Building a Planner for Information Gathering: A Report from the Trenches

Craig A. Knoblock
Information Sciences Institute and
Department of Computer Science
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, USA
knoblock@isi.edu

Abstract

Information gathering requires locating and integrating data from a set of distributed information sources. These sources may contain overlapping data and can come from different types of sources, including traditional databases, knowledge bases, programs, and Web pages. In this paper we focus on the problem of how to apply a general-purpose planner to produce plans for information gathering. We identify the critical functionality of the basic planner, describe how the information gathering problem can be cast as a planning problem, and present our approach to efficiently generating high-quality plans in this application domain. The resulting information gathering planner is used as the query processor in the SIMS information mediator, which is being applied to provide access to data for transportation logistics and trauma care. We present empirical results in the transportation domain to demonstrate that this planner can efficiently produce information gathering plans on a set of example queries that were provided with the databases.

Introduction

This paper examines the issues in applying a general-purpose planner to the information gathering task. Information gathering involves locating and integrating information to answer queries from a set of heterogeneous and distributed information sources. An information gathering plan includes the source for the information, the specific operations that are to be performed on the data, and the order in which the operations are to be performed. The general problem is quite hard since there are a very large number of ways a query can be processed. The choice of plans is critical since the cost of executing different plans for the same query can range from a few milliseconds to a few years or longer.

Consider the example query, shown in Table 1, from a transportation domain, which requests the names of all seaports that have channels that can accommodate small tankers. A plan to answer this query is shown in Figure 1. In this example, the plan partitions the given query such that in parallel the seaport information is retrieved from the **SEAPORT** information source and the tanker information is retrieved from the **ASSETS** information source. The results of those two queries are then joined in the local system. The plan then retrieves the information from the **COUNTRY** source and compares it to the intermediate results of the other subplan. Once the system generates the final set of data, it is sent to the output.

```
(retrieve (?port-name)
   (:and (seaport ?sport)
         (port-name ?sport ?port-name)
         (country-name ?sport "Saudi Arabia"
         (channel-of ?sport ?channel)
         (channel-depth ?channel ?depth)
         (transport-ship ?ship)
         (vehicle-type-name ?ship "Small Tanker")
         (max-draft ?ship ?draft)
         (< ?draft ?depth)))
```

Table 1: An Example Query

The information gathering task has a number of characteristics that make it an interesting planning problem. First, the basic problem is to generate a sequence of actions to efficiently retrieve the requested set of information. Unlike much of the previous work on planning, a *satisficing* solution is not sufficient since the quality of the final plan is an important consideration. Second, the search space is large, but there are fewer interactions than there are in domains such as the blocksworld, so it is possible to search a large space of possible plans. Third, the plans produced in this domain can be directly executed, which makes it possible to consider issues of planning and execution without the use of a simulator. Finally, and most importantly, this is a real problem.

This problem is similar to the problem of generating a query access plan for a database in that both require producing a sequence of actions to produce the requested data. However, it differs from query access planning in several ways. The first difference is that query access planning only requires generating a plan for how to process the data, while information gathering also requires identifying the relevant sources for use in producing the requested information. The sec-

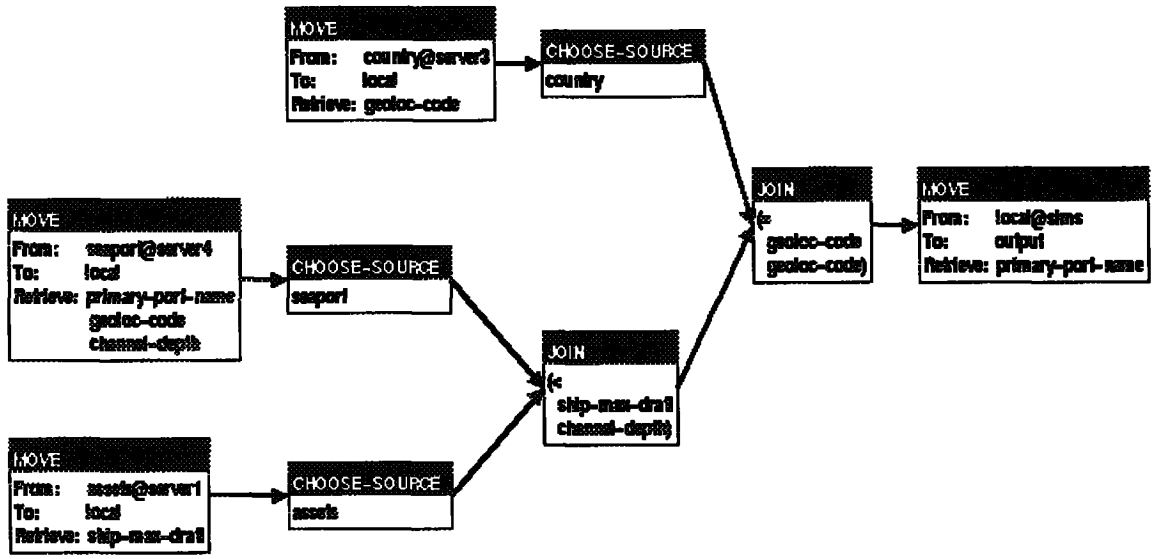

Figure 1: An information gathering plan

ond difference is that query access planning is also concerned with the low-level implementation of the query processing to minimize the processing time and amount of disk I/O. In our formulation of the information gathering task, we are only concerned with determining how the queries should be partitioned into subqueries to the remote sources and we leave it up to the individual systems to implement the subqueries efficiently.

In earlier papers (Knoblock 1994; 1995) we addressed various issues related to planning for information gathering, such as simultaneous actions, integrating planning and execution, replanning, and sensing. This paper focuses on the issues that arise in taking a general-purpose planner and using it to construct a planner for the information gathering task. In order to provide insight into some of the design decisions we include some discussion of how this planner has evolved over the last few years.

The paper is organized as follows: First, we describe the underlying general-purpose planner and identify both the important features and additional functionality that had to be added. Next, we describe how we represent the information gathering task as a planning problem and identify some of the important design principles. Given this representation, we then describe the control knowledge and evaluation function used to efficiently generate high-quality information gathering plans. In order to demonstrate that the system can efficiently generate plans, we provide some empirical results that compare the planning time to the execution time. Then we compare this work to work on query access planning as well as other related work on information gathering. Finally, we identify some of the critical planning research problems, and then describe some of our plans for future work.

The Basic Planner

The first prototype of the overall system (Arens & Knoblock 1992; Arens *et al.* 1993) was built using the Prodigy 2.0 planner (Minton *et al.* 1989). Prodigy was chosen because it has a expressive operator representation language and a rich language for expressing control knowledge. This initial version of the system produced a satisficing solution in that it searched the space using a set of heuristics that produced a reasonably good solution. Since an important aspect of this problem is plan quality, generating high quality plans requires exploiting the fact that remote sources can be accessed in parallel. To do this required explicitly representing the potential parallelism in the plan. Since Prodigy produces totally-ordered plans, the plans produced by Prodigy were converted into a parallel-execution plans using the algorithm of Veloso (Veloso 1989), which converts a totally-ordered plan into a partially ordered plan.

In the next version of the system we switched to UCPOP 2.0 (Barrett *et al.* 1993) because we needed a partial-order planning capability. The initial version in Prodigy returned the first plan produced using the set of heuristics, but as the plans became more complex it became clear that this was insufficient. In order to consider the tradeoffs in different ways of processing a query, we needed to have the planner construct alternative plans and evaluate the alternatives to find a high-quality plan. To do this using a total-order planner would be costly since the final result is a partially-ordered plan and a number of different totally-ordered plans can all map to the same partially-ordered plan. To avoid considering redundant plans and to evaluate the partially-constructed plans, a partial-order planner is better suited to this particular task. In addition to providing a partial-order planning capability, UCPOP

provides a number of important features of Prodigy, including an expressive operator language, a separate control language, and the capability of defining preconditions using *functional predicates* (described below).

The expressive operator language is necessary for defining the operators in the information gathering domain. The language features used in the current version of the domain are conjunction, disjunction, existential, and universal quantification. The control language was used initially for this domain in both Prodigy and UCPOP, but was later abandoned for efficiency reasons.

The ability to define preconditions using functional predicates is also a critical feature for representing this problem. This feature allows certain predicates to be defined as functions, where given bindings for some of the variables it computes the bindings of the other variables. This capability allows the operators to be defined at an appropriate level of abstraction. The aspect of the information gathering problem that is well suited to a planner is the search through the space of possible operations for selecting the sources and manipulating the data. There are other aspects to this problem such as computing the possible partitions of a query or determining whether a set of data is contained in a particular information source that would be difficult to capture in a planner. The ability to define preconditions using functional predicates turns out to be particularly useful since much of the low-level query processing can be done in Lisp code, which allows the planner to focus on the search at an appropriate level of granularity.

There are several capabilities that are not provided by UCPOP that are required to solve the task. These are the ability to represent and manipulate complex terms and the explicit representation of resources. The representation of complex terms is essential since the specification of a set of data can be quite involved. It includes the specification of the classes of objects, the relevant roles on those objects, constraints on the individual objects as well as constraints between objects, and so on. An example of a complex term is shown in the example query shown in Table 1. Terms such as these would be impractical to define as a flat literal with a fixed number of parameters. It would also be difficult to plan for queries if they were represented by a set of flat literals since that would require reasoning about the achievement of several simultaneous goals and existing planners are not particularly well suited to that problem.

In order to execute several actions simultaneously in a partially-ordered plan requires an explicit representation of reusable resources. A partially-ordered plan only allows actions that are unordered with respect to each other to be executed in either order. It does not sanction simultaneous execution. To provide simultaneous execution requires the addition of explicit resource constraints (Wilkins 1988; Knoblock 1994). The advantage of parallel plans are that they can greatly reduce the overall execution time of queries by supporting simultaneous execution. The representation of reusable resources allows the system to explicitly reason about potential resource conflicts and take them into account in order to generate the best plan to solve the problem. An alternative approach to address this problem would be to schedule the operations after the planning is complete; however, this approach would be less efficient since in some cases there will be several different operators for achieving the same goal and the choice of operator, which would be made before the scheduling is performed, could have a significant impact on the schedule.

In order to address the needs of this application, we built a general planner called Sage, which extends the UCPOP planner in several important ways. First, we added support for compound objects, which required extending the matching algorithm. Second, we added an explicit representation of reusable resources and extended the planner to identify possible resource conflicts and refine the plan to eliminate them. This allows the planner to generate plans with simultaneous actions. (See (Knoblock 1994) for more detail.)

In addition, we added support in Sage for simultaneous and interleaved planning and execution. This is done by tightly integrating the execution into the planning process and letting the planner decide which actions to execute and when to execute them. This allows the planner to replan failed actions, handle asynchronous goals, and interleave the planning and sensing. To support the sensing we added explicit run-time variables (Ambros-Ingerson & Steel 1988; Etzioni *et al.* 1992) which provide a mechanism for the planner to use the results of a sensing action. The support for execution, replanning, and sensing are described in (Knoblock 1995) and will not be described further in this paper. In the remainder of this paper we will focus on how the information gathering task is represented in Sage and how the planner efficiently generates plans in this domain.

Representing the Problem

The problem to be solved in this task is to determine a sequence of actions to efficiently retrieve a requested set of data (i.e., the query). In the initial representation of this problem in Prodigy, we cast as much of this problem as possible as a planning problem (Arens *et al.* 1993). The operators manipulated the individual terms of a query to determine which terms depended on other terms, which in turn determines the order in which different queries would be executed. Once a complete plan graph was constructed, the resulting plan had to then be converted into an actual query plan that could be executed. There were several important limitations of this representation. First, the operators in the resulting plan did not correspond directly to the actual actions that would be executed,

which made it difficult to evaluate intermediate plans to more efficiently search the space. Second, much of the work that was being done by the planner involved analyzing the structure of the query, which could be done more efficiently in Lisp code.

In the next version implemented in UCPOP, we redesigned the representation of the problem with two design goals in mind. First, we wanted the operators in the plan to correspond directly to the operators that would actually be executed. This simplified the evaluation of intermediate plans and made it possible to integrate the planning and execution. Second, we wanted the preconditions of these operators to be completely self-contained goals. This simplified and modularized the overall design of the operators and made it possible to extend the domain in new directions such as the integration of sensing into the system. The difficulty in this choice of representation is that it placed a much greater burden on the processing required for each individual operator. There are now more than 10,000 lines of Lisp code used to define the functional predicates used in the operators.

Using this representation, a query is cast as an information goal. Such a goal consists of a description of a set of desired data (i.e., the query) as well as the location where that data is to be sent. The form of this goal is:

```
(available <source> <server> <query>).
```

The **<source>** is either the name of a remote source, **local** to indicate the local knowledge base, or **output** to indicate that the results should be displayed. The **<server>** is either the name of a server that can provide one or more sources (e.g., an Oracle DBMS), or it is the name of the local information mediator, which is called **sims** in the example. Finally, the **<query>** is a description of desired information. An example of an information goal is shown in Table 2.

```
(available output sims
   (retrieve (?port-name)
      (:and (seaport ?sport)
            (port-name ?sport ?port-name)
            (channel-of ?sport ?channel)
            (channel-depth ?channel ?depth)
            (transport-ship ?ship)
            (vehicle-type-name ?ship "breakbulk")
            (max-draft ?ship ?draft)
            (< ?draft ?depth))))
```

Table 2: An information gathering goal

All of the operators in this domain manipulate these information goals. There are two types of operators: those that correspond to data manipulation operators and those that simply reformulate an information goal into an equivalent goal and are used to select an appropriate set of sources. The data manipulation operators include **move**, **join**, **update**, **assign**, **select**, and **compute**. The reformulation operators include: **choose-source**, **infer-equivalence**, **substitute-definition**, and **decompose**. The reformulation operators are used to rewrite the query expressed in domain terms into queries expressed in terms of the available information sources. The details of the reformulation operators are provided in (Arens, Knoblock, & Shen 1996).

Consider the operator shown in Table 3, which defines a join performed in the local system. This operator is used to achieve the goal of making some information available in the local knowledge base of the SIMS information mediator. It does this by partitioning the request into two subsets of the requested data, retrieving that information into the local system, and then joining the data together to produce the requested set of data.

```
(define (operator join)
  :parameters (?join-op ?data ?data-a ?data-b)
  :precondition
     (:and (join-partition ?data ?join-op
                           ?data-a ?data-b)
           (available local sims ?data-a)
           (available local sims ?data-b))
  :effect (available local sims ?data))
```

Table 3: The join operator

The **join-partition** precondition is defined by a functional predicate that produces the relevant partitions of the requested data. For example, the query described above would be partitioned into two subqueries based on how the information is organized in the underlying information sources. Since information on **transport-ship** and **seaport** is located in different sources, the function would return the partition shown in Table 4.

Join Constraint:

```
(< ?draft ?depth)
```

Subquery 1:

```
(retrieve (?port-name ?depth)
   (:and (seaport ?sport)
         (port-name ?sport ?port-name)
         (channel-of ?sport ?channel)
         (channel-depth ?channel ?depth)))
```

Subquery 2:

```
(retrieve (?draft)
   (:and (transport-ship ?ship)
         (vehicle-type-name ?ship "breakbulk")
         (max-draft ?ship ?draft)))
```

Table 4: Result of calling Join-Partition on the example query

The functional predicates process queries based on a model of the domain and models of the contents of the information sources. This information comprises the static part of the initial state information. To organize this information and access it efficiently, these

models are stored in a knowledge representation system called Loom (MacGregor 1990). The information is then accessed directly through the functional predicates, which make direct calls to Loom.

The dynamic part of the initial state information is comprised of literals that define the available information sources (e.g., databases) and the servers (e.g., an Oracle DBMS) they are running on. The example shown in Table 5 defines four databases running on three different servers. The first two lines define two Oracle databases running on a single Oracle server. The second two lines define two Loom knowledge bases running on different servers.

```
((source-available assets server1)
 (source-available geo server1)
 (source-available country server3)
 (source-available seaport server4))
```

Table 5: The dynamic part of the initial state

Searching for High-Quality Plans

This information gathering task is naturally cast as a planning problem since the problem is to find a sequence of actions to retrieve and integrate the requested data. However, this task differs from much of the previous planning work in that the problem is to find a high-quality plan rather than any plan that solves the goal. In addition, since the cost of planning must be added to the overall processing time, the system must efficiently produce plans. This section describes how Sage efficiently generates high-quality plans.

The size of the search space is potentially quite large in this domain. Unlike many other planning domains, there are not a large number of interactions between operations. However, there are still a number of aspects to this problem that all contribute to the size of the search space. First, there may be a number of different possible orders in which the information can be processed. For example, if there are multiple sets of data that must be combined (i.e., a join), then there could be many possible orders in which to join the data. The order in which the data is combined is important since it determines the amount of intermediate data that will be produced. Second, there may be more than one source where the requested information can be retrieved and different sources may have different access costs. This may also affect what information must be processed locally and what processing can be done by the remote system. Third, there may be interactions between actions when two actions require the same resource. For example, if you have two queries to the same server, then these will have to be done sequentially. Fourth, there may be several alternative ways to combine information. For example, instead of retrieving two sets of data and performing a join, it may be more efficient to retrieve the first set of data, and then use the results to compose a query to the second source.

A natural approach to control the search is through the use of control rules. The control rules can be used to prune portions of the search space that are redundant or unnecessary to produce the best plan. In our initial implementation of the information gathering task we used the control rule language provided by UCPOP. Unfortunately, the overhead involved in considering the control rules was too high and greatly reduced the number of search nodes that could be considered in the same amount of time. This problem could probably be addressed by adding an efficient rule matching facility to UCPOP. We took the more direct approach of simply moving the control information directly into the definition of the functional predicates used in the operators. This type of control information prunes portions of the search space that will never need to be considered. For example, the operators consider only joins across data that are distributed in different information sources. It will generally be less efficient to pull two sets of information from the same information source and perform the join locally rather than in the remote source. In this case, this heuristic is captured in the implementation of the `join-partition` function. We carefully crafted the functional predicates to include this information so that they only generate choice points that are relevant to solving the problem.

In addition to constraining the space as much as possible, we also want to minimize the portion of the space that will need to be considered, but still consider alternative plans for processing the same query. To do this we use a branch-and-bound search with an evaluation function (specific to information gathering) for estimating the cost of alternative plans. Branch-and-bound expands the plan with the lowest estimated cost at each step in the search process, which will produce the optimal plan relative to the given operator definitions and evaluation function. The evaluation function employs standard database estimation techniques to estimate the cost of processing a query with a particular plan. The evaluation function estimates the cost of each operation by maintaining information about the size of each class and the number of different possible values for each role of a class (i.e., the cardinality of a role). Assuming a uniform distribution of the data, it then estimates the amount of intermediate data that will be retrieved and manipulated, which is usually the dominant cost in handling multidatabase queries. Using the estimated cost of each operation, the function computes an estimate for the overall cost of a plan, taking into account the parallelism of some of the actions. The evaluation function allows the planner to compare different partially-constructed plans; those plans that are more expensive than what the evaluation function determines to be the optimal plan will never be expanded further.

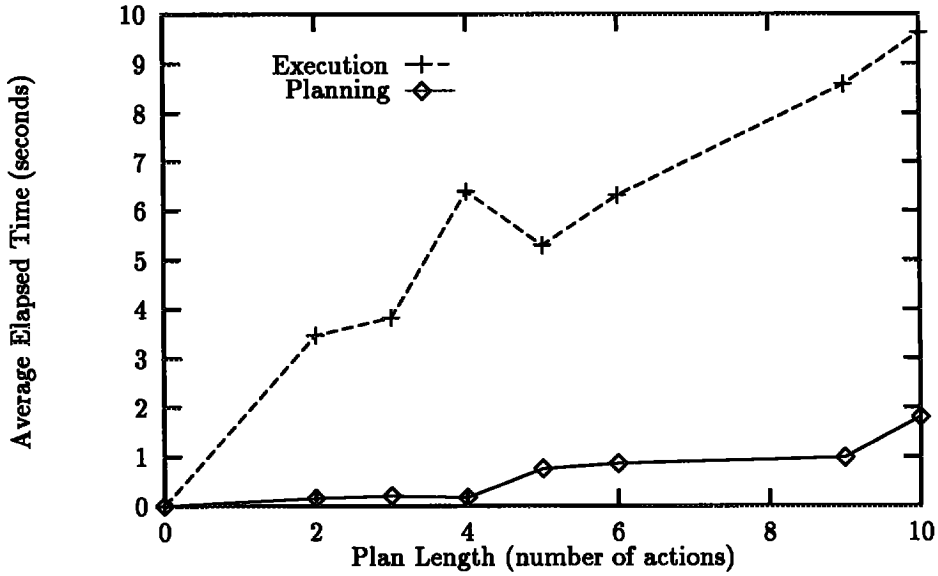

Figure 2: Comparison of the elapsed time for planning versus execution

Results

The Sage planner serves as the query processing engine for the SIMS information mediator (Arens *et al.* 1993; Knoblock, Arens, & Hsu 1994). Using SIMS we have built mediators to provide access to data for transportation planning and trauma-care medical information. We are also in the process of building mediators to integrate information for genetics counseling and military logistics. Note that for each of these applications the planner and planner domain remain the same, but the underlying information sources and the models of the information sources are changed.

In the transportation planning domain, the planner efficiently generates plans for queries that have as many as 30 terms and require access from up to three different information sources. To demonstrate that the system can efficiently generate plans for queries in this domain, Figure 2 compares the time to construct the plan for a query to the time for executing a plan (which does not include the planning time). The problems are ordered by plan size and range from 2 steps to 10 steps. The graph compares elapsed time and not CPU times since the execution of a plan occurs across multiple processes on different machines. Since there is some variation in elapsed time, each of the queries was run 10 times and the average time for each length plan is shown in the graph. The graph clearly shows that the planner can efficiently generate information gathering plans for the given test set. In addition, this time is only a small fraction of the overall execution time.

Related Work

The database community has been building specialized systems for performing query access planning for many years (Jarke & Koch 1984; Selinger *et al.* 1988). In order to address this problem these systems exploit a variety of heuristics and techniques to constrain the search space. However, the problem is inherently a search problem, so there is no way to completely eliminate the search. These approaches have been carefully optimized for the problem of query processing in the single database environment and the distributed database environment. However, there has been less work on the problem of query processing for multidatabase systems (Landers & Rosenberg 1982; Reddy, Prasad, & Reddy 1989) where there are a set of distributed, autonomous, and heterogeneous databases. The existing approaches in this environment are inflexible in the sense that choice of the information sources for a given query is fixed and the integration of this information is predefined. This essentially reduces the problem to the single-database query access planning problem.

The information gathering problem described here differs from the more traditional query access planning in two significant ways. First, instead of assuming that the choice of information source is fixed, as part of the planning process the system selects the information sources that will be used for processing a query. Second, traditional query access planning reasons about the query processing at a more detailed level which involves not just the selection of which operation to perform on the data, but also how to implement the operation efficiently (e.g., performing a join using a *sort join* or a *hash join*). Since we use other systems to actually implement the query processing actions, we leave it to these system to decide on the most efficient way to implement these operations. Instead, we have

focused on the problem of deciding where to get the information, where to process it, and what order to do the processing.

A variety of work on planning has explored various aspects of the query access planning problem. The LADDER system (Sacerdoti 1977) had a similar goal of integrating multiple sources of information. They developed a planner (Furukawa 1977) based on a theorem prover that is used to find the minimal set of sources to cover a query, and they combined it with a specialized heuristic algorithm that produces an efficient query access plan. In contrast, Sage uses a general planner for the query access planning and integrates the source selection and query access planning into a single planning process, which allows the system to generate more efficient plans.

More recent work on query access planning has focused on plan merging and plan optimization. Qiang et al. (1992) developed an approach to efficiently merging plans, including query access plans. Shekhar (1989) developed an approach to trading off search time with execution time in optimizing query access plans. Both of these systems assume that the query access plans are given.

Another planner that has been built for a information gathering task is the XII planner (Etzioni, Golden, & Weld 1994). This planner serves as the query processor for the Unix Softbot (Etzioni & Weld 1994). Compared to Sage, XII reasons about the information at a different level of granularity. Instead of representing general actions for manipulating data, each operator corresponds to a Unix command. The advantage of their approach is that it provides finer-grained control and reasoning of the information. The disadvantages are first, that the operators are application specific and a new set of operators would have to be engineered to support another application domain. Second, since the system reasons about the information as individual tuples instead of sets of information, it would be impractical to efficiently reason about and manipulate large amounts of data.

Levy et al. (1995) present a different approach to information gathering. In their work they have been developing specialized algorithms for the problems of source selection and planning (essentially building a special-purpose planner). They have focused largely on efficient algorithms for source selection.

Discussion

This paper described our experiences in applying a general-purpose planner to solve the information gathering problem. An important lesson from this work is that the issues that arise in this particular real-world problem are different than the ones many researchers have focused on in the past and continue to work on. Previous work on planning has focused extensively on both developing more expressive representations and handling interactions between operators. While these are certainly important problems, other issues such as constraining the space of plans so that it can be searched efficiently and generating high-quality plans have been largely ignored.

Consider the issue of constraining the space of plans. There are a number of speed-up learning systems that learn control knowledge to constrain the search space. But these systems do not fully address the problem since these systems often reduce a very large search to a smaller one, but in most cases they do not reduce search to the point where realistic problems in these domains can be solved. Other important work on using planning technology to solve real problems, such as in SIPE (Wilkins 1988) and O-Plan(Currie & Tate 1991), has relied on careful engineering of the domains such that problems can be solved with only a modest amount of search. When people build specialized planners for specific applications they develop techniques and heuristics that make some interesting class of problems tractable. Planning research has focused heavily on the techniques and seems to have neglected the infrastructure required for developing and exploiting the heuristics required to solve real problems.

The issue of plan quality has received even less attention. Perhaps this is due to the fact that without good heuristics there is not much hope of doing anything but a satisficing search. However, in many real-world domains, (e.g., query access planning, process planning, logistics planning, etc.) finding a high-quality solution is an integral part of the problem.

The next step in our work will focus on gathering and integrating semi-structured information in the World Wide Web. In the Web we will have to deal with the problem of combining information from many more sources. In addition, sensing actions will probably play a larger role since a query may require gathering additional information in order to locate the relevant sources to answer a query (Knoblock & Levy 1995). We expect that the combination of larger plans and more operators will require us to push even harder on techniques for efficiently generating high-quality plans.

Acknowledgments

Thanks to Jose Luis Ambite for his work on developing a good evaluation function for the planner. And thanks to Steve Minton and Andre Valente for their comments on earlier drafts of this paper.

The research reported here was supported in part by Rome Laboratory of the Air Force Systems Command and the Advanced Research Projects Agency under contract number F30602-94-C-0210, and in part by the National Science Foundation under grant number IRI-9313993. The views and conclusions contained in this report are those of the author and should not be interpreted as representing the official opinion or policy of RL, ARPA, NSF, the U.S. Government, or any person or agency connected with them.

References

Ambros-Ingerson, J., and Steel, S. 1988. Integrating planning, execution, and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 83–88.

Arens, Y., and Knoblock, C. A. 1992. Planning and reformulating queries for semantically-modeled multidabase systems. In *Proceedings of the First International Conference on Information and Knowledge Management*, 92–101.

Arens, Y.; Chee, C. Y.; Hsu, C.-N.; and Knoblock, C. A. 1993. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems* 2(2):127–158.

Arens, Y.; Knoblock, C. A.; and Shen, W.-M. 1996. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems.*

Barrett, A.; Golden, K.; Penberthy, S.; and Weld, D. 1993. UCPOP user's manual (version 2.0). Technical Report 93-09-06, Department of Computer Science and Engineering, University of Washington.

Currie, K., and Tate, A. 1991. O-plan: The open planning architecture. *Artificial Intelligence* 52(1):49–86.

Etzioni, O., and Weld, D. S. 1994. A softbot-based interface to the Internet. *Communications of the ACM* 37(7).

Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 115–125.

Etzioni, O.; Golden, K.; and Weld, D. 1994. Tractable closed-world reasoning with updates. In *Fourth International Conference on Principles of Knowledge Representation and Reasoning.*

Furukawa, K. 1977. A deductive question answering system on relational data bases. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 59–66.

Jarke, M., and Koch, J. 1984. Query optimization in database systems. *ACM Computing Surveys* 16(2):111–152.

Knoblock, C. A., and Levy, A. 1995. Exploiting runtime information for efficient processing of queries. In *Working Notes of the AAAI Spring Symposium on Information Gathering in Heterogeneous, Distributed Environments.*

Knoblock, C. A.; Arens, Y.; and Hsu, C.-N. 1994. Cooperating agents for information retrieval. In *Proceedings of the Second International Conference on Cooperative Information Systems.*

Knoblock, C. A. 1994. Generating parallel execution plans with a partial-order planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems.*

Knoblock, C. A. 1995. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence.*

Landers, T., and Rosenberg, R. L. 1982. An overview of Multibase. In Schneider, H., ed., *Distributed Data Bases.* North-Holland.

Levy, A. Y.; Srivastava, D.; and Kirk, T. 1995. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval* 5(2).

MacGregor, R. 1990. The evolving technology of classification-based knowledge representation systems. In Sowa, J., ed., *Principles of Semantic Networks: Explorations in the Representation of Knowledge.* Morgan Kaufmann.

Minton, S.; Knoblock, C. A.; Kuokka, D. R.; Gil, Y.; Joseph, R. L.; and Carbonell, J. G. 1989. PRODIGY 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, School of Computer Science, Carnegie Mellon University.

Reddy, M.; Prasad, B.; and Reddy, P. 1989. Query processing in heterogeneous distributed database management systems. In Gupta, A., ed., *Integration of Information Systems: Bridging Heterogeneous Databases.* NY: IEEE Press. 264–277.

Sacerdoti, E. D. 1977. Language access to distributed data with error recovery. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 196–202.

Selinger, P. G.; Astrahan, M.; Chamberlin, D.; Lorie, R.; and Price, T. 1988. Access path selection in a relational database management system. In *Artificial Intelligence and Databases.* Los Altos, CA: Morgan Kaufmann. 511–522.

Shekhar, S., and Dutta, S. 1989. Minimizing response times in real time planning and search. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.*

Veloso, M. M. 1989. Nonlinear problem solving using intelligent casual-commitment. Technical Report CMU-CS-89-210, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm.* San Mateo, CA: Morgan Kaufmann.

Yang, Q.; Nau, D. S.; and Hendler, J. 1992. Merging separately generated plans with restricted interactions. *Computational Intelligence* 8(4).