Omnipotence Without Omniscience:
Efficient Sensor Management for Software Agents

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 the UNIX softbot (software robot) which demonstrate that LCW can substantially improve the softbot’s performance by eliminating redundant information gathering.

1 Introduction
Classical planners (e.g., [2]) presuppose correct and complete information about the world. Although recent work has sketched a number of algorithms for planning with incomplete information (e.g., [1, 13, 10, 16, 4, 6, 8]), 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 [12] or UCPOP [15] 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 section 5).

This paper reports on the fully-implemented xII planner which addresses these challenges. We allow incomplete information, but assume the information that is known is correct. xII’s planning algorithm is based on UCPOP [15], but xII interleaves planning and execution (following IPEM [1]) and, unlike UCPOP, does not make the closed world assumption.

Section 2 introduces the central concept underlying xII’s operation: local closed world information (LCW). In section 3 we describe how incorporating LCW in a planner enables it to solve universally quantified goals in the presence of incomplete information. We then (section 4) show how the same mechanism addresses the problem of redundant information gathering. Section 5 gives experimental results which demonstrate the advantages of eliminating redundant sensing. We conclude with a discussion of related and future work.

2 Local Closed World Information
Our agent’s model of the world is represented as a set of ground literals stored in a database D_M. Since D_M is incomplete, the closed world assumption is invalid — the agent cannot automatically infer that any sentence absent from D_M is false. Thus, the agent is forced to represent false facts explicitly — as D_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, scanning with a TV camera shows all objects in a room, and the UNIX ls command lists all files in a given directory. After executing ls, 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 \( D_M \), since there is an infinite number of such sentences. Instead, the agent represents closed world information explicitly in a meta-level database, \( D_C \), containing sentences of the form LCW(\( \Phi \)) that record where the agent has closed world information. LCW(\( \Phi \)) means that for all variable substitutions \( \theta \), if the ground sentence \( \Phi \theta \) is true in the world then \( \Phi \theta \) is represented in \( D_M \). For instance, we represent the fact that \( D_M \) contains all the files in \( /\text{tex} \) with LCW(parent.dir(/,\text{tex})) and that it contains the length of all such files with LCW(parent.dir(/,\text{tex}) \& length(f,l)).

When asked whether an atomic sentence \( \Phi \) is true, the agent first checks to see if \( \Phi \) is in \( D_M \). If it is, then the agent returns the truth value (\( T \) or \( F \)) associated with the sentence. However, if \( \Phi \notin D_M \) then \( \Phi \) could be either \( F \) or \( U \) (unknown). To resolve this ambiguity, the agent checks whether \( D_C \) entails LCW(\( \Phi \)). If so, \( \Phi \) is \( F \), otherwise it is \( U \).

### 2.1 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 \( D_C \). We assume here, and throughout, the absence of hidden exogenous events that invalidate \( X_{11} \)'s information. In other words, we assume that the rate of change in the world is slower than the rate at which \( X_{11} \) plans and executes. This is the standard assumption of correct information made by most planners.

When \( X_{11} \) executes an operator which ensures that \( D_M \) contains all instances of \( \Phi \) that are true in the world, \( X_{11} \) adds a sentence LCW(\( \Phi \)) to \( D_C \). For example, \( X_{11} \) is given an axiom stating that each file has a unique word count. Thus, executing the UNIX command \( \text{wc paper.tex} \) adds the sentence LCW(word.count(paper.tex,c)) to \( D_C \) as well as adding the actual length (\( e.g., \) word.count(paper.tex,42)) to \( D_M \). Since the \( ls \) command has a universally quantified effect, executing \( ls /\text{tex} \) yields LCW(parent.dir(/,\text{tex})).

It would be cumbersome if the author of each operator were forced to list its LCW effects. In fact, this is unnecessary. \( X_{11} \) automatically elaborates operator schemata with LCW effects. Even in the worst case, this compilation process takes time linear in the length of the operator schemata and the number of unique-value axioms.

Observational effects (\( e.g., \) those of \( ls \)) can only create LCW, but causal effects can both create and destroy LCW. For example, deleting all files in \( /\text{tex} \) provides complete information on the contents of the directory regardless of what the agent knew previously. Compressing a file in \( /\text{tex} \), on the other hand, invalidates previously obtained LCW of the lengths of all files in that directory.

The theory behind LCW is complex; \[3\] defines LCW formally, explains the connection to circumscription, and presents a set of tractable update rules for the case of conjunctive LCW sentences. In this paper, we show how to incorporate conjunctive LCW into a least commitment planner and argue that this addresses the challenges described in section 1: satisfying universally quantified goals and avoiding redundant sensing.

### 3 Universally quantified goals

In this section we explain how \( X_{11} \) 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 universal base. The universal base of such a sentence equals the conjunction \( \Delta_1 \wedge \ldots \wedge \Delta_n \) in which the \( \Delta_i \)s 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 \( t \) [7, p. 10]. In each \( \Delta_i \), 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 \( /\text{papers} \) 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 \( \text{ printed(a.dvi) \& printed(b.dvi) } \).

A classical planner can satisfy \( V \) 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 \( V \) goal. Since the presence of incomplete information invalidates the closed world assumption, the \( X_{11} \) planner uses two new mechanisms for satisfying \( V \) goals:

1. Sometimes it is possible to directly support a \( V \) goal with a \( V \) effect, without expanding the universal base. For example, given the goal of having all files in a directory group readable, \( X_{11} \) can simply execute \( \text{ chmod g+r } \); it doesn’t need to know which files (if any) are in the directory. Section 3.1 elaborates this option.

2. Alternatively, \( X_{11} \) can subgoal on obtaining LCW on the type \( \Phi_i(v_i) \) of each universal variable \( v_i \) in the goal. Once \( X_{11} \) has LCW(\( \Phi_i \)), the universe of discourse for \( v_i \) is completely represented in its world model. At this point \( X_{11} \) 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 \( /\text{papers} \), \( X_{11} \) would plan and execute an \( ls \) command, then plan to print each file causal effects (that change the state of the external world) and observational effects (that only change the state of \( X_{11} \)'s model) as explained in \[4\].
it found, and then execute that plan. Section 3.2 describes this option in detail.

For completeness, XII also considers combinations of these mechanisms to solve a single V goal; see [9] for details. In the remainder of this section we explain these two techniques in more detail.

3.1 Protecting V links
In the simplest case, XII can use a universally quantified effect to directly support a universally quantified goal. However, V goals, like ordinary goals, can get clobbered by subgoal interactions; to avoid this, XII uses an extension of the causal link [11] mechanism used for protecting other goals. A causal link is a triple, written $A_p \rightarrow \neg G \rightarrow A_c$, 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 a V goal directly (i.e., without expanding into the universal base) it creates a link whose label, $G$, is a universally quantified sentence (instead of the traditional literal); we call such links "V 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 V link, but V links are also threatened by effects that possibly add an object to the quantifier's universe of discourse. For example, if XII adds a $\text{chmod} g+r$ 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 V links can be handled using the same techniques used to resolve ordinary threats: demotion, promotion, and confrontation. Additionally, the following rule applies.

- Protect forall:
  Given a link $A_p \rightarrow \neg G \rightarrow A_c$ in which $G = \forall x S(x)$ and the type $\text{type1}$ equals $\{x|P(x) \land Q(x) \land R(x)\}$ and a threat $A_t$ with effect $P(\text{foo})$, subgoal on achieving $S(\text{foo}) \lor Q(\text{foo}) \lor \neg R(\text{foo})$ by the time $A_c$ is executed.

  For example, suppose a V 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.

3.2 Protecting LCW
The other way to satisfy a V goal is to subgoal on obtaining LCW, and then expand 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, the agent might execute $\text{wc} \ast$. 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 sentence. Since LCW($P(x) \land Q(x)$) asserts knowledge of $P$ and $Q$ over all the members of the set $\{x | P(x) \land Q(x)\}$, 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 by a method called enlarging.

Information Loss We say that $A_t$ threatens $A_p \rightarrow A_c$ with information loss if $G = \text{LCW}(P_1 \land \ldots \land P_n)$, $A_t$ 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 $A_p \rightarrow A_c$ in which

$$H = \text{LCW}(\text{parent.dir}(f,/papers) \land \text{word.count}(f,n))$$

indicating that the link is protecting the subgoal of knowing the word counts of all the files in directory /papers. If XII now adds a step which has the action $\text{compress} \text{myfile.txt}$, then the new step threatens the link, since $\text{compress}$ has the effect of making the word count of myfile.txt unknown.

- Shrinking LCW: Given a link with condition LCW($P(x) \land Q(x) \land R(x)$) and threat causing $P(\text{foo})$ to be unknown, XII can protect the link by subgoaling to achieve $\neg Q(\text{foo}) \lor \neg R(\text{foo})$ at the time that the link's consumer is executed. For example, compressing myfile.txt threatens the link $A_p \rightarrow A_c$ described above, because if myfile.txt is in directory /papers, then the word counts 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 $A_p \rightarrow A_c$ with domain growth if $G = \text{LCW}(P_1 \land \ldots \land P_n)$, $A_t$ 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 $A_p \rightarrow A_c$ protects LCW on the word count of every file in /papers, addition of a step which moved a new file into /papers would result in a domain-growth threat, 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, the agent might execute $\text{wc} \ast$. 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 sentence. Since LCW($P(x) \land Q(x)$) asserts knowledge of $P$ and $Q$ over all the members of the set $\{x | P(x) \land Q(x)\}$, 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 by a method called enlarging.

Information Loss We say that $A_t$ threatens $A_p \rightarrow A_c$ with information loss if $G = \text{LCW}(P_1 \land \ldots \land P_n)$, $A_t$ 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 $A_p \rightarrow A_c$ in which

$$H = \text{LCW}(\text{parent.dir}(f,/papers) \land \text{word.count}(f,n))$$

indicating that the link is protecting the subgoal of knowing the word counts of all the files in directory /papers. If XII now adds a step which has the action $\text{compress} \text{myfile.txt}$, then the new step threatens the link, since $\text{compress}$ has the effect of making the word count of myfile.txt unknown.

- Shrinking LCW: Given a link with condition LCW($P(x) \land Q(x) \land R(x)$) and threat causing $P(\text{foo})$ to be unknown, XII can protect the link by subgoaling to achieve $\neg Q(\text{foo}) \lor \neg R(\text{foo})$ at the time that the link's consumer is executed. For example, compressing myfile.txt threatens the link $A_p \rightarrow A_c$ described above, because if myfile.txt is in directory /papers, then the word counts 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 $A_p \rightarrow A_c$ with domain growth if $G = \text{LCW}(P_1 \land \ldots \land P_n)$, $A_t$ 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 $A_p \rightarrow A_c$ protects LCW on the word count of every file in /papers, addition of a step which moved a new file into /papers would result in a domain-growth threat, 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, the agent might execute $\text{wc} \ast$. But if it then compresses a file in /papers, it no longer has LCW on all the lengths.
since the agent might not know the length of the new
file.

- **Enlarging LCW:** Given a link with condition
  \(LCW(P(x) \land Q(x) \lor R(x))\) and threat causing \(P(\text{foo})\)
to be true, \(x_H\) can protect the link by subgoaling to
achieve \(LCW(Q(\text{foo}) \land R(\text{foo}))\) at the time that the
link's consumer is executed. For example, moving a
new file \(x_{II<?>>\text{tex}\) into directory /papers
threatens
the link \(A_p H A_\text{tex}\), described above, because the word
count of \(x_{II<?>>\text{tex}\) may be unknown. The threat can
be resolved by observing the word count of \(x_{II?><\text{tex}\).

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.

4 Redundant Information Gathering

The problem of redundant information gathering is
best illustrated by a simple example. Suppose that
we ask a softbot to find out whether \(\text{paper}.\text{tex}\) is in
the directory /\text{tex}, and that when it executes \(ls -a\),
the softbot discovers that the file is not there. Unless
it knew that \(ls -a\) has provided exhaustive information
about the contents of the directory, the softbot
would backtrack and consider alternative ways of satisfi-
ing the goal, such as executing \(ls \text{ paper}.\text{tex}\) or \(find\)
in the directory, not realizing that these information-
gathering actions will never succeed. In general,
one *any* exhaustive information gathering action is
successfully executed, additional information gathering
actions are redundant.\(^7\)

The magnitude of the redundant sensing problem
should not be underestimated (see Section 5 for em-
pirical measurements). Furthermore, the problem of re-
dundant sensing is both domain and planner inde-
pendent; 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 precondi-
tions, the amount of wasted work can increase exponen-
tially with the length of the information-gathering plan — unless the planner has some criterion for de-
ciding 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 sentence can serve a wide range of goals. For ex-
ample, \(LCW(\text{parent} . \text{dir}(f,/\text{tex}))\), which results from
executing \(ls \text{ in }/\text{tex}\), indicates that \(x_{II}\) knows all the
files in /\text{tex}. Thus, it can satisfy any goal of the form
"Find out whether some file \(x\) is in /\text{tex}\" by examine-
ing its model world — no information gathering is
necessary. In addition, \(x_{II}\) can combine LCW sen-
centes to avoid redundant information gathering on com-
posite goals. For example, if \(x_{II}\) knows all the files owned
by Smith, and all the files in /\text{tex}, then it can satisfy
the conjunctive goal "Give me all the files in /\text{tex} that
are owned by Smith" by consulting its model.

\(x_{II}\) utilizes LCW in three ways:

- **Execution pruning:** when \(x_{II}\) is about to execute
an observational step \(A_p\) which only supports links
labeled with goals \(G_1, \ldots, G_n\), \(x_{II}\) checks whether
\(LCW(G_i)\) holds for all \(i\). If so, \(A_p\) is redundant and
\(x_{II}\) does not execute it. Instead, it replaces all links
from \(A_p\) with links from the model (\(\mathcal{D_M}\)), since any
information that could be obtained by executing \(A_p\)
is already recorded in \(\mathcal{D_M}\). This simple test pre-
vents \(x_{II}\) from executing some redundant informa-
tion gathering steps. However, \(x_{II}\) might still do re-
dundant planning (and execution!) to satisfy \(A_p\)'s
preconditions, and the preconditions' preconditions,
etc.

- **Option pruning:** to address this problem, \(x_{II}\) tests
for LCW when it computes the set of actions \(A\) that
could *potentially* support a goal \(G\). If LCW(G) holds,
\(x_{II}\) can omit *observational* actions from the set.\(^8\)

- **Post hoc pruning:** \(x_{II}\) may gain LCW(G) af-
after \(A\) is computed (so option pruning did not ap-
ply) but considerably before any of the steps in \(A\)
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 con-
straint is asserted that constrains one or more of
the variables in \(G\). For instance, \(x_{II}\) may not have
\(LCW(\text{parent} . \text{dir}(f,d))\), but once \(d\) is instan-
tiated to, say, /\text{tex}, \(LCW(\text{parent} . \text{dir}(f,/\text{tex}))\) can result
in significant pruning.

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

5 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
due from pruning \(x_{II}\)'s search space. To address such
concerns, and to empirically evaluate our LCW imple-
mentation, we plugged \(x_{II}\) into the UNIX softbot [6],
providing \(x_{II}\) 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 mech-
anism 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

\(^7\)We cannot simply associate exactly one sensory action
with each goal, *a priori*, because the agent may fail to sat-
ify that action's preconditions — in which case trying a
different sensory action is warranted.

\(^8\)Since \(x_{II}\) can subsequently lose LCW due to information
loss or domain growth (Section 3.2), it has to record this
pruning decision and recompute the options for \(G\) once
LCW(G) is lost. Doing this in an efficient but sound man-
er is actually quite complex — see [9] for the details.
include `mv` (which can destroy LCW), observational actions such as `ls`, `wc` and `grep`, and more. Each experiment was started with $\mathcal{P}_M$ and $\mathcal{P}_C$ initialized empty, but they were not purged between problems; so for each problem the softbot benefits from the information gained in solving the previous problems.

<table>
<thead>
<tr>
<th>PROBLEM SET</th>
<th>USE LCW?</th>
<th>PLANS EXAM'D</th>
<th>STEPS EXEC'D</th>
<th>TOT. TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>22 PROBLEMS, 13 SOLVABLE</td>
<td>Yes</td>
<td>420</td>
<td>55</td>
<td>109</td>
</tr>
<tr>
<td>14 PROBLEMS, ALL SOLVABLE</td>
<td>Yes</td>
<td>373</td>
<td>55</td>
<td>94</td>
</tr>
<tr>
<td>14 PROBLEMS, ALL SOLVABLE</td>
<td>No</td>
<td>1002</td>
<td>140</td>
<td>160</td>
</tr>
</tbody>
</table>

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 $\mathcal{P}_C$ (the softbot’s store of LCW information) is 155 sentences. The maximum size is 167.

Maintaining $\mathcal{P}_C$ introduced less than 15% overhead per plan explored, and reduced the number of plans explored substantially. In addition, the plans produced are often considerably shorter, since redundant sensing steps are eliminated. Without LCW, the softbot performed 16 redundant `ls` operations, and 6 redundant `pwd` 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 closed world reasoning, as implemented in xII, has the potential to substantially improve performance in a real-world domain.

6 Related work

xII is based on the ucPoP algorithm [15]. The algorithm we used for interleaving planning and execution closely follows IPM, by Ambros-Ingerson and Steel [1]. Our action language borrows both from ADL [14] and UWL [4].

The problem of redundant information gathering was addressed by the SOCRATES planner, discussed in [5]. Like xII, SOCRATES utilized the UNIX domain as its testbed, supported the UWL representation and interleaved planning with execution. 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.

Ganesereth and Nourbakhsh [8] share our goal of avoiding redundant information gathering, but do so using radically different mechanisms, and in the context of state-space search. 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 $\mathcal{P}_C$, or our threat resolution techniques.

Other researchers have investigated alternative approaches for planning with incomplete information (see [13] for a nice taxonomy). Contingent planners [18, 17, 16] 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 [19]. Our approach sacrifices the elegance of a probabilistic framework to achieve a complete implementation able to tackle practical problems.

7 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) described in Section 3, and the LCW-based pruning techniques described in Section 4. As demonstrated by 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


