Heracles II: Conditional Constraint Networks for Interleaved Planning and Information Gathering

José Luis Ambite, Craig A. Knoblock, Maria Muslea

Information Sciences Institute
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292, USA
{ambite, knoblock, mariam}@isi.edu
http://www.isi.edu/info-agents/

Steven Minton

Fetch Technologies,
2041 Rosecrans Ave., El Segundo, CA 90245, USA
Steve.Minton@fetch.com
http://www.fetch.com/

Abstract

Harnessing the increasing amount of information available through public and private networks to inform decision-making presents a considerable challenge. There is a critical need for a system that integrates and structures diverse information in support of the user tasks and goals. The system must focus on the relevant information, evaluate trade-offs, and suggest courses of action to the user. Since we cannot ensure that all information and preferences can ever be captured by the system, the planning process should be conducted in a mixed-initiative fashion where the user can explore different alternatives and override the system suggestions as needed. In this article we present Heracles II, a constraint-based framework for interactive planning and information gathering. We describe the two contributions that address the limitations of previous work: (1) a hierarchically-partitioned conditional constraint network representation that models the task structure of the application domain, and (2) a constraint propagation algorithm that supports flexible user interaction. Heracles II is fully implemented and has been applied to several practical domains such as travel planning and geospatial information integration.

Introduction

For any activity there is a wealth of information available through public and private networks. Unfortunately, such information is distributed among many sites, with different data formats, schemas, and semantics. Moreover, information access per se is of limited value. What is needed is a system that integrates and structures diverse information in support of the user tasks and goals. The system must gather the relevant information, evaluate tradeoffs, and suggest courses of action to the user.

As an example consider travel planning. There are numerous sites with relevant travel information: flight schedules and fares (e.g., www.orbitz.com), hotel locations and rates (e.g., www.itn.com), car rental sites (e.g., www.hertz.com), weather information (e.g., weather.yahoo.com), maps and route planning (e.g., www.mapquest.com), airport parking rates (e.g., www.airwise.com), etc. This information needs be integrated with user preferences, such as preferred airlines or flying times (e.g., avoid red-eye flights), cost constraints, and company policies, such as allowable airlines, expense caps, per-diem or mileage reimbursement rates. Although the user can visit these sites and take into account

all the constraints and preferences manually, it is extremely tedious, error-prone, and time-consuming. A system that queries the remote sites, accesses local information, and enforces the constraints and preferences is much more desirable. Planning support and interactivity are the main requirements for this kind of systems. In order to support planning, the system must (1) gather and integrate the information in a coherent structure that captures the tasks needed for the application domain, (2) evaluate tradeoffs and select among different alternative courses action, (3) allow the user to explore and override systems suggestions. To provide flexible interaction, the system must (1) allow the user to input data or change choices at any time during the planning process, and (2) handle information sources which return results asynchronously.

In this article, we present an approach to mixed-initiative planning and information gathering based on conditional constraint networks (Mittal & Falkenhainer 1990) that addresses these challenges. The rest of the article is structured as follows. First, we discuss the limitations of closely related previous work, including our own initial approach (Knoblock *et al.* 2001). Second, we describe the core contributions of Heracles II: (1) mapping the hierarchical task structure of the planning domain into a conditional constraint network, and (2) a constraint propagation algorithm that ensures correct propagation in the presence of cycles, user interactions, and asynchronous sources. Third, we briefly discuss an HeraclesMAPS, and application of Heracles that integrates geospatial and online data. Finally, we discuss related work, future work, and conclude.

Background and Previous Work

Our own initial approach to mixed-initiative planning and information gathering was Heracles (Knoblock *et al.* 2001). Heracles models each piece of information as a variable in a constraint network. The different pieces of information are integrated using constraints. The resulting constraint network provides a coherent view of the user activities and captures the relevant information and user preferences.

For any non-trivial user activity the number of variables and constraints is very large. Thus, Heracles partitions the network hierarchically corresponding to the task structure of the application domain, in a manner similar to Hierarchical Task Network (HTN) planning (Erol, Hendler, & Nau

1994). The application designer groups variables and constraints related to a distinct task into a package that we call a *template*. As an example consider the templates of Figure 1. Figure 1(a) shows the top level template of our travel planner, which includes the most important information about the trip such as the dates of travel, origin and destination. The next layer of decisions include (1) the alternative means of transportation, such as flying, taking a train, renting a car, driving the user's own car, or taking a taxi; and (2) choices of accommodation at the destination. Figure 1(a) shows that the system suggests to Fly. The variables and constraints related to flying constitute another template, which is shown in Figure 1(b). Each template is further decomposed into more specific subtemplates. For example, once flying is chosen as the main mode of transportation, the system needs to evaluate how to get to the airport: by taxi, driving one's car and leaving it parked at the airport, and so on. Figure 2 shows the task/template hierarchy for the travel planner.

In Heracles the information gathered by the system or input by the user is propagated automatically in the constraint network. Heracles also allows the user to override the values suggested by the system. For example, Figure 3 shows the changes to the Taxi template when the user selects a different departure airport in the Round Trip Flights template. Figure 3(a) shows the information relevant to going to the airport by Taxi, including cost, time estimate, map, and route from the user location (USC) to Los Angeles airport (LAX). Figure 3(b) shows the user selecting Long Beach airport (LGB) instead of LAX. Finally, Figure 3(c), shows the Taxi template reflecting this change including new cost, time, and maps.

Heracles showed that a constraint-based approach was well suited to support mixed-initiative planning where user interaction and asynchronous information gathering were central requirements, but had two serious limitations. First, the template selection mechanism was hardcoded into the implementation and not integrated with the constraint network. Template selection in Heracles was performed by a procedure that inspected the values of distinguished variables called *expansion variables*. For example, the ModeToDestination variable in Figure 2 (labeled Outbound Mode in Figure 1(a)) is an expansion variable that can take the values Fly, Drive, RentCar, or Taxi. Whenever an expansion variable in a parent template was set, Heracles added the corresponding child template to the constraint network and removed the alternative (child) templates. This hardcoded behavior meant that most of the logic for selecting among templates needed to appear at the parent template, even if such information logically belonged to the child templates. This diminished the modularity of templates and tended to create large, monolithic templates. The problem becomes ever more acute the deeper the template/task hierarchy. A second limitation was that Heracles could not handle cycles in the constraint network. This required the template designer to specify the constraints sometimes in an unintuitive way or forego some lines of reasoning altogether.

Heracles II provides solutions to these limitations while preserving the advantages of Heracles. First, Heracles II is uniformly represented as a *conditional constraint network*
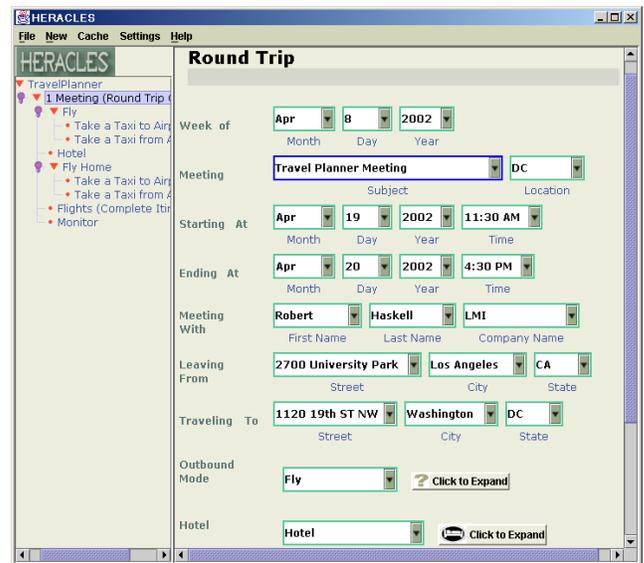
(a) Top-level template
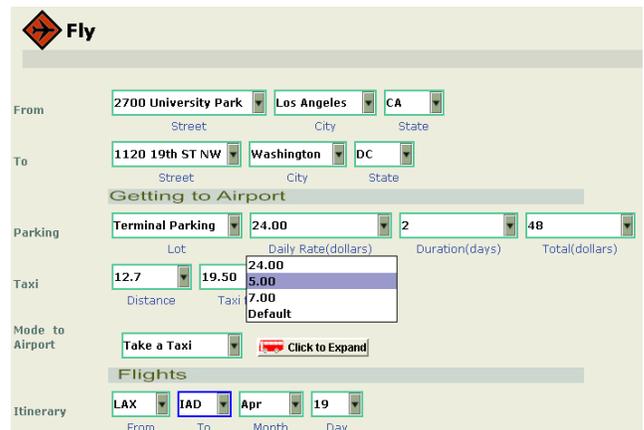
(b) Fly template
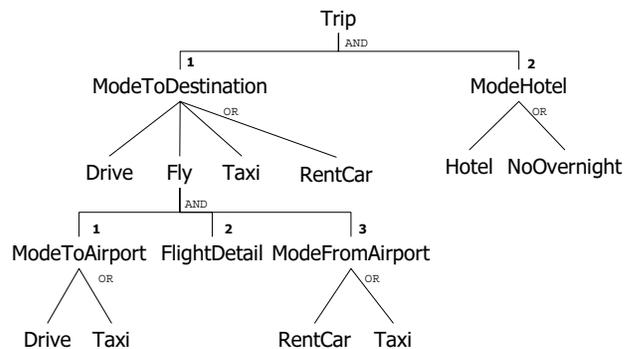
Figure 1: Travel planner templates

Figure 2: Hierarchical Organization of Templates

(a) Taxi template

(b) User changes value in Round Trip Flights

(c) Changes propagate to Taxi template

Figure 3: Travel planner: user interaction and constraint propagation

and template selection follows naturally from the evaluation of *activity constraints*. To ensure that all information relevant to the choice of templates/tasks is always considered we introduce the concept of a *core network* which is always active. Second, we designed a new constraint propagation algorithm that can handle cyclical networks in the presence of user interaction and asynchronous sources. In the following sections we describe these contributions in detail, but before that, it is instructive to consider other approaches to the problem of combining planning, information gathering, and user interaction.

The SmartClients system (Torrens, Faltings, & Pu 2002) uses constraint satisfaction to support the user in planning and information gathering. They also focus on travel planning as an application domain. In response to the user input of origin, destination, and dates for a trip, SmartClients automatically compiles a constraint network to explore the possible trips. Torrens et al. assume that SmartClients can access a remote database with flight information such as Sabre, so that it can retrieve all the flights between the selected cities in the given dates and populate the appropriate variables and constraints in the network. Once the network has been initialized, SmartClients searches for a solution trip that satisfies all the constraints using classical constraint satisfaction algorithms.

There are several crucial differences between Heracles and SmartClients. First, SmartClients retrieves all information beforehand assuming that the size of the relevant sections of the flights database can be compactly encoded and efficiently transmitted to the user client. However, this approach cannot scale to larger problems that incorporate a broader range of information sources. Retrieving all the relevant information, such as flights schedules, hotel location and rates, rental cars, maps, etc, before the search starts is not feasible. In contrast Heracles only accesses the external sources as the planning process proceeds, and for those values that are already part of a consistent partial solution. The more focused search of Heracles is more scalable and allows for an arbitrary exploration of the information space.

Second, the user can interact with Heracles at any point during the planning process and can change the values of variables throughout the network resulting in arbitrary retrievals of external information. In the SmartClients approach the domains of the variables in the constraint network are fixed initially, so the user can only explore solutions contained within such space (or needs to restart the whole constraint network construction and search process).

Finally, SmartClients performs full constraint satisfaction, so it finds optimal solutions according to the user preferences. However, this in done in the smaller search space defined at initialization time. Heracles performs constraint propagation and does not attempt to find an optimal solution. However, in Heracles the user can explore the full solution space with the latest information obtained in real-time from external sources and understand the different tradeoffs. In our experience letting the user guide the process toward a desirable plan interactively yields better results.

A third approach to collaborative planning and information gathering is the Trip-planner agent framework (Homb *et*

al. 1999). Trip-planner is also based on a constraint network that integrates different sources related to travel planning. In the Trip-planner the user specifies his/her preferences at the start of the planning process. During the constraint network evaluation the system calls different sources guided by the user preferences. The user is consulted again at predefined points during planning. For example, after the ten cheapest flight have been found the user is prompted to select one. However, the user cannot interact with the constraint reasoner at any point during constraint evaluation as Heracles allows.

In summary, previous approaches are limited in one or more of our requirements for a uniform framework that combines user interaction, information gathering, and planning/constraint reasoning capabilities. Table 1 compares the approaches. Heracles satisfies all three requirements, but it was limited as we described above, thus our development of Heracles II. SmartClients excels in reasoning with the data using full constraint satisfaction, but it gathers all information at initialization time. Trip-planner does interleave constraint reasoning with information gathering, but the choices for the user to impact planning are quite limited. Finally, note that travel web sites like Expedia, Travelocity, or Orbitz provide good support for information gathering and user interaction. However, they lack support for an integrated view of the (travel) planning process that satisfies complex constraints and preferences of the user.

	Planning/ CSP	User Interaction	Information Gathering
Heracles I, II	✓	✓	✓
SmartClients	✓	✓	
Trip-planner	✓		✓
Expedia, etc.		✓	✓

Table 1: Comparison of approaches

Conditional Constraint Networks and Hierarchical Planning

This section describes the first contribution of this article, namely, the formalization of hierarchical planning in Heracles II as a conditional constraint network. First, we describe the hierarchical partitioning of the Heracles II constraint network. Second, we show how we map this hierarchical decomposition into a conditional constraint network. Finally, we describe how template selection follows naturally from the evaluation of the activity constraints in the network.

As the complexity of a planning domain grows designing and maintaining a monolithic network becomes infeasible. Similarly, presenting a large network for the user to interact with quickly becomes unmanageable and confusing. Thus, in Heracles II we group the variables and constraints related to the performance of a task into a unit called a template.

Template Specification. A *template* is comprised of a name, arguments, variables, constraints, and expansions. The name uniquely identifies the template. The arguments specify the input variables, those that receive values from other templates, and the output variables, those whose values are used in other templates. All other variables are internal to the template. Each expansion specifies how a template is elaborated into the appropriate subtemplates based on the value of a so-called *expansion variable*. Figure 4 shows a fragment of the specification of the Fly template focusing on the selectModeToAirport decision. The input variables include the OriginAddress, DepartureDate among others. The output variables are SelectedFlight and FlyCost. The selectModeToAirport constraint selects the cheapest mode of transportation to the airport.

Heracles II receives as input a set of declarative XML template definitions. Figure 4 shows some of the main constructs. The values of variables can also be XML objects which are processed using XQuery. A template also includes a declarative specification of the user interface. The constraint network can control which widgets are shown in the interface depending on run-time values. A full description of the XML syntax and the constraint-based control of the user interface is outside of the scope of this article.

```
<template name="Fly">
 <args><in>OriginAddress</in> <in>DepartureDate</in> ...
       <out>SelectedFlight<out/> <out>FlyCost</out></args>
 <vars>
  <var name="OriginAddress"/> <var name="FlyCost"/>
  <var name="TaxiFare"/> <var name="ParkingCost"/>
  <var name="ModeToAirport"/> ... </vars>
 <constraints>
  <constraint name="selectModeToAirport" type="XQueryConstraint">
   <args><in>TaxiFare</in> <in>ParkingCost</in>
         <out>ModeToAirport</out> </args>
   <output><xquery><![cdata[
    <row><TaxiFare>{$TaxiFare}</TaxiFare>
         <ParkingCost>{$ParkingCost}</ParkingCost>
         <ModeToAirport>{ if ($TaxiFare >= $ParkingCost)
                          then "Drive"
                          else "Taxi"}</ModeToAirport>
    </row>]]></xquery></output></constraint> ... </constraints>
 <expansions>
  <expansion><var name="ModeToAirport"/>
   <template-call name="Drive" printname="Drive and Park">
    <in>DepartureAirport</in> <in>Duration</in>
    <out>ParkingCost</out> </template-call>
   <template-call name="Taxi" printname="Take a Taxi">
    <in>OriginAddress</in> <in>DepartureAirport</in>
    <out>Taxifare</out> </template-call>
 </expansion> ... </expansions> </template>
```

Figure 4: Fragment of the Fly template specification

Template Hierarchy. The templates are organized hierarchically to model the task structure of the planning domain, to help the user understand the planning process, and facilitate the presentation of the information. Figure 2 shows the (simplified) hierarchy of our travel planner. There are two subtasks that must be achieved for a trip: getting to the destination and finding an accommodation. These decisions are associated with two *expansion variables*: ModeToDestination and ModeHotel. Since *all* these subtasks must be achieved for a successful trip, we label the subtask decomposition with an AND. Each of these subtasks can be achieved by several alternative means. The figure shows the choices as OR branches. For example, the ModeToDestination is a expansion variable that can take the values Fly, Drive, RentCar, or Taxi. The Fly template is further decomposed in three subtemplates that handle the ground transportation at the origin and destination airports and the flight details.
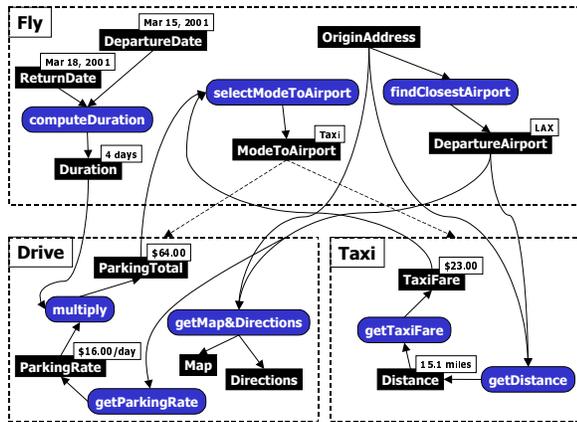
Conditional Constraint Network. Heracles II reads the set of declarative template specifications for a given appli-

cation and automatically constructs a conditional constraint network based on them.

Figure 5 shows a fragment of the constraint network for travel planning that addresses the selection of the method of travel from the user's initial location to the airport. The choices under consideration are: driving one's car (which implies leaving it parked at the airport for the duration of the trip) or taking a taxi. Figure 5 (a) shows the constraint network that the original Heracles would evaluate to make this decision. This network would need to be in the same template although some of the variables naturally should appear in subtemplates. Figure 5 (b) shows the same network in Heracles II. The variables and constraints are partitioned across several templates in a more modular way that corresponds more closely with the task structure of the domain.
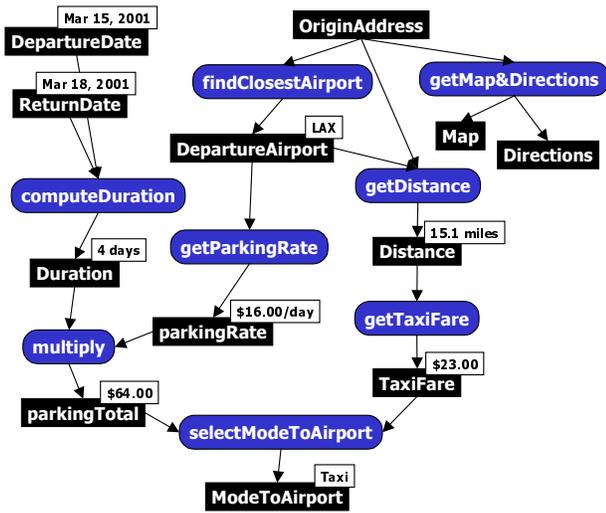
(a) Flat Constraint Network

(b) (Hierarchical) Conditional Constraint Network

Figure 5: Comparing Driving Versus Taking a Taxi

Variables. Each distinct piece of information in an application is represented as a variable in the constraint network. These values are set by the system by constraint propagation or directly by the user from the graphical interface. In a conditional constraint networks each variable not only has a value like in the classical case, but it also has an activity status. If a variable is inactive it does not participate in the network. In Figure 5 the variables are shown as dark rectangles and the values as white rectangles next to them. For example, the DepartureAirport has the value LAX (Los Angeles International), which is assigned by the system since it is the closest airport to the user's address.

Classical Constraints. A constraint is a subset of the Cartesian product of the domains of the participating variables. A constraint is a computable component which may be implemented by a local table look-up, by the computation of a local function, by retrieving a set of tuples from a remote source, or by calling an arbitrary external program.

In Figure 5 the constraints are shown as rounded rectangles. For example, the computeDuration constraint involves three variables (DepartureDate, ReturnDate, and Duration), and it's implemented by a function that computes the duration of a trip given the departure and return dates. The constraint getParkingRate is implemented by calling a wrapper that accesses a web site that contains parking rates for airports in the USA (www.airwise.com).

In Heracles (I&II) most of the constraints only have an implementation in one direction. For example, we can find out the parking cost at a given airport, but we cannot find out which are the airports that have parking lots of less than $7/day. This is one reason why Heracles performs only constraint propagation instead of full constraint satisfaction.

Activity Constraints. An activity constraint controls the activity status of a variable given the values of other variables. For example, consider an activity constraint $ac(v_1, ..., v_{n-1}, v_n)$ that is computed by the rule:

$$(v_1 = k_1) \wedge ... \wedge (v_{n-1} = k_{n-1}) \rightarrow active(v_n)$$

That is, the activity constraint will make variable v_n active whenever the variables $[v_1, ..., v_{n-1}]$ take the values $[k_1, ..., k_{n-1}]$, respectively.

Template Selection using Activity Constraints. Heracles II uses activity constraints and expansion variables to select among alternative templates.

Heracles II automatically generates activity constraints based on the expansion section of each template specification. For each expansion variable, Heracles II adds an activity constraint that act as a multiplexor, making the selected template active and making the alternative templates inactive. There is no special mechanism to achieve this effect, other than the normal evaluation of the activity constraints in the conditional network. For example, in Figure 5(b), the expansion variable ModeToAirport in the Fly template selects between the Drive and Taxi subtemplates. The activity constraint, represented in the figure by the dashed lines from the ModeToAirport variable to the Drive and Taxi subtemplates is:

Fly.ModeToAirport = "Taxi" →
active(Taxi.TaxiFare) ∧ active(Taxi.Distance) ∧
¬active(Drive.Map) ∧ ¬active(Drive.Directions)

Core Network. To ensure that template selection is responsive to changing user inputs or new information from asynchronous sources, Heracles II maintains the subset of

the network that affects the computation of the expansion variables, called the *core network*, always active. Otherwise, when a template becomes inactive its variables cannot affect the rest of the network and the choices would lack relevant information. For example, in Figure 5(b) all variables and constraints, except Map, Directions, and getMapDirections, belong to the core network since they contribute to the computation of the ModeToAirport expansion variable. This is the reason why the activity constraint above does not make inactive the core variables ParkingRate and ParkingTotal of the Drive template. And also why we did not include the case for Fly.ModeToAirport = "Drive" in the activity constraint, since all the variables of the Taxi template are core variables. We call those variables not in the core network *information variables* since their values do not affect task choices but provide additional information to the user.

Heracles II determines the core network automatically by reachability analysis. It searches the constraint network starting from the expansion variables and traversing constraints from output to input variables until it reaches the "source" variables (i.e., those variables that are not the output of any constraint and must be set by the user). The variables and constraints visited in this search constitute the core network.

Analysis. To understand the savings afforded by the conditional constraint network and the template selection mechanism, consider a task hierarchy of depth d where each task has a single decision point (expansion variable) with two possible alternatives (subtemplates). This hierarchy induces a binary OR-tree with $2^d - 1$ nodes. Further assume that each template has c core constraints and i information constraints. After evaluating the core network and deciding on the top level choice, all the information variables and constraints in one of the top subtrees remain inactive. Moreover, this behavior repeats at each level of the selected subtree, so the only information constraints (and variables) that become active are those of the selected course of action, a total of $(d + 1)i$, saving the evaluation of an exponential number, $(2^d - d)i$, of information constraints. In practice, the core network is often a small subset of the constraint network. Thus, much of the constraint evaluation effort is saved by the Heracles II template selection mechanism.

Interactive Constraint Propagation

This section describes the second contribution of this article, namely, a constraint propagation algorithm that handles user interaction and asynchronous sources in cyclic networks.

The basic constraint propagation algorithm proceeds as follows. When a given variable is assigned a value, either directly by the user or by the system, the algorithm fires all constraints that have that variable as an input. This may cause the output variables to change their value (or activity status) and the process continues recursively until there are no more changes in the network. For example, consider the network in Figure 5. First, the constraint that finds the closest airport to the user's home address assigns the value LAX to the variable DepartureAirport. Then, the constraint getParkingRate, which is a call to a web wrapper, produces

the rate \$16.00/day. This value is multiplied by the duration of the trip to compute the ParkingTotal of \$64 (using the simple local constraint multiply). A similar chain of events results in the computation of the TaxiFare, based on the distance between the origin address and the airport. Once both the ParkingTotal and the TaxiFare are computed, the selectModeToAirport constraint compares the costs and chooses the cheapest means of transportation, which in this case is to take a Taxi.

Constraint propagation in Heracles II can be seen as following a cyclic directed graph. Since the user can change the value of a variable in the network at any time and remote sources return data asynchronously, Heracles II must take special care to prevent infinite loops, to ensure that all the appropriate constraints are fired and the values propagated, and to disregard obsolete values. Heracles II constraint propagation algorithm includes a time-stamping mechanism that addresses these requirements. We describe it in the remainder of the section.

Constraint propagation is initiated when the user inputs a value. Each variable is annotated with a *user-time*, an integer incremented every time that the user inputs a new value or changes a value in the network. In addition, each variable is annotated with the set of variables that has contributed to its value (i.e., those variables that were visited in the chain of constraints that set the variable in the current user-time). This *visited* set is necessary to prevent cycles. Both these annotations are propagated as the constraints are evaluated along with the actual values assigned to the variables.

The algorithm for time-stamped constraint propagation is described by the following rules:

R1 Whenever the user changes some value in the GUI, the user-time of the corresponding variable is incremented and the visited set is set to empty.

R2 A constraint fires whenever any of its inputs changes.

R3 When a constraint fires, each output variable: (1) inherits the latest user-time of the input variables, and (2) gets an updated visited set consisting of the input variables unioned with the input variables' visited sets.

R4 A constraint blocks (does not fire) if there exists a variable V_o in the constraint's outputs and a variable V_i in the constraint's inputs such that user-time(V_o) ≥ user-time(V_i) and $V_o \in$ visited(V_i).

Figure 6 shows a sample simulation of the constraint propagation algorithm operating in a cyclic network. Constraints are shown as rectangles and variables as simple nodes. The direction of propagation of the constraints is shown by arcs. Each variable annotation has the syntax "simulation-step) user-time / visited variables [value". For example, the annotation "1) 1 / ∅ [LA" of variable v1 means that at simulation step 1 the user-time was 1, there were no other variables used in the computation of its value (i.e., the user set the value), and its value was "LA".

Figure 6(a) shows a simulation of constraint propagation in a cyclic network that picks the geocoordinates of interest (v3) in one of our applications. The network has 3 constraints: C1 that given the name of a city (v1) produces the

geocoordinates of the city center (v2), C2 that copies the city center coordinates into v3, and C3 that finds the closest city (v1) to the given latitude and longitude (v3). The rationale for this network is to give the user flexibility in selecting a geopoint of interest by either entering a city (v1) and getting the city center as the coordinates of interest (in v3), or entering a lat/long (v3) and determining the closest city (in v1). The intention of the constraint C2 is to copy its input to its output only when it is consistent to do so. The time-stamping algorithm helps to enforce those semantics even though the implementation of C2 can be a simple copy. The simulation in the figure starts with the user entering inputing Los Angeles (LA) in v1, the corresponding city center coordinates (34N118N) are propagated to v3 (in step 3). Since v3 has got a new value, constraint C3 is scheduled to fire. However, such firing is blocked by rule R4 since v1 had already been visited for the computation of v3 during the same user-time phase. The possible infinite cycle of propagation is prevented (step 4). With such state of the network, the user now inputs the lat/long 40N70W in v3 (step 5). C3 fires normally since the user input resets the visited variables of v3. Propagation continues setting v1 to New York (NY) and v2 to its city center (40N73W), but rule R4 prevents the firing of C2, again blocking a possible infinite cycle.

Both the user-time and visited set annotations are required for the constraint propagation algorithm. Consider the acyclic network in Figure 6(b) where constraint propagation reaches a given variable through different paths. When the user inputs a value in v1, C1 fires and the values of both v2 and v3 are changed. This causes both C2 *and* C3 to fire. So by simulation step 3, v5 obtains a new value. However, by simulation step 5, the updated value of v4 makes C3 fire again, and correctly changes v5 a second time even though v5 has the same user-time (1) as v3 and v4 (the inputs of C3). Since $v5 \notin visited(v4)$ and $v5 \notin visited(v3)$ the propagation through C3 is not blocked. Only propagating the user-time or only keeping track of visited variables would not produce the correct propagation. Further user inputs at simulation steps 6 and 9 continue to produce correct propagation.

HeraclesMaps: Geospatial Data Integration

In order to build a new application in Heracles the designer just needs to specify a set of templates that represent the hierarchical task structure of the application along with constraints that access the relevant sources. This specification is a set of declarative files in XML. To facilitate development, our framework already includes predefined constraint types for the most common types of sources, including wrappers, databases, XML files, and data manipulation, including XQuery processing and arbitrary Java functions.

We used the Heracles framework to rapidly build and maintain HeraclesMAPS, a system that integrates multiple geospatial and online sources. HeraclesMAPS is an application that allows a user to gather information about a specific area, including satellite images, maps, topography, hydrography, transportation networks, points of interest, airport and seaport data, and weather information. HeraclesMAPS has been deployed to Special Operations Forces.
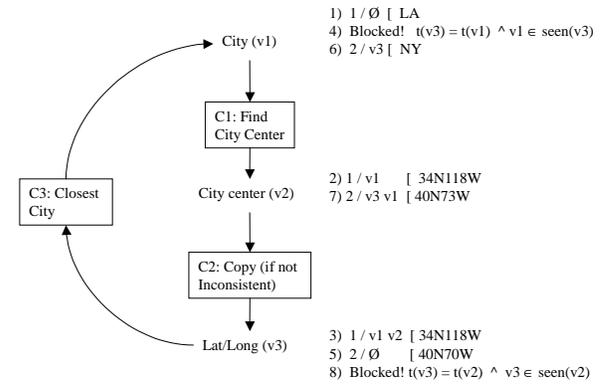
(a) Cyclic Network
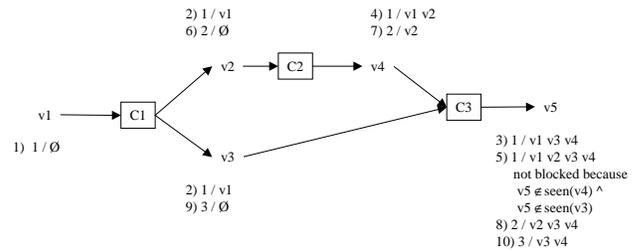
(b) Acyclic Network (race condition)

Figure 6: Interactive Constraint Propagation

In the top level template of HeraclesMAPS the user selects a location in the world by drilling down to a continent, country, and city (populated place) or by entering a latitude and longitude directly. Constraint computation is triggered by this initial location. All the different pieces of information are kept consistent by using constraints. For example, if the user recenters a map in a subtemplate, all the related maps, images, vector, and point data will be recomputed to match the new area of interest. Heracles issues the corresponding information gathering constraints that retrieve such data as the user navigates the information space.

Figure 7 shows two sample templates of HeraclesMAPS. Figure 7(a) shows a satellite image and the corresponding map of the port of Umm Qasr in Iraq. Vector data of transportation features, including roads, railroads, etc, appears overlaid on the image. Figure 7(b) shows an elevation map (top, left), the area visible (yellow) and not visible (red) from a given point (top, right), and the altitude profile (bottom, right) between two given points (from the origin specified in the top, right image, and the target in the bottom, left image). The altitude profile explains why the target point is not visible from the origin.

Related Work

We already discussed our own previous work (Knoblock *et al.* 2001), the SmartClients system (Torrens, Faltings, & Pu 2002), and the Trip planner (Homb *et al.* 1999). Here we discuss other related work.
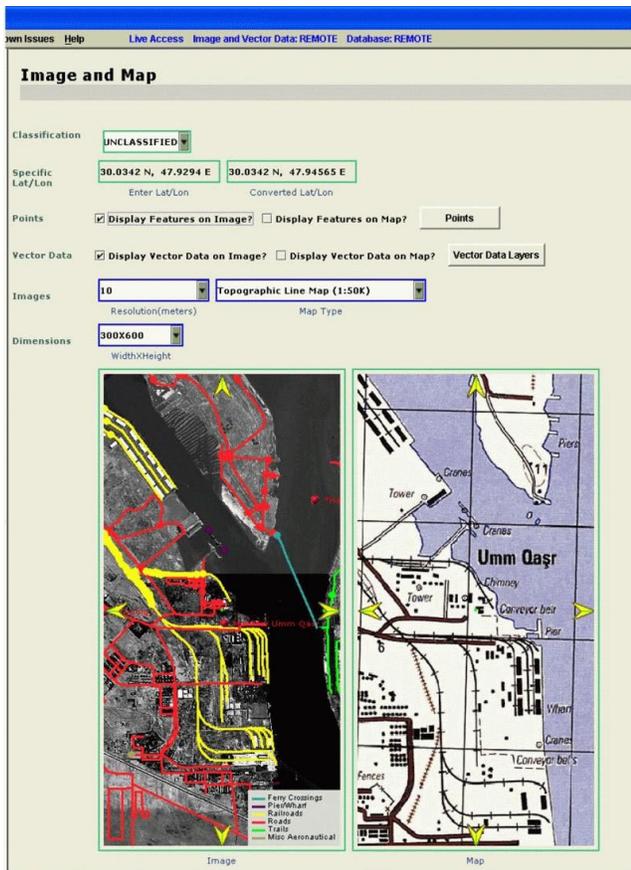
(a) Satellite image, vector data, and map
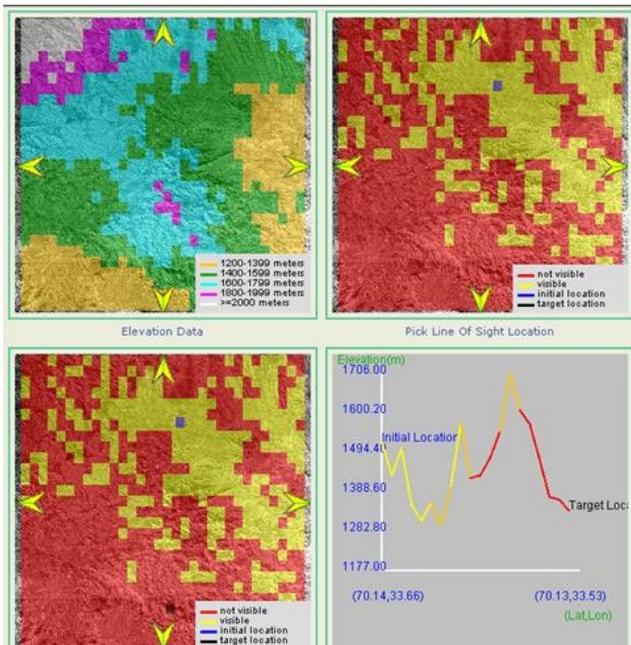
(b) Line of sight computation using elevation data

Figure 7: HeraclesMaps

The work on mixed-initiative (Burstein & McDermott 1996; Myers *et al.* 2003; Veloso, Mulvehill, & Cox 1997) and collaborative planning (Allen & Ferguson 2002) is related in spirit to the work on Heracles. A central goal in these systems is to support the user in developing a plan through some type of interactive process. The underlying planning technology is different in each of these systems, but they all share the property that the user is interactively modifying the planning process. In the work of Allen and Ferguson (2002) the user can perform actions such as refining a goal or rejecting an option, in Myers et al. (2003), the user can drop or modify constraints and tasks, and in Veloso et al. (1997), the system uses a case-based planner to propose modifications to an initial plan. Heracles, in comparison to these systems, uses a predefined set of templates that define the space of possible plans and uses constraint propagation techniques instead of a general-purpose planner to do the reasoning about the plans. The advantage of this is that it supports the combination of the planning, user interaction, and information gathering in contrast to these systems, which focus only on the integration of the user interaction and the planning.

Lamma et al. (1999) propose a framework for interactive constraint satisfaction problems (ICSP) in which the acquisition of values for each variable is interleaved with the enforcement of constraints. The interactive behavior of our constraint reasoner can be seen as a form of ICSP. However, our approach includes a notion of hierarchical decomposition and task orientation.

Discussion

We have described the core algorithms in Heracles II, an approach to planning and information-gathering based on conditional constraint networks. Planning is performed in a mixed-initiative mode, where the user can explore the space of alternative plans and override the system's suggestions. The article presented two contributions. The first one is a hierarchically-partitioned conditional constraint network representation suitable to support planning tasks. Exploring the different courses of action can be naturally handled using the activity constraints of the conditional network. The second contribution is an extended constraint propagation algorithm that handles interactive cyclical networks. Heracles II is fully implemented. We have deployed applications for travel planning and geospatial information integration.

Our future plans include seeking a formalization of the Heracles II planning and information-gathering approach combining ideas from conditional (Mittal & Falkenhainer 1990) and interactive (Lamma *et al.* 1999) constraint satisfaction. Also, we plan to explore the tradeoffs between performing constraint propagation and satisfaction in this type of systems. Although our experience is that full constraint satisfaction tends to confuse the users as they interact with the system, perhaps local constraint satisfaction may be valuable.

In conclusion, despite the breadth of research in planning and constraint programming, the interplay between planning, information gathering, and user interaction that many

important applications require presents a significant challenge. The Heracles II framework and the techniques presented in this article bring us closer to meet such challenge.

Acknowledgments

References

Allen, J., and Ferguson, G. 2002. Human-machine collaborative planning. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduled for Space*.

Burstein, M. H., and McDermott, D. V. 1996. Issues in the development of human-computer mixed-initiative planning systems. In Gorayska, B., and Mey, J., eds., *Cognitive Technology: In Search of a Humane Interface*. Elsever Science, B.V.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, 1123–1128. Seattle, Washington, USA: AAAI Press/MIT Press.

Homb, A.; Mundhe, M.; Kimsen, S.; and Sen, S. 1999. Trip-planner: An agent framework for collaborative trip planning. In *Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence*.

Knoblock, C. A.; Minton, S.; Ambite, J. L.; Muslea, M.; Oh, J.; and Frank, M. 2001. Mixed-initiative, multi-source information assistants. In *Proceedings of the Tenth International World Wide Web Conference*.

Lamma, E.; Mello, P.; Milano, M.; Cucchiara, R.; Gavanelli, M.; and Piccardi, M. 1999. Constraint propagation and value acquisition: Why we should do it interactively. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 468–477.

Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 25–32.

Myers, K. L.; Jarvis, P. A.; Tyson, W. M.; and Wolverton, M. J. 2003. A mixed-initiative framework for robust plan sketching. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*, 254–265.

Torrens, M.; Faltings, B.; and Pu, P. 2002. Smart clients: Constraint satisfaction as a paradigm for scaleable intelligent information systems. *Special issue on Constraints and Agents. CONSTRAINTS: an Internation Journal. Kluwer Academic Publishers* (7):49–69.

Veloso, M. M.; Mulvehill, A. M.; and Cox, M. T. 1997. Rationale-supported mixed-initiative case-based planning. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*, 1072–1077. Menlo Park, CA: AAAI Press.

José Luis Ambite is a senior research scientist at USC's Information Sciences Institute. His research interests include information integration, automated planning, databases, and knowledge representation, and his current focus is on automatic Web service composition. He received his PhD in computer science from USC. He is a member of the AAAI and ACM SIGMOD. He can be reached at USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292; ambite@isi.edu, www.isi.edu/~ambite.

Craig A. Knoblock is a senior project leader at USC's Information Sciences Institute and a research associate professor in computer science. He is also the chief scientist for Fetch Technologies, a company commercializing some work developed at USC. His research interests include information agents, information integration, automated planning, machine learning, and constraint reasoning. He received his PhD in computer science from Carnegie Mellon University. He can be reached at USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292; knoblock@isi.edu, www.isi.edu/~knoblock.

Maria Muslea is a research programmer at USC's Information Sciences. Her research interests include information integration and constraint programming. She received a M.S. in Computer Science from West Virginia University and a B.S. in Computer Sciences from Technical University of Cluj-Napoca, Romania. She can be reached at USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292; mariam@isi.edu, www.isi.edu/~mariam.

Steve Minton is the chief executive officer of Fetch Technologies. His research interests include machine learning, planning, and constraint satisfaction. He received his PhD in computer science from Carnegie Mellon University. Minton founded the *Journal of Artificial Intelligence Research* (JAIR) and served as its first executive editor. He has also served as an editor of the journal *Machine Learning*, and in 1998 he was elected to be a fellow of the American Association for Artificial Intelligence. He can be reached at Fetch Technologies, 2041 Rosecrans Ave., Suite 245, El Segundo, CA 90245; minton@fetch.com, www.fetch.com.