than satisficing method; it allows complex planning to be used in real-time robotic systems; and it helps construct systems when resource availability is unknown at design time.

In addition to the sensing and planning algorithms presented in this paper, there is a growing number of anytime algorithms that are readily available and that can handle complex tasks. When combined with our compilation and monitoring techniques, these algorithms have major advantages over traditional methods whose output quality is predetermined.

Further work on real-world applications is needed in order to evaluate our approach. By improving the various components of the system, we aim at constructing a general, flexible mechanism for developing self-optimizing autonomous systems whose perception, decision making and action are implemented as anytime modules. This approach offers a more realistic theoretical foundation for robot planning by addressing the problems of uncertainty, limited computational power, and imprecise sensing.

Acknowledgements

Part of this work was performed at UC Berkeley in collaboration with my graduate advisor, Stuart Russell. The author wishes to thank the anonymous reviewers for providing useful comments and insights. Support for this work was provided in part by the National Science Foundation un-

der grant IRI-9409827 and by a Faculty Research Grant from the University of Massachusetts.

Notes

1. The term "sensing" in this paper refers to the computational task of perceptual data interpretation rather then the mechanical process of sensing.
2. A comprehensive survey of classical planning systems can be found in (Drummond and Tate, 1989).
3. The actual score function to determine the worst segment is the sum of the abstraction level and the number of positions that are blocked with respect to the plan moving direction.
4. Running the anytime planner until completion involves calculating an initial path at the highest abstraction level and refining all its segments until a base-level plan is produced.

References

Agre, P. E. and Chapman, D. 1990. What are plans for? *Robotics and Autonomous Systems*, 6:17–34.

Alterman, R. 1988. Adaptive planning. *Cognitive Science*, 12:393–422.

Boddy, M. and Dean, T. L. 1994. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–285.

Bresina, J. and Drummond, M. 1990. Integrating planning and reaction. In J. Hendler, Ed., *AAAI Spring Symposium on Planning in Uncertain, Unpredictable or Changing Environments*, Stanford, California, pp. 24–28.

Carver, N. and Lesser, V. R. 1991. A new framework for sensor interpretation: Planning to resolve sources of uncertainty. In *Proc. 9th National Conference on Artificial Intelligence*, pp. 724–731.

Christensen, J. 1990. A hierarchical planner that creates its own hierarchies. In *Proc. 8th National Conference on Artificial Intelligence*, Boston, Massachusetts, pp. 1004–1009.

Cohen, P. R. 1995. *Empirical Methods for Artificial Intelligence*. MIT Press: Cambridge, Massachusetts.

D'Ambrosio, B. 1989. Resource bounded agents in an uncertain world. In *Proc. IJCAI-89 Workshop on Real-Time Artificial Intelligence Problems*, Detroit, Michigan.

Dean, T. L. and Boddy, M. 1988. An analysis of time-dependent planning. In *Proc. 7th National Conference on Artificial Intelligence*, Minneapolis, Minnesota, pp. 49–54.

Drummond, M. and Tate, A. 1989. AI Planning: A Tutorial and Review. Technical Report AIAI-TR-30, AI Applications Institute, University of Edinburgh.

Elkan, C. 1990. Incremental, approximate planning. In *Proc. 8th National Conference on Artificial Intelligence*, Boston, Massachusetts, pp. 145–150.

Fikes, R. E., Hart, P. E. and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288.

Fox, B. R. and Kempf, K. G. 1985. Opportunistic scheduling for robotic assembly. In *IEEE International Conference on Robotics and Automation*, pp. 880–889.

Fraichard, T. and Laugier, C. 1991. On-line reactive planning for a non-holonomic mobile in a dynamic world. In *IEEE International Conference on Robotics and Automation*, Sacramento, California, pp. 432–437.

Gat, E. 1992. Integration planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proc. 10th National Conference on Artificial Intelligence*, San Jose, California, pp. 809–815.

Hager, G. and Mintz, M. 1987. Searching for information. In *Proc. AAAI Workshop on Spatial Reasoning and Multi-Sensor Fusion*, Los Altos, California: Morgan Kaufmann, pp. 313–322.

Hammond, K. J. 1990. Explaining and repairing plans that fail. *Artificial Intelligence*, 45:173–228.

Hanks, S. 1990. Practical temporal projection. In *Proc. 8th National Conference on Artificial Intelligence*, Boston, Massachusetts, pp. 158–163.

Hanks, S., Pollack, M. and Cohen, P. R. 1993. Benchmarks, Testbeds, Controlled Experimentation, and the Design of Agent Architectures. *AI Magazine*, 14(4):17–42.

Hansen, E. A. and Zilberstein, S. 1995. Monitoring the progress of anytime algorithms. In *Proc. IJCAI-95 Workshop on Anytime Algorithms and Deliberation Scheduling*, Montreal, Canada, pp. 28–32.

Hayes-Roth, B. *et al.* 1992. Guardian: A prototype intelligent agent for intensive-care monitoring. *Artificial Intelligence in Medicine*, 4:165–185.

Hendler, J., Tate, A. and Drummond, M. 1990. AI planning: Systems and techniques. *AI Magazine*, 11(2):61–77.

Howard, R. A. 1966. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, SSC-2(1):22–26.

Howe, A. E., Hart, D. M. and Cohen, P. R. 1990. Addressing real-time constraints in the design of autonomous agents. In the *Journal of Real-Time Systems*, 2(1/2):81–97.

Horvitz, E. J. 1989. Reasoning about beliefs and actions under computational resource constraints. In *Uncertainty in Artificial Intelligence*, Vol. 3, Eds. L.N. Kanal, T.S. Levitt and J.F. Lemmer, pp. 301–324, North-Holland: Amsterdam.

Horvitz, E. J., Suermondt, H. J. and Cooper, G. F. 1989. Bounded conditioning: Flexible inference for decision under scarce resources. In *Proc. Workshop on Uncertainty in Artificial Intelligence*, Windsor, Ontario, pp. 182–193.

Hutchinson, S. A. and Kak, A. C. 1989. Planning sensing strategies in a robot work cell with multi-sensor capabilities. *IEEE Transactions on Robotics and Automation*, 5(6):765–783.

Ingrand, F. F., Georgeff, M. P. and Rao, A. S. 1992. An architecture for real-time reasoning and system control. *IEEE Expert*, December 1992, pp. 34–44.

Knoblock, C. A. 1990. Learning abstraction hierarchies for problem solving. In *Proc. 8th National Conference on Artificial Intelligence*, Boston, Massachusetts, pp. 923–928.

Lozano-Pérez, T. and Brooks, R. A. 1984. An approach to automatic robot programming. In *Solid Modeling by Computers*, M. S. Pickett and J. W. Boyse, Eds. pp. 293–327, Plenum: New York.

Lyons, D. M. and Hendriks, A. J. 1992. A practical approach to integrating reaction and deliberation. In *Proc. First International Conference on AI Planning Systems*, College Park, Maryland, pp 153–162.

Michalski, R. S. and Winston, P. H. 1986. Variable precision logic. *Artificial Intelligence* 29(2):121–146.

Musliner, D. J., Durfee, E. H. and Shin, K. G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence*, 74:83–127.

Newell, A. and Simon, H. A. 1976. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 12:113–126.

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

Russell, S. J. and Wefald, E. H. 1991. *Do the Right Thing: Studies in limited rationality*. MIT Press: Cambridge, Massachusetts.

Russell, S. J. and Zilberstein, S. 1991. Composing real-time systems. In *Proc. 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, pp. 212–217.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135.

Sacerdoti, E. D. 1977. *A Structure for Plans and Behavior*. American Elsevier: New York.

Simon, H. A. 1982. *Models of Bounded Rationality, Volume 2*. MIT Press: Cambridge, Massachusetts.

Tate, A. 1995. Representing plans as a set of constraints – the <I-N-OVA> model. *ACM SIGART Bulletin*, 6(1):26–32.

Wellman, M. P. and Liu, C.-L. 1994. State-space abstraction for anytime evaluation of probabilistic networks. In *Proc. 10th Conference on Uncertainty in Artificial Intelligence*, Seattle, Washington, pp. 567–574.

Xiaodong, X. and Bekey, G. A. 1988. Sroma: An adaptive scheduler for robotic assembly systems. In *IEEE International Conference on Robotics and Automation*, Philadelphia, Penn, pp. 1282–1287.

Zilberstein, S. 1993. *Operational Rationality Through Compilation of Anytime Algorithms*. Ph.D Dissertation, Computer Science Division, University of California, Berkeley, CA.

Zilberstein, S. 1995. Optimizing decision quality with contract algorithms. In *Proc. 14th International Joint Conference on Artificial Intelligence*, Montreal, Canada, pp. 1576–1582.

Zilberstein, S. and Russell, S. J. 1993. Anytime sensing, planning and action: A practical model for robot control. In *Proc. 13th International Joint Conference on Artificial Intelligence*, Chambery, France, pp. 1402–1407.

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

Shlomo Zilberstein is an Assistant Professor of Computer Science at the University of Massachusetts Amherst. He received his B.A. in Computer Science from the Technion, Israel Institute of Technology, graduating *summa cum laude* and first in his class in 1981, and his PhD in Computer Science from the University of California at Berkeley in 1993. In 1986, as a research scientist at Shalev Systems in Israel, he developed heuristic search algorithms for geometric pattern matching that contributed to the development of an innovative robotic system for automated carving of precious stones. Professor Zilberstein's current research interests include resource-bounded reasoning, autonomous agent architectures, real-time problem solving, and reasoning under uncertainty. At the University of Massachusetts, he has developed the most advanced programming environment and scheduling algorithms for construction, composition, and control of anytime algorithms. Professor Zilberstein is the recipient of Israel's Security Prize (1992), and an NSF Research Initiation Award (1994). He is the co-Chair of the 1996 AAAI Fall Symposium on Flexible Computation in Intelligent Systems.