Omnipotence Without Omniscience:
Efficient Sensor Management for Planning

Keith Golden Oren Etzioni Daniel Weld*
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{kgolden, etzioni, weld}@cs.washington.edu

Abstract

Classical planners have traditionally made the closed world assumption — facts absent from the planner's world model are false. Incomplete-information planners make the open world assumption — the truth value of a fact absent from the planner's model is unknown, and must be sensed. The open world assumption leads to two difficulties: (1) How can the planner determine the scope of a universally quantified goal? (2) When is a sensory action *redundant*, yielding information already known to the planner?

This paper describes the fully-implemented XII planner, which solves both problems by representing and reasoning about *local closed world information* (`LCW`). We report on experiments utilizing our UNIX softbot (software robot) which demonstrate that `LCW` can substantially improve the softbot's performance by eliminating redundant information gathering.

Introduction

Classical planners (*e.g.*, (Chapman 1987)) presuppose correct and complete information about the world. Although recent work has sketched a number of algorithms for planning with incomplete information (*e.g.*, (Ambros-Ingerson & Steel 1988; Olawsky & Gini 1990; Krebsbach, Olawsky, & Gini 1992; Peot & Smith 1992; Etzioni *et al.* 1992; Etzioni, Lesh, & Segal 1993; Genesereth & Nourbakhsh 1993)), substantial problems remain before these planners can be applied to real-world domains. Since the presence of incomplete information invalidates the Closed World Assumption, an agent cannot deduce that a fact is false based on its absence from the agent's world model. This leads to two challenges:

- **Satisfying Universally Quantified Goals:** Goals of the form "Move all widgets to the warehouse" or "Make all files in `/tex` write-protected" are common in real-world domains. Classical planners such as PRODIGY (Minton *et al.* 1989) or UCPOP (Penberthy & Weld 1992) reduce universally quantified goals to the set of ground instances of the goal, and satisfy each instance in turn. But how can a planner compute this set in the absence of complete information? How can the planner be certain that it has moved *all* the widgets or protected *all* the relevant files?
- **Avoiding Redundant Sensing:** Should the planner insert a sensory action (*e.g.*, scan with the camera, or the UNIX command `ls`) into its plan? Or is the action *redundant*, yielding information already known to the planner? Since satisfying the preconditions of a sensory action can require arbitrary planning, the cost of redundant sensing is potentially unbounded and quite large in practice (see the Experimental Results section).

This paper reports on the fully-implemented XII planner[1] which addresses these challenges. We allow incomplete information in the initial conditions, and uncertainty in the effects,[2] but assume the information that *is* known is correct, and that there are no exogenous events. XII's planning algorithm is based on UCPOP (Penberthy & Weld 1992), but XII interleaves planning and execution and, unlike UCPOP, does not make the closed world assumption.

The next section introduces the central concept underlying XII's operation: *local closed world information* (`LCW`). In the following section we describe how incorporating `LCW` in a planner enables it to solve universally quantified goals in the presence of incomplete information. We then show how the same mechanism addresses the problem of redundant information gath-

*We thank Denise Draper, Steve Hanks, Terrance Goan, Nick Kushmerick, Neal Lesh, Rich Segal, and Mike Williamson for helpful discussions. This research was funded in part by Office of Naval Research Grants 90-J-1904 and 92-J-1946, and by National Science Foundation Grants IRI-8957302, IRI-9211045, and IRI-9357772. Golden is supported in part by a UniForum Research Award.

[1] XII stands for "eXecution and Incomplete Information."

[2] All effects of operators must be specified, but XII supports a three-valued logic which allows us to specify a limited form of uncertainty in the effects.

ering. The Experimental Results section demonstrates the advantages of eliminating redundant sensing. We conclude with a discussion of related and future work.

Local Closed World Information

Our agent's model of the world is represented as a set of ground literals stored in a database $\mathcal{D}_{\mathbf{M}}$. Since $\mathcal{D}_{\mathbf{M}}$ is incomplete, the closed world assumption is invalid — the agent cannot automatically infer that any sentence absent from $\mathcal{D}_{\mathbf{M}}$ is false. Thus, the agent is forced to represent false facts explicitly — as $\mathcal{D}_{\mathbf{M}}$ sentences with the truth value `F`.

In practice, many sensing actions return exhaustive information which warrants limited or "local" closed world information. For example, the UNIX `ls -a` command lists *all* files in a given directory. After executing `ls -a`, it is not enough for the agent to record that `paper.tex` and `proofs.tex` are in `/tex` because, in addition, the agent knows that *no other* files are in that directory. Note that the agent is not making a closed world *assumption*. Rather, the agent has executed an action that yields closed world *information*.

Although the agent now knows that `parent.dir(foo, /tex)` is false, it is impractical for the agent to store this information explicitly in $\mathcal{D}_{\mathbf{M}}$, since there is an infinite number of such sentences. Instead, the agent represents closed world information explicitly in a meta-level database, $\mathcal{D}_{\mathbf{C}}$, containing formulas of the form `LCW`(Φ) that record *where* the agent has closed world information. `LCW`(Φ) means that for all variable substitutions θ, if the ground sentence $\Phi\theta$ is true in the world then $\Phi\theta$ is represented in $\mathcal{D}_{\mathbf{M}}$. For instance, we represent the fact that $\mathcal{D}_{\mathbf{M}}$ contains all the files in `/tex` with `LCW(parent.dir(`f`,/tex))` and that it contains the length of all such files with `LCW(parent.dir(`f`,/tex)`$\wedge$`length(`f`,`l`))`.

When asked whether an atomic sentence Φ is true, the agent first checks to see if Φ is in $\mathcal{D}_{\mathbf{M}}$. If it is, then the agent returns the truth value (`T` or `F`) associated with the sentence. However, if $\Phi \notin \mathcal{D}_{\mathbf{M}}$ then Φ could be either `F` or `U` (unknown). To resolve this ambiguity, the agent checks whether $\mathcal{D}_{\mathbf{C}}$ entails `LCW`(Φ). If so, Φ is `F`, otherwise it is `U`.

`LCW` Updates

As the agent is informed of the changes to the external world — through its own actions or through the actions of other agents — it can gain and lose `LCW`; these changes must be recorded in $\mathcal{D}_{\mathbf{C}}$. We assume here, and throughout, the absence of hidden exogenous events that invalidate XII's information. In other words, we assume that the rate of change in the world is slower than the rate at which XII plans and executes. This is the standard assumption of correct information made by most planners.[3]

When XII executes an action which ensures that $\mathcal{D}_{\mathbf{M}}$ contains all instances of Φ that are true in the world, XII adds a formula `LCW`(Φ) to $\mathcal{D}_{\mathbf{C}}$. For example, XII is given an axiom stating that each file has a unique word count. Thus, executing the UNIX command `wc paper.tex` adds the formula `LCW(word.count(paper.tex,`c`))` to $\mathcal{D}_{\mathbf{C}}$ as well as adding the actual length (*e.g.*, `word.count (paper.tex,42)`) to $\mathcal{D}_{\mathbf{M}}$. Since the `LS` operator (Figure 1) has a universally quantified effect, executing `ls -a /tex` yields `LCW(parent.dir(`f`,/tex))`.

It would be cumbersome if the author of each operator were forced to list its `LCW` effects. In fact, this is unnecessary. XII automatically elaborates operator schemata with `LCW` effects. For example, the following effects are automatically added to the `LS` operator:

`LCW(parent.dir(`f_1`, ?d))`
`LCW(parent.dir(`f_2`, ?d)` $\wedge$ `(filename` f_2`,` p_2`))`
`LCW(parent.dir(`f_3`, ?d)` $\wedge$ `(pathname` f_3`,` n_2`))`

where the subscripted symbols indicate new unique variables, and `?d` is a parameter that will be substituted with a constant value at run-time. This compilation process takes time linear in the length of the operator schemata and the number of unique-value axioms (Golden, Etzioni, & Weld 1994).

Observational effects (*e.g.*, those of `LS`) can only create `LCW`, but causal effects can both create and destroy `LCW`.[4] For example, deleting all files in `/tex` *provides* complete information on the contents of the directory regardless of what the agent knew previously. Compressing a file in `/tex`, on the other hand, makes the length of the file unknown,[5] thus *invalidating* previously obtained `LCW` on the lengths of all files in that directory.

The theory behind `LCW` is complex; (Etzioni, Golden, & Weld 1994) defines `LCW` formally, explains the connection to circumscription, and presents a set of tractable update rules for the case of conjunctive `LCW` formulas. In this paper, we show how to incorporate conjunctive `LCW` into a least commitment planner and argue that this addresses the challenges described in the introduction: satisfying universally quantified goals and avoiding redundant sensing.

Universally quantified goals

In this section we explain how XII utilizes `LCW` to satisfy universally quantified goals. Traditionally, planners that have dealt with goals of the form "Forall v of type `t` make $\Delta(v)$ true" have done so by expanding the goal into a universally-ground, conjunctive goal called the

[3] In fact, the softbot relaxes this assumption by associating expiration times with beliefs in $\mathcal{D}_{\mathbf{M}}$ and $\mathcal{D}_{\mathbf{C}}$ and by recovering from errors that result from incorrect information. However, a discussion of this mechanism is beyond the scope of this paper.

[4] XII operator schemata explicitly distinguish between causal effects (that change the state of the external world) and observational effects (that only change the state of XII's model) as explained in (Etzioni *et al.* 1992).

[5] This is written in the operator effects as `(cause (length ?f ?l) U)`.

```
(defoperator LS ((directory ?d) (path ?dp))
  (precond (and (satisfy (current.shell csh))
                (satisfy (current.dir ?d))
                (satisfy (protection ?d readable))
                (find-out (pathname ?d ?dp))))
  (effect (forall ((file !f) :in (parent.dir $ ?d))
               (exists ((path !p) (name !n))
                   (and (observe (parent.dir !f ?d))
                        (observe (pathname !f !p))
                        (observe (filename !f !n))))))
  (interface (execute-unix-command ("ls -a"))
             (sense-func (!f !n !p) (ls-sense ?dp))))
```

Figure 1: **UNIX operator.** The XII `LS` operator lists all files in the current directory. The last two lines specify the information needed to interface to UNIX. The first of these says to output the string "`ls -a`" to the UNIX shell. The second says to use the function `ls-sense` to translate the output of the shell into a set of bindings for the variables `!f`, `!n` and `!p`.

universal base (Weld 1994). The universal base of such a formula equals the conjunction $\Delta_1 \wedge \ldots \wedge \Delta_n$ in which the Δ_is correspond to each possible interpretation of $\Delta(v)$ under the universe of discourse, $\{C_1, \ldots, C_n\}$, *i.e.* the possible objects of type `t` (Genesereth & Nilsson 1987, p. 10). In each Δ_i, all references to v have been replaced with the constant C_i. For example, suppose that `pf` denotes the type corresponding to the files in the directory `/papers` and that there are two such files: $C_1 =$ `a.dvi` and $C_2 =$ `b.dvi`. Then the universal base of "Forall f of type `pf` make `printed`(f) true" is `printed(a.dvi)`$\wedge$`printed(b.dvi)`.

A classical planner can satisfy $\forall$ goals by subgoaling to achieve the universal base, but this strategy relies on the closed world assumption. Only by assuming that all members of the universe of discourse are known (*i.e.*, represented in the model) can one be confident that the universal base is equivalent to the $\forall$ goal. Since the presence of incomplete information invalidates the closed world assumption, the XII planner uses two new mechanisms for satisfying $\forall$ goals:

1. Sometimes it is possible to directly support a $\forall$ goal with a $\forall$ effect, without expanding the universal base. For example, given the goal of having all files in a directory group readable, XII can simply execute `chmod g+r *`; it doesn't need to know which files (if any) are in the directory.

2. Alternatively, XII can subgoal on obtaining `LCW` on the type Φ_i of each universal variable v_i in the goal. Once XII has `LCW`(Φ_i), the universe of discourse for v_i is completely represented in its world model. At this point XII generates the universal base and subgoals on achieving it. Note that this strategy differs from the classical case since it involves interleaved planning and execution. Given the goal of printing all files in `/papers`, XII would plan and *execute* an `ls -a` command, then plan to print each file it found, and finally execute that plan.

For completeness, XII also considers combinations of these mechanisms to solve a single $\forall$ goal, via a technique called *partitioning*; see (Golden, Etzioni, & Weld 1994) for details.[6] In the remainder of this section we explain these two mechanisms in more detail.

Protecting $\forall$ links

In the simplest case, XII can use a universally quantified effect to directly support a universally quantified goal. However, $\forall$ goals, like ordinary goals, can get clobbered by subgoal interactions; to avoid this, XII uses an extension of the *causal link* (McAllester & Rosenblitt 1991) mechanism to protect $\forall$ goals. A causal link is a triple, written $A_p \xrightarrow{G} A_c$, where G is a goal, A_p is the step that produces G and A_c is the step that consumes G. We refer to G as the *label* of the link. When XII supports a $\forall$ goal directly (*i.e.*, without expanding into the universal base) it creates a link whose label, G, is a universally quantified formula (instead of the traditional literal); we call such links "$\forall$ links." In general, a link is *threatened* when some other step, A_t, has an effect that possibly *interferes* with G and A_t can possibly be executed between A_p and A_c. For normal links, interference is defined as having an effect that unifies with $\neg G$. Such an effect also threatens a $\forall$ link, but $\forall$ links are additionally threatened by effects that possibly add an object to the quantifier's universe of discourse. For example, if XII adds a `chmod g+r *` step to achieve the goal of having all files in a directory group readable, the link would be threatened by a step which moved a new file (possibly unreadable) into the directory. Threats to $\forall$ links can be handled using the same techniques used to resolve ordinary threats: *de-*

[6] Note also that the classical universal base mechanism requires that a type's universe be static and finite. XII correctly handles dynamic universes. Furthermore, XII's policy of linking to $\forall$ effects handles infinite universes, but this is not of practical import.

motion, *promotion*, and *confrontation*.[7] Additionally, the following rule applies.

- **Protect forall:** Given a link $A_p \xrightarrow{G} A_c$ in which $G = \forall_{\texttt{type1}} x\ \texttt{S}(x)$ and the type `type1` equals $\{x | \texttt{P}(x) \wedge \texttt{Q}(x) \wedge \ldots \wedge \texttt{Z}(x)\}$ and a threat A_t with effect `P(foo)`, subgoal on achieving $\texttt{S(foo)} \vee \neg\texttt{Q(foo)} \vee \ldots \vee \neg\texttt{Z(foo)}$ by the time A_c is executed.

For example, suppose a $\forall$ link recording the condition that all files in `/tex` be group readable is threatened by step A_t, which creates a new file, `new.tex`. This threat can be handled by subgoaling to ensure that `new.tex` is either group readable or not in directory `/tex`.

Protecting LCW

The other way to satisfy a $\forall$ goal is to subgoal on obtaining `LCW`, and then satisfy each subgoal in the universal base. However, since `LCW` goals can also get clobbered by subgoal interactions, XII has to ensure that actions introduced for sibling goals don't cause the agent to *lose* `LCW`. For example, given the goal of finding the lengths all files in `/papers`, XII might execute `ls -la`. But if it then compresses a file in `/papers`, it no longer has `LCW` on all the lengths.

To avoid these interactions, we use `LCW` links which are like standard causal links except that they are labeled with a conjunctive `LCW` formula. Since $\texttt{LCW(P}(x) \wedge \texttt{Q}(x)\texttt{)}$ asserts knowledge of `P` and `Q` over all the members of the set $\{x \mid \texttt{P}(x) \wedge \texttt{Q}(x)\}$, an `LCW` link is threatened when information about a member of the set is possibly lost or a new member, for which the required information may be unknown, is possibly added to the set. We refer to these two cases as *information loss* and *domain growth*, respectively, and discuss them at length below. Like threats to ordinary causal links, threats to `LCW` links can be handled using *demotion*, *promotion*, and *confrontation*. In addition, threats due to information loss can be resolved with a new technique called *shrinking*, while domain-growth threats can be defused either by shrinking or by a method called *enlarging*.

Information Loss We say that A_t threatens $A_p \xrightarrow{G} A_c$ with information loss if $G = \texttt{LCW(}P_1 \wedge \ldots \wedge P_n\texttt{)}$, A_t possibly comes between A_p and A_c, and A_t contains an effect that makes R unknown, for some R that unifies with some P_i in G. For example, suppose XII's plan has a link $A_p \xrightarrow{H} A_c$ in which

$$H = \texttt{LCW(parent.dir(}f\texttt{,/papers)} \wedge \texttt{length(}f, n\texttt{))}$$

indicating that the link is protecting the subgoal of knowing the lengths of all the files in directory `/papers`. If XII now adds a step which has the action `compress myfile.txt`, then the new step threatens the link, since `compress` has the effect of making the length of `myfile.txt` unknown.

- **Shrinking `LCW`:** Given a link with condition $\texttt{LCW(P}(x) \wedge \texttt{Q}(x) \wedge \ldots \wedge \texttt{Z}(x)\texttt{)}$ and threat causing `P(foo)` to be unknown (or true), XII can protect the link by subgoaling to achieve $\neg\texttt{Q(foo)} \vee \ldots \vee \neg\texttt{Z(foo)}$[8] at the time that the link's consumer is executed. For example, compressing `myfile.txt` threatens the link $A_p \xrightarrow{H} A_c$ described above, because if `myfile.txt` is in directory `/papers`, then the lengths of *all* the files in `/papers` are no longer known. However, if `parent.dir(myfile.txt,/papers)` is false then the threat goes away.

Domain Growth We say that A_t threatens $A_p \xrightarrow{G} A_c$ with domain growth if $G = \texttt{LCW(}P_1 \wedge \ldots \wedge P_n\texttt{)}$, A_t possibly comes between A_p and A_c, and A_t contains an effect that makes R true, for some R that unifies with some P_i. For the example above in which the link $A_p \xrightarrow{H} A_c$ protects `LCW` on the length of every file in `/papers`, addition of a step which moved a new file into `/papers` would result in a domain-growth threat, since the agent might not know the length of the new file. Such threats can be resolved by the following.

- **Shrinking `LCW`** (described above): If XII has `LCW` on the lengths of all postscript files in `/tex`, then moving a file into `/tex` threatens `LCW`. However, if the file isn't a postscript file, `LCW` is not lost.
- **Enlarging `LCW`:** Given a link with condition $\texttt{LCW(P}(x) \wedge \texttt{Q}(x) \wedge \ldots \wedge \texttt{Z}(x)\texttt{)}$ and threat causing `P(foo)` to be true, XII can protect the link by subgoaling to achieve $\texttt{LCW(Q(foo)} \wedge \ldots \wedge \texttt{Z(foo))}$ at the time that the link's consumer is executed. For example, moving a new file `xii.tex` into directory `/papers` threatens the link $A_p \xrightarrow{H} A_c$ described above, because the length of `xii.tex` may be unknown. The threat can be resolved by observing the length of `xii.tex`.

Note that an effect which makes some P_i *false* does *not* pose a threat to the link! This corresponds to an action that moves a file *out* of `/papers` — it's not a problem because one still knows the lengths of all the files that remain.

Discussion

Given the new ways of resolving goals and threats, how much larger is the XII search space than that of UCPOP? XII has an additional type of open condition: the `LCW` goal. Since `LCW` goals, being conjunctive, can be solved using a combination of `LCW` effects, this would seem

[7] The first two techniques order the threatening action before the link's producer or after its consumer. Confrontation works when the threatening effect is conditional; the link is protected by subgoaling on the negation of the threat's antecedent (Penberthy & Weld 1992).

[8] Note the difference between shrinking and protecting a $\forall$ link. Unlike the $\forall$ link case, shrinking does not have a disjunct corresponding to `S(foo)`.

to result in a large branching factor. In practice, this is not the case, because LCW goals tend to be short. In XII, $\forall$ goals can be solved by two additional mechanisms: $\forall$ links and partitioning. The addition of $\forall$ links increases the branching factor, but often results in shorter plans, and thus less search. Partitioning, in the worst case, has a branching factor equal to the number of predicates in the domain theory, but in practice, XII partitions only on predicates that could potentially be useful. The number of such predicates is typically small. Nonetheless, partitioning can still be expensive, and search control heuristics that limit partitioning are useful.

XII also adds three new ways of resolving threats; none of them apply to the standard causal links supported by UCPOP, so the branching factor for threats to these links is unchanged. For the new links supported by XII, the branching factor for threat resolution is increased by at most k, where k is the number of conjuncts in the LCW condition or in the universe of the $\forall$ condition. Presently, $k \leq 3$ in all of our UNIX operators.

The question of completeness for XII is difficult to answer, because the notion of completeness is ill-defined in an environment that involves execution. Given the existence of irreversible actions, such as `rm`, visiting part of the search space may make a previously solvable goal unsolvable. For example, the dilemma posed in Stockton's classic story "The lady, or the tiger?" (Stockton 1888) is solvable; opening the correct door will result in winning the game. However, the protagonist cannot determine what is behind a door without first opening it, and opening the wrong door means losing the game (and his life). By the formal definition of completeness, a complete planner must produce a plan guaranteed to win the game, since such a plan exists, but clearly such a guarantee is impossible. In future work, we hope to define a notion of completeness that is meaningful in such domains, and prove that XII conforms to that definition.

Redundant Information Gathering

The problem of redundant information gathering is best illustrated by a simple example. Suppose that we ask a softbot to find an Alaska Airlines flight from Seattle to San Francisco, cheaper than \$80. The softbot can contact travel agents and airlines, which are listed in various telephone directories (cf (Levy, Sagiv, & Srivastava 1994)). In general, the separate information sources will contain overlapping information. A given travel agent might provide information on all domestic flights within a given price range, while an airline will provide information on all flights it offers. Suppose that the softbot has contacted Alaska Airlines and failed to find a fare less that \$80. Unless it knows that contacting Alaska directly provides exhaustive information on Alaska flights, it will be forced to backtrack and pursue its other options. To make matters worse, a travel agency might be listed in multiple directories, and may have several phone numbers. Thus, exploring all possible plans to exhaustion would involve contacting the same travel agency multiple times. In general, once *any* exhaustive information gathering action is successfully executed, additional information gathering actions are redundant.[9]

The magnitude of the redundant sensing problem should not be underestimated (see Table 1 for empirical measurements). Furthermore, the problem of redundant sensing is both domain and planner independent; when trying alternative ways of satisfying a goal, a planner is forced to consider *every* sensory action at its disposal. Since each action has preconditions, and there are multiple ways of achieving these preconditions, the amount of wasted work can increase exponentially with the length of the information-gathering plan — unless the planner has some criterion for deciding which actions will not yield new information.

Fortunately, LCW is just that: *An agent should not execute, or plan to execute, observational actions (or actions in service of observational actions) to support a goal when it has* LCW *on that goal.* In fact, a single LCW formula can service a wide range of goals. For example, `LCW(parent.dir(`f`,/tex))`, which results from executing `ls -a` in `/tex`, indicates that XII knows all the files in `/tex`. Thus, it can satisfy *any* goal of the form "Find out whether some file x is in `/tex`" by examining its world model — no information gathering is necessary. In addition, XII can combine LCW formulas to avoid redundant information gathering on composite goals. For example, if XII knows all the files owned by Smith, and all the files in `/tex`, then it can satisfy the conjunctive goal "Give me all the files in `/tex` that are owned by Smith" by consulting its model.

XII utilizes LCW in three ways:

- **Execution pruning:** when XII is about to execute an observational step A_p which only supports links labeled with goals $G_1, \ldots, G_n$, XII checks whether LCW(G_i) holds for all i. If so, A_p is redundant and XII does not execute it. Instead, it replaces all links from A_p with links from the model ($\mathcal{D}_{\mathbf{M}}$), since any information that could be obtained by executing A_p is already recorded in $\mathcal{D}_{\mathbf{M}}$. This simple test prevents XII from executing some redundant information gathering steps. However, XII might still do redundant planning (and execution!) to satisfy A_p's preconditions, and the preconditions' preconditions, *etc.*
- **Option pruning:** to address this problem, XII tests for LCW when it computes the set of actions $\mathcal{A}$ that could *potentially* support a goal G. If LCW(G) holds, XII can omit *observational* actions from the set.[10]

[9] We cannot simply associate exactly one sensory action with each goal, *a priori*, because the agent may fail to satisfy that action's preconditions — in which case trying a different sensory action *is* warranted.

[10] Since XII can subsequently lose LCW due to information

Problem Set	Planner Version	Plans Examined	Steps Executed	Total Time
22 problems, 13 solvable	With LCW	420	55	109
	Without LCW	3707	724	966
14 problems, all solvable	With LCW	373	55	94
	Without LCW	1002	140	160

Table 1: Reasoning about local closed world information (LCW) improves the performance of the softbot on two suites of UNIX problems. Times are in CPU seconds on a Sun Microsystems SPARC-10. Without LCW inference the softbot fails to complete eight of the problems in the first set, and one of the problems in the second set, before reaching a 100 CPU second time bound. With LCW, the softbot completes all the problems. The mean size of $\mathcal{D}_C$ (the softbot's store of LCW information) is 155 formulas. The maximum size is 167.

- **Post hoc pruning:** XII may gain LCW(G) after $\mathcal{A}$ is computed (so option pruning did not apply) but considerably before any of the steps in $\mathcal{A}$ are about to be executed (so execution pruning is not yet applicable). This occurs when executing an action yields LCW(G), or when a binding constraint is asserted that constrains one or more of the variables in G. For instance, XII may not have `LCW(parent.dir(`f`,`d`))`, but once d is instantiated to, say, `/tex`, `LCW(parent.dir(`f`,/tex))` can result in significant pruning.

In concert, these pruning techniques are surprisingly powerful, as demonstrated in the next section.

Experimental Results

The reader might question whether redundant sensing is as common as we suggest, or wonder whether the cost of utilizing the LCW machinery outweighs the benefit from pruning XII's search space. To address such concerns, and to empirically evaluate our LCW implementation, we plugged XII into the UNIX softbot (Etzioni, Lesh, & Segal 1993), providing XII with operator descriptions of standard UNIX commands, and enabling it to actually execute the commands by sending (and receiving) strings from the UNIX shell. We gave the softbot a sequence of goals and measured its performance with and without LCW. Table 1 quantifies the impact of the LCW mechanism on the softbot's behavior. We found that our LCW machinery yielded a significant performance gain for the softbot.

In this experiment, the softbot's goals consisted of simple file searches (*e.g.*, find a file with word count greater than 5000, containing the string "theorem," *etc.*) and relocations. The actions executed in the tests include `mv` (which can destroy LCW), observational actions such as `ls, wc` and `grep`, and more. Each experiment was started with $\mathcal{D}_M$ and $\mathcal{D}_C$ initialized empty, but they were not purged between problems; so for each problem the softbot benefited from the information gained in solving the previous problems.

Maintaining $\mathcal{D}_C$ introduced less than 15% overhead per plan explored, and reduced the number of plans explored substantially. In addition, the plans produced were often considerably shorter, since redundant sensing steps were eliminated. Without LCW, the softbot performed 16 redundant `ls` operations, and 6 redundant `pwd`s in a "typical" file search. With LCW, on the other hand, the softbot performed no redundant sensing. Furthermore, when faced with unachievable goals, the softbot with LCW inference was able to fail quickly; however, without LCW it conducted a massive search, executing many redundant sensing operations in a forlorn hope of observing something that would satisfy the goal. While much more experimentation is necessary, these experiments suggest that local closed world reasoning, as implemented in XII, has the potential to substantially improve performance in a real-world domain.

Related work

XII is based on the UCPOP algorithm (Penberthy & Weld 1992). The algorithm we used for interleaving planning and execution closely follows IPEM, by Ambros-Ingerson and Steel (Ambros-Ingerson & Steel 1988). Our action language borrows both from ADL (Pednault 1986) and UWL (Etzioni *et al.* 1992).

Our research on LCW has its roots in the SOCRATES planner, where the problem of redundant information gathering was initially discovered (Etzioni & Lesh 1993). Like XII, SOCRATES utilized the UNIX domain as its testbed, supported the UWL representation, and interleaved planning with execution. In addition, SOCRATES supported a restricted representation of LCW, which enabled it to avoid redundant information gathering in many cases. Our advances over SOCRATES include the ability to satisfy universally quantified goals, and the machinery for automatically generating LCW effects and for detecting threats to LCW links.

Genesereth and Nourbakhsh (Genesereth & Nourbakhsh 1993) share our goal of avoiding redundant information gathering, but do so using radically different mechanisms, and in the context of state-space search.

loss or domain growth (described in the previous section), it has to record this pruning decision and recompute the options for G if LCW(G) is lost. Doing this in an efficient but sound manner is complex — see (Golden, Etzioni, & Weld 1994) for the details.

They derive completeness-preserving rules for pruning the search as well as rules for terminating planning and beginning execution. However, they do not have notions that correspond to LCW, a database like $\mathcal{D}_{\mathbf{C}}$, or our threat resolution techniques.

Other researchers have investigated alternative approaches for planning with incomplete information (see (Olawsky & Gini 1990) for a nice taxonomy). Contingent planners (Warren 1976; Schoppers 1987; Peot & Smith 1992) seek to exhaustively enumerate alternative courses of action; while this strategy is appropriate in critical domains with irreversible actions, the exponential increase in planning time is daunting. Decision theory provides an elegant framework for computing the value of information; however, although work in this direction is promising, many challenges remain (Wellman 1993). Our approach sacrifices the elegance of a probabilistic framework to achieve a complete implementation able to tackle practical problems.

Conclusion

This paper describes the fully-implemented XII planner which uses *local closed world information* (LCW) to handle universally quantified goals and to avoid the problem of redundant sensing. Our technical innovations include the LCW machinery (effects, goals, and novel techniques for resolving threats to LCW links) and the LCW-based pruning techniques which solve the problem of redundant information gathering. As demonstrated in Table 1, the savings engendered by LCW can be quite large in the UNIX domain. Although our experiments, and illustrative examples, are drawn from the UNIX domain, we emphasize that the notion of LCW, and the techniques introduced in XII, are domain independent. In future work, we plan to measure the costs and benefits of LCW in other domains, and to remove the assumption of correct information made by the XII planner.

References

Ambros-Ingerson, J., and Steel, S. 1988. Integrating planning, execution, and monitoring. In *Proc. 7th Nat. Conf. on A.I.*, 735–740.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Etzioni, O., and Lesh, N. 1993. Planning with incomplete information in the UNIX domain. In *Working Notes of the AAAI Spring Symposium: Foundations of Automatic Planning: The Classical Approach and Beyond*, 24–28. Menlo Park, CA: AAAI Press.

Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., and Williamson, M. 1992. An Approach to Planning with Incomplete Information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning.* Available via FTP from pub/ai/ at cs.washington.edu.

Etzioni, O., Golden, K., and Weld, D. 1994. Tractable closed-world reasoning with updates. In *Proc. 4th Int. Conf. on Principles of Knowledge Representation and Reasoning.*

Etzioni, O., Lesh, N., and Segal, R. 1993. Building softbots for UNIX (preliminary report). Technical Report 93-09-01, University of Washington. Available via anonymous FTP from pub/etzioni/softbots/ at cs.washington.edu.

Genesereth, M., and Nilsson, N. 1987. *Logical Foundations of Artificial Intelligence.* Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Genesereth, M., and Nourbakhsh, I. 1993. Time-saving tips for problem solving with incomplete information. In *Proc. 11th Nat. Conf. on A.I.*, 724–730.

Golden, K., Etzioni, O., and Weld, D. 1994. XII: Planning for Universal Quantification and Incomplete Information. Technical report, University of Washington, Department of Computer Science and Engineering. Available via FTP from pub/ai/ at cs.washington.edu.

Krebsbach, K., Olawsky, D., and Gini, M. 1992. An empirical study of sensing and defaulting in planning. In *Proc. 1st Int. Conf. on A.I. Planning Systems*, 136–144.

Levy, A., Sagiv, Y., and Srivastava, D. 1994. Towards efficient information gathering agents. In *Working Notes of the AAAI Spring Symposium: Software Agents*, 64–70. Menlo Park, CA: AAAI Press.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. on A.I.*, 634–639.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., and Gil, Y. 1989. Explanation-based learning: A problem-solving perspective. *Artificial Intelligence* 40:63–118. Available as technical report CMU-CS-89-103.

Olawsky, D., and Gini, M. 1990. Deferred planning and sensor use. In *Proceedings, DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control.* Morgan Kaufmann.

Pednault, E. 1986. *Toward a Mathematical Theory of Plan Synthesis.* Ph.D. Dissertation, Stanford University.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 103–114. Available via FTP from pub/ai/ at cs.washington.edu.

Peot, M., and Smith, D. 1992. Conditional Nonlinear Planning. In *Proc. 1st Int. Conf. on A.I. Planning Systems*, 189–197.

Schoppers, M. 1987. Universal plans for reactive robots in unpredictable environments. In *Proceedings of IJCAI-87*, 1039–1046.

Stockton, F. R. 1888. *The lady, or the tiger? and other stories.* New York: Charles Scribner's Sons.

Warren, D. 1976. Generating Conditional Plans and Programs. In *Proceedings of AISB Summer Conference*, 344–354.

Weld, D. 1994. An introduction to least-commitment planning. *AI Magazine.* Available via FTP from pub/ai/ at cs.washington.edu.

Wellman, M. 1993. Challenges for decision-theoretic planning. In *Proceedings of the AAAI 1993 Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond.*