A Constraint-Based Model for Cooperative Response Generation in Information Dialogues

Yan Qu
CLARITECH Corporation
5301 Fifth Avenue
Pittsburgh, PA 15232
yqu@claritech.com

Steve Beale
Computing Research Laboratory
New Mexico State University
Box 30001, Las Cruces, NM 88003-0001
sb@crl.nmsu.edu

Abstract

This paper presents a constraint-based model for cooperative response generation for information systems dialogues, with an emphasis on detecting and resolving situations in which the user's information needs have been over-constrained. Our model integrates and extends the AI techniques of constraint satisfaction, solution synthesis and constraint hierarchy to provide an incremental computational mechanism for constructing and maintaining partial parallel solutions. Such a mechanism supports immediate detection of over-constrained situations. In addition, we explore using the knowledge in the solution synthesis network to support different relaxation strategies.

Introduction

In typical human-computer information-seeking dialogues, the user and the system do not have perfect and detailed models of each other. Over-constrained situations occur when the preferences and restrictions in the user's information needs cannot all be satisfied. Generation of cooperative response in over-constrained situations has been addressed in many human-computer dialogue systems, e.g., (Kaplan 1979; Abella, Brown, & Buntschuh 1996; Pieraccini, Levin, & Eckert 1997; Litman, Pan, & Walker 1998). In identifying relaxation candidates, however, existing systems often employ heuristics (e.g., relaxation based on constraint weights (Abella, Brown, & Buntschuh 1996; Pieraccini, Levin, & Eckert 1997)) which do not take into account the interaction effects of constraints and lack a principled and efficient way to explore the interaction effects in arriving at relaxation candidates. In our work, we implement a constraint satisfaction-based model by integrating various AI techniques such as constraint satisfaction, solution synthesis (Tsang & Foster 1990; Beale 1997) and constraint hierarchy. Solution synthesis, when integrated with constraint satisfaction and constraint hierarchy, provides a mechanism where the effects of constraint interaction are maintained as partial solutions. These effects can be exploited as a system's knowledge sources for diagnosing and resolving over-constrained situations.

Our constraint-based model supports the framework of incremental problem formulation and solution construction and refinement, consisting of *cycles of constraint acquisition, solution construction, solution evaluation, and solution modification.* The stage of constraint acquisition relies on interaction with the user. The stages of solution construction, solution evaluation and solution modification are conducted by the constraint-based problem solver. Through solution evaluation, over-constrained situations can be evaluated immediately. Through solution modification, the system can use its knowledge about the solution states to guide the adoption of cooperative strategies for resolving over-constrained situations. Repeating the cycles allows the system to help users with their problem formulation until a satisfying solution is found.

The next sections detail our constraint-based model for cooperative response generation. First, we present some terminology related to constraint satisfaction problems. Then we present in detail how the user's information needs and solutions are incrementally formulated and refined. We then illustrate how the use of knowledge sources in the constraint-based problem solver supports generation of cooperative responses in over-constrained situations. Finally, we discuss related work and summarize the paper.

Definitions

A *constraint satisfaction problem (CSP)* is typically defined as the problem of finding consistent assignment of values to a fixed set of variables given some constraints over these variables. In modeling human-computer interaction in information systems, a user's information request can be readily modeled as a CSP, with the set of attributes that constitutes a user's information need as the variables in a CSP, and the user's preferences and restrictions over these attributes as constraints. For instance, in the travel domain, attributes such as `arrival-city`, `departure-city`,

`date`, `carrier`, `time` are treated as variables. The domains for these variables are legal values found in the database (e.g., `carrier` can be {UA,AA,USAIR,...}). The variables are constrained by domain relations in the database and by user preferences and restrictions on these variables.

In information domains, the user's preferences and restrictions may be of different strengths. For example, in the travel domain, the departure city and the arrival city are usually *required* to be satisfied, while the airline carrier is *preferred* but not required. We introduce *labeled constraints*, constraints which are labeled with their respective strengths. There can be an arbitrary number of strengths reflecting varying degrees of preferences. Constraints and their strengths constitute a *constraint hierarchy*. In information domains, the task of the system is to provide users with information satisfying their information needs as much as possible. In over-constrained situations, constraints with weaker strengths should be relaxed before constraints with higher strengths. The constraint strengths used for describing examples in our system include `required`, `strong`, `weak`, and `weakest`.

We use the labeled constraint formalism to represent various types of relationships found in the information domain. *Database constraints* reflect the functional dependencies between attributes in a database (assuming a relational one in this work). Such dependencies are usually represented as tuples. Database constraints have the default strength `required`. *Domain constraints* record attributes and their importance in solving stereotypical domain problems. *User constraints* represent the restrictions and preferences in the user's information needs. User constraints are usually domain reduction constraints for variables. *User profiles* record general constraints for a certain types of users or idiosyncratic constraints for individual users.

In CSPs, the variables, domains and constraints are fixed and known beforehand. In many problems, the set of variables and the set of constraints can change in the problem solving process. Such problems are modeled as *dynamic constraint satisfaction problems* (DSCPs). A dynamic CSP can be considered as a sequence of static CSPs each resulting from a change in the preceding one, representing new facts about the environment being modeled. As a result of such an incremental change, the set of solutions of the CSP may potentially decrease (i.e., a **restriction**) or increase (i.e., a **relaxation**). In information-seeking applications, the set of variables that are relevant to a user's information need and the values that can be assigned to them change dynamically in response to user input and the negotiation between the user and the system during the course of interaction. Therefore, information-seeking human-computer dialogues is a dynamic CSP.

Dynamic CSP-based Problem Solving

In this section, we present in detail each phase in the cycles of constraint-based constraint acquisition and solution construction/refinement model of information dialogues: constraint acquisition, solution construction, solution evaluation, and solution modification.

Constraint Acquisition

In the constraint acquisition phase, the system acquires constraints to update the problem definition which is modeled as a CSP. The system gathers constraints through (1) recognition of constraints from user input, (2) requesting constraints from the user, or (3) proposing constraints for user to confirm. Constraints gathered from (1) are user-initiated constraints. Constraints gathered through (2) and (3) are system-initiated constraints. The system's requests and proposals are initiated based on the recommended strategies from the solution modification phase, incorporating its knowledge of the solution status, domain-specific task solving knowledge sources, and user profiles. The user's answer to the system-initiated requests results in new constraints being added. As a cooperative agent, system-initiated proposals need to be negotiated with the user before they are finalized in the problem definition. In contrast, user-initiated constraints are generally incorporated into the current CSP the moment they are recognized without negotiation.

Currently, the strengths of the acquired constraints are deduced based on the linguistic cues in the user's utterances. From a corpus analysis of naturally occurring dialogues (Transcripts 1992), we classify linguistic cues into three strengths, `required`, `strong`, and `weak`. For example, we assign a `required` strength to constraints expressed through *I need to*, a `strong` strength to *it'd better be*, and a `weak` strength to *maybe* or *it could be*. When no linguistic cues are available, constraints get default strengths from user profiles, which are calculated based on corpus analysis of distributions of attributes and distributions of relaxed or modified attribute-value pairs in the corpus. The current constraint strength recognition mechanism could be extended by taking into account conversation circumstances and endorsing confidence measures as discussed in (Elzer, Chu, & Carberry 1994).

Solution Construction

We use solution synthesis techniques (Tsang & Foster 1990; Beale 1997) to generate all solutions to a CSP by iteratively combining partial answers to arrive at a complete list of all correct answers. In solution synthesis, the variables in a CSP are represented as the base level nodes in a graph (SS-graph). Subsets of base level nodes are combined yielding higher level nodes which represent legal compound labels satisfying k-variable constraints. Partial solutions for a

subset of constraints are represented by the legal compound labels at the highest nodes covering the participating variables. The arcs represent the combination method used for combining lower level nodes into higher level nodes. Through solution synthesis, all assignments of values to variables that satisfy the problem's constraints are produced. Often, this list is then rated according to some separate criteria in order to pick the most suitable answer. Solution synthesis is applicable for problems when all possible solutions are required and for optimization problems.

We extend the solution synthesis technique in two ways: (1) we adapt the technique to dynamic CSPs, and (2) we integrate solution synthesis with constraint hierarchy.

Dynamic solution synthesis Applications that utilize solution synthesis are typically static CSPs (Tsang & Foster 1990; Beale 1997). A constraint-based model of interaction, however, is a DCSP: constraints and variables are dynamically added or removed during interaction in forming the problem definition and constructing a solution. Figure 1 demonstrates how solution synthesis is used and dynamically updated for constructing partial parallel solutions based on incrementally acquired constraints from dialogue excerpt 1 where the user specifies his or her constraints for a flight:

Dialogue excerpt 1:
User:
(U1) I need to reserve a flight to Dallas.
(U2) Maybe American Airlines.
(U3) It'd better be a night flight.
System:
(S1) American Airlines do not have any night flights to Dallas.

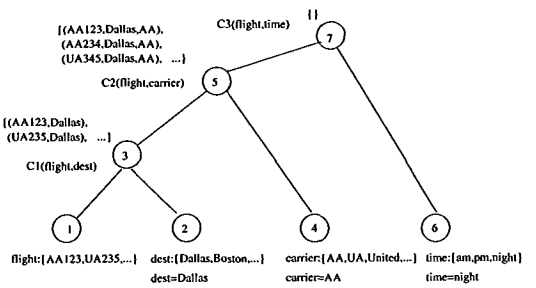

Figure 1: Solution synthesis

Through constraint acquisition, the system acquires new variables and constraints from utterances U1 to U3. The variables acquired include `flight`, `dest`, `carrier`, and `time`. The user constraints acquired include (1) the `dest` being Dallas with a `required` strength, (2) the `carrier` being American Airlines with a `weak` strength, and (3) the travel `time` being night with a `strong` strength. For utterance U1, the variables `flight` and `dest` are posted to the solution space. Their initial domains are shown at nodes 1 and 2. The constraint that the flight have a destination of Dallas reduces the domain at node 2, with the synthesized solutions shown at node 3 containing all flights with a destination of Dallas as constrained by the database constraint $C_1(flight, dest)$[1][2], which specifies the functional dependency relation between `flight` and `dest` in the database. For utterance U2, the variable `carrier` is posted to the solution space (node 4). The constraint that the carrier be American Airlines reduces its domain. This can then be synthesized with node 3 to produce a set of answers at node 5 which include all American Airlines flights to Dallas, constrained by the database constraint $C_2(flight, carrier)$. Similarly, the variable travel `time` is posted to the solution space (node 6). After domain reduction, it is synthesized with node 5 to produce a set of answers at node 7 which include all American Airlines flights to Dallas departing at night, constrained by the database constraint $C_3(flight, time)$. In this dialogue, the final set of flights is empty.

In the above example, solution synthesis is extended by incrementally adding variables and constraints. Partial solutions are constructed and maintained in the dynamically updated SS-graph as new variables and constraints are introduced. In general, solution synthesis can be extended for DCSPs through operations for adding/removing variables and adding/removing/modifying constraints (Tsang & Foster 1990).

Adding or removing variables affects the structure of the SS-graph. In our system, added variables are simply appended to the tail of the ordered nodes at the base level, but synthesized with the top level nodes of the current SS-graph. Adding one variable in this way to an existing N-variable SS-graph involves constructing two extra nodes, one at the base level for the new variable, and the other at level $N+1$ for the top level node. Removing variables from a SS-graph is in general complicated. Basically, when a node which represents the domain of the deleted variable is removed, all the nodes which are ancestors to the node must be either deleted or re-constructed. In our model of problem solving for the information domain, however, a variable is added into the solution space as a result of the negotiation process between the system and the user; variables never are deleted.

Adding, relaxing or modifying a constraint affects the size of nodes in the solution synthesis graph. Adding or tightening a constraint involves possible deletion of elements in some nodes. Relaxing con-

[1]Constraints C_1, C_2 and C_3 are simplified database constraints for illustration purposes. Actual database constraints can be n-ary rather than binary as in our illustration.

[2]This is where the constraint-based problem solver would normally interact with the back-end database.

straints involves possible addition of elements in some nodes. (Qu Forthcoming) gives the details on operations for updating the sets of variables and constraints and their complexity analysis.

Solution synthesis with constraint hierarchy The information of the preferential choices specified by a constraint hierarchy can be encoded in a graph such as an SS-graph with an incrementally maintained value called the *walkabout strength* (Maloney 1991). Specifically, every node in the SS-graph is annotated with a walkabout strength, which indicates the strength of the weakest constraint in the current graph that could be removed from the graph to allow some other constraints to be enforced by changing that node. The walkabout strength of a node may reflect the existence of a constraint quite far away in the SS-graph. Thus, the walkabout strengths encapsulate information for updating the SS-graph, which would otherwise have to be acquired by traversing the graph.

Walkabout strengths of nodes in an SS-graph are calculated by looking at the strengths of constraints in which the nodes participates, and the strengths of all the input nodes. The walkabout strength of a node in an SS-graph is defined as follows:

- if a node N is a base level node representing a variable without any constraint over it, then it gets a system-supplied walkabout strength `required`. This technicality simply means that once a variable is added to the graph, it stays there and never gets removed.
- if a node N is a base level node representing a variable, and a domain constraint C constrains the size of the node, then its walkabout strength is the weaker of C's strength and the node's `required` walkabout strength supplied by the system.
- if a node N is not a base level node, and a constraint C constrains the size of the node, then its walkabout strength is the weaker of C's strength and the weakest walkabout strengths among all the input nodes that participate in generating N.

Walkabout strength annotation can be straightforwardly incorporated into the dynamic solution synthesis procedures by annotating each node with its walkabout strength when the node is being constructed or when the node is being updated as a result of constraint update, yielding an SS-graph annotated with walkabout strengths.

For instance, suppose the constraints participating in solution synthesis have the following strengths (S) for the CSP of dialogue excerpt 1[3]:

strengths for user constraints:
$S(dest = Dallas)$ = `required`
$S(carrier = AA)$ = `weak`
$S(time = night)$ = `strong`
strengths for database constraints:
$S(C_1(flight, dest))$ = `required`
$S(C_2(flight, carrier))$ = `required`
$S(C_3(flight, time))$ = `required`

According to the walkabout strength definition, the walkabout strengths (WS) of the nodes in the SS-graph for the CSP in Figure 2 are calculated as follows:

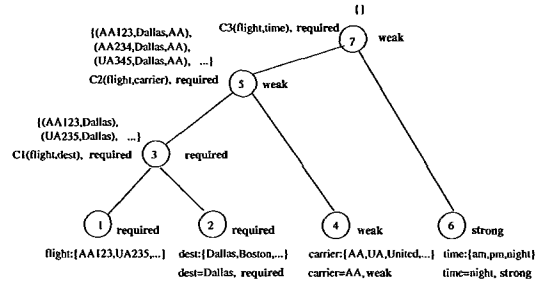

Figure 2: Solution synthesis with walkabout strengths

WS_{Node1} = `required`
$WS_{Node2} = min\{$**required**$,S(dest = Dallas)\}$
= `required`
$WS_{Node3} = min\{WS_{Node1}, WS_{Node2}, S(C_1)\}$
= `required`
$WS_{Node4} = min\{$**required**$,S(carrier = AA)\}$
= `weak`
$WS_{Node5} = min\{WS_{Node3}, WS_{Node4}, S(C_2)\}$
= `weak`
$WS_{Node6} = min\{$**required**$,S(time = night)\}$
= `strong`
$WS_{Node7} = min\{WS_{Node5}, WS_{Node6}, S(C_3)\}$
= `weak`

Solution Evaluation

Solution evaluation characterizes the solution space into different solution situations. We use an evaluation function to characterize the solution space. Specifically, the evaluation function evaluates the top level node of the SS-graph, which records all the partial solutions obtained so far, to three possible values. If the top node evaluates to `NIL`, then an over-constrained

[3]It is important to note the difference between walkabout strengths for variables (base level nodes in an SS-graph) and the strengths for user constraints. Walkabout strengths assigned to variables such as `dest` and `carrier` are based on the definition of walkabout strength for an SS-graph. The system-supplied `required` walkabout strength reflects the claim in our model that once an attribute is introduced as a variable into the problem definition, the attribute (variable) itself never gets dropped from the problem definition (SS-graph), even though the constraints over the variable may get modified. The fact that the destination city being a certain city is usually required for the information task while the carrier constraint can be optional is reflected by the strengths assigned to user constraints, e.g., $S(dest = Dallas)$ =`required`, and $S(carrier = AA)$ =`weak`.

1 PROCEDURE LocateRelaxCandidate (NodesOfSS)
2 RelaxCandidate < – NIL;
3 TopNode < – top level node from NodesOfSS
4 while RelaxCandidate not found,
(4.a) if walkabout strength of TopNode resulted from a constraint C,
then RelaxCandidate < – constraint C;
(4.b) else if walkabout strength resulted from an InputNode, then TopNode < – InputNode;
(4.c) else if there is a tie between the candidate InputNodes or constraints,
then TopNode < – a randomly selected InputNode;
5 return RelaxCandidate;

Figure 3: Use of walkabout strengths for relaxation

situation is detected, which suggests that some constraints need to be relaxed to get a solution. If the top node evaluates to a set whose number of solutions exceeds a pre-defined threshold k (e.g., $k = 5$ is a good heuristic number), then the problem is under-constrained, which suggests that a cooperative system should employ initiative taking strategies to help the user deal with the under-constrained situations. If the top node evaluates to a set whose number of solutions is within a pre-defined threshold k, then the solution set is small enough, and the system can decide to present the solutions to the user at this point.

Consider again Figure 2 for dialogue excerpt 1. The constraints from utterance U2 are incorporated into the SS-graph resulting in a new top level node 7, which evaluates to `NIL`. This signals an over-constrained situation.

Solution Modification

When no solutions can be found to satisfy the user's information needs, a cooperative system need to provide relaxed solutions in addition to informing the user of the over-constrained situation. The solution modification module is invoked when over-constrained situations are detected. Its task is to support strategies for relaxation in over-constrained situations to help the user with the problem definition. Efficient instantiation of the parameters in these strategies are made possible by exploiting knowledge sources recorded in the solution synthesis graph, and by interacting with domain knowledge sources and user models. Our model recognizes the fact that over-constrained situations result from *interactions* between constraints, e.g. between user constraints and database constraints, and that the partial solutions in an SS-graph encode the effects of such interactions.

Knowledge sources for constraint relaxation The knowledge sources we explore for resolving over-constrained situations include the walkabout strengths and node density ratios.

Constraint hierarchy and walkabout strengths We use the constraint hierarchy as a systematic way of ordering the importance of the constraints. As we mentioned earlier, such preferential information can be encoded as walkabout strengths of the nodes in an SS-graph, while the partial solutions are computed. The walkabout strength of a node indicates the strength of the weakest constraint in the current solution graph that could be removed or modified from the solution graph to allow some other constraints to be enforced by changing that node. When an over-constrained situation is detected, the system tries to satisfy the constraints with higher strengths in the constraint hierarchy, while relaxing constraints with lower strengths first. Figure 3 describes the procedure for locating the constraint with the weakest walkabout strength as the relaxation candidate. The incremental running time of this algorithm grows linearly with the number of variables in the CSP.

Solution synthesis network structure The sizes of the nodes in the solution synthesis network, which encode partial solutions, yield another source of information that we can exploit. For example, relaxation of constraints is typically most advantageous at a point where a solution synthesis node yields a partial solution that is relatively small compared to its inputs. This indicates that some constraint has removed many possible solutions. We introduce the notion of *node density* with respect to its input nodes to record the number of compound labels satisfying participating constraints at this node over the number of possible compound labels as a result of product combination of the values from its input nodes. The node density ratio reflects the combined efforts of all the constraints effective over the compound labels of a node. The smaller the node density ratio, the more constraining the participating constraints. The node density information can be combined with the walkabout strengths in identifying candidates for relaxation. We give priority to the walkabout strengths over node density ratios, and use the latter only for breaking the ties among candidates. In Figure 3, we use a simple tie-breaking heuristic in preferring input nodes over constraints with the same strength. This heuristic reflects an artifact of our domain where the user constraints are usually domain reduction constraints represented at the base level of the SS-graph, while domain and database constraints are usually exerted at higher level nodes to reduce the set of possible solutions. Since domain and database constraints usually have **required** strengths, the relaxation is likely to happen with the user constraints over base level nodes. Using the combined knowledge sources, we introduce another heuristic to break the tie between candidate nodes. The revised tie-breaking act becomes:

(4.c') if there is a tie between the candidate InputNodes or constraints,
then TopNode < – InputNode with the lowest node density ratio;

Relaxation example Relaxation candidates can be identified based on the knowledge sources we just discussed.

Relaxation using walkabout strengths Consider again the SS-graph for dialogue excerpt 1 repeated here in Figure 4. When an over-constrained situation is detected, solution modification is evoked to identify the candidate constraint for relaxation. The traversal starts from the top node 7. Since the **weak** walkabout strength of node 7 results from the walkabout strength of node 5, node 5 is chosen as the candidate node. This process repeats recursively until node 4 is identified as a candidate node. Since the **weak** walkabout strength of node 4 results from the user constraint carrier being American Airlines, this constraint is identified as the constraint for relaxation.

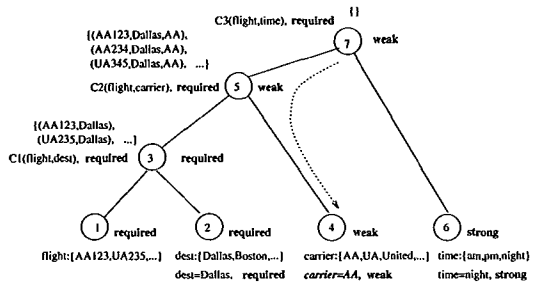

Figure 4: Relaxation based on walkabout strengths

Relaxation based on solution synthesis network structure and walkabout strength Suppose in dialogue excerpt 1, instead of saying in U3 *"It'd better be a night flight"*, the user said *"I could take a night flight"*, then during constraint acquisition, the strength of the constraint travel time being night would be recognized as **weak** as illustrated in Figure 5 instead of **strong** (in Figure 4). Consequently, the walkabout strength for node 6 is **weak** instead of **strong**. Also suppose that as a result of the constraint $C_1(flight, dest)$, the number of legal solution tuples at node 3 is 100. Since the number of possible solution tuples at node 4 is 1 due to the constraint $carrier = AA$, the number of possible compound labels at node 5 as a result of product combination of its input nodes (nodes 3 and 4) is 100. Suppose after satisfying the constraint $C_2(flight, carrier)$, only 87 tuples remain at node 5, then node 5 has a node density ratio (ND) of 87/100. At node 6, the constraint $time = night$ results in a ND ratio of 1/3. Now when traversal starts from the top node 7, both node 5 and node 6 are tied as relaxation candidates based on walkabout strengths. Our revised tie-breaking rule (4.c') chooses node 6, as the constraints at node 6 more tightly constrain the solution tuples at that node. As the **weak** walkabout strength of node 6 results from the user constraint travel time being night, this constraint is identified as the constraint for relaxation.

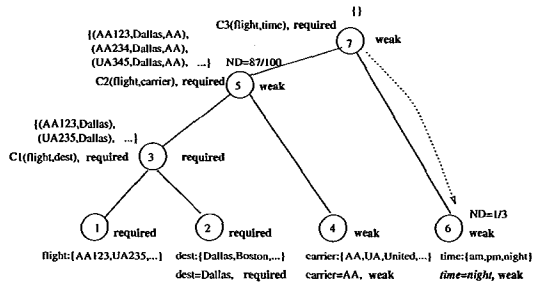

Figure 5: Relaxation based on walkabout strengths and node density ratios with walkabout strength of node 6 changed to **weak**

Evaluation of relaxation candidates Simply identifying a constraint for relaxation does not guarantee that a solution will be found. As a cooperative agent who has access to the back-end databases to evaluate the relaxation candidates, the system should make sure that relaxation of the candidate constraint will lead to a solution. In our system, when a user constraint is chosen for relaxation, the user constraint $v_i = val$ is replaced by assigning all the possible domain values to v_i, with the strength **required**, as no further relaxation is possible when all domain values are considered. Then all the ancestor nodes to the current node are re-computed, yielding a new solution synthesis graph. If a solution is found at the top node, then such a relaxation is valid; the system can in turn initiate appropriate dialogue acts to inform the user of such relaxation proposal. If however, after such a relaxation, the relaxed problem is still over-constrained, a new relaxation candidate is further identified by repeating the procedure in Figure 3 until a solution is found.

Before a solution is found, a set of constraints may have been relaxed. If there exists a solution for all the **required** constraints of the CSP, our procedure is guaranteed to find that solution by incrementally relaxing constraints of weaker strengths. We define solution optimality for a relaxed CSP as (1) satisfying the constraints with higher strengths first, and (2) satisfying as many original constraints as possible[4]. The incremental relaxation procedure guarantees the first condition of optimality. The second condition of optimality is satisfied by ranking all solution tuples based on the order of maximally satisfying (a) the original set of user constraints, (b) the constraints in user models, and (c) the constraints in domain knowledge. The solution satisfying more constraints in a set is ranked higher than other solutions. Such an ordering could be further augmented by introducing metrics

[4]Note that in over-constrained situations, solution ordering only becomes meaningful when the system has the turn.

such as closeness of the matched values (Elzer, Chu, & Carberry 1994) and semantic distances between solutions (Pieraccini, Levin, & Eckert 1997).

Cooperative Response Generation

The focus of solution modification for over-constrained problems is to identify the candidate constraints for relaxation that will yield solutions to resolve the over-constrained situations. The system will be considered uncooperative, however, if it modifies the problem definition without the user's consent. Thus cooperative principles require that the proposed modifications by the system be accepted by the user. This requirement causes the system to go back to the constraint acquisition phase to interact with the user by generating natural language utterances to negotiate the changes in the problem definition. The selection of a particular dialogue act should be based on the system's existing knowledge of the solution status and the degrees of system initiative. We conducted corpus analysis of naturally occurring information dialogues (Transcripts 1992) to identify problem solving initiative-taking dialogue acts for various dialogue situations. Detailed enumeration of these dialogue acts and the criteria under which they occur are discussed in (Qu Forthcoming). Three dialogue acts, presented in an increasing degree of system initiative, could be selected to resolve the over-constrained situations.

Dialogue act 1: Request a new value for a variable. The system takes the initiative in indicating the constraint that requires relaxation. An example NL utterance for this dialogue act is: ***Does any other airline work for you?*** in which the system proposes replacing the previous constraint *carrier* = *AA* with some new constraint. Specific airline values provided by the user or simple positive answer to this request will result in an update in the problem definition and in new solutions in the solution construction phase.

Dialogue act 2: Propose a new value for a variable. The system takes the initiative in proposing a new constraint to replace an existing one. The proposed instantiation of the variable is obtained from the optimal solution obtained during solution modification. Being a cooperative agent, the system needs to obtain the user's acceptance first before the constraint can be finalized into the SS-graph. An example NL utterance for this act is: ***Does United Airlines work for you?*** in which the system proposes replacing the previous constraint *carrier* = *AA* with the new constraint *carrier* = *United*. Acceptance of this utterance by the user will result in an update in the problem definition and in new solutions in the solution construction phase.

Dialogue act 3: Propose a new value for an attribute and inform user of solutions. The system takes the problem solving initiative one step further than that in act 2, in also informing the optimal solutions resulted from such proposed modification. An example NL utterance for this act is ***United Airlines has one leaving at 8:45pm.*** in which the system proposes replacing the previous constraint *carrier* = *AA* with the new constraint *carrier* = *United*, and informs the user that once such constraint modification is made, solution is possible.

A possible future extension is to generate reasons to support the above relaxation dialogue acts, e.g., ***Does United Airlines work for you? Since no other airlines have such flights.*** Such supporting rationales could be generated in our framework by comparing the attribute values between different solutions.

Related Work

Generation of cooperative response in over-constrained situations has been addressed in many human-computer dialogue systems, e.g., (Kaplan 1979; Abella, Brown, & Buntschuh 1996; Pieraccini, Levin, & Eckert 1997; Litman, Pan, & Walker 1998). Our work is similar to many of these systems in (1) using solution evaluation in identifying over-constrained situations, and (2) interleaving information gathering and solution update. In identifying relaxation candidates, however, existing systems employ heuristics (e.g., relaxation based on constraint weights (Abella, Brown, & Buntschuh 1996; Pieraccini, Levin, & Eckert 1997)) which do not take into account the interaction effects of constraints and lack a principled way to explore the interaction effects in arriving at relaxation candidates. The solution synthesis-based problem solver in our model provides a mechanism where the effects of constraint interaction are maintained as partial solutions, which can be exploited as the system's knowledge sources when over-constrained situations occur. The use of walkabout strengths provides a principled way to explore the SS-graph for relaxation candidates in linear running time. In addition, the SS-graph supports re-use of many partial solutions in constructing or modifying solutions.

Relaxing over-constrained queries and possibly proposing relaxation modification is also analogous to previous work on detecting invalid beliefs/plans and suggesting possible alternative solutions, e.g., (Joshi, Webber, & Weischedel 1984; van Beek 1987; Chu-Carroll & Carberry 1994). In particular, the cycles in the framework of incremental constraint acquisition and solution construction/refinement are similar to the *Propose-Evaluation-Modify* cycle for collaborative response generation in (Chu-Carroll & Carberry 1994). However, the above work addresses cooperative response generation based on modeling of plans and beliefs, while our work focuses on using CSP-based models for problem solving. The CSP-based problem solver could be incorporated with other aspects of dialogue processing, such as belief and plan modeling, to formulate different types of cooperative responses.

Constraint computation has been used for modeling the dynamic nature of discourse in (Donaldson & Co-

hen 1997). The focus of their work, however, is to use local repair techniques to generate solutions in managing turn-taking goals.

Conclusions and Future Work

In this paper, we presented a novel way to integrate and extend the AI techniques of constraint satisfaction, solution synthesis and constraint hierarchy for the problem of cooperative response generation. We illustrated that the knowledge sources in an SS-graph can be exploited in a principled and efficient way for suggesting relaxation candidates in over-constrained situations.

Our ongoing work addresses several open issues of the CSP model. The current framework for incremental problem definition and solution construction requires database retrieval whenever database constraints are encountered, e.g., constraints C_1-C_3. Access to back-end databases incur communication cost and retrieval cost, which can be expensive in excess. In many tasks, it is reasonable to assume that an information need is not over-constrained without actually accessing the database (e.g., at the beginning of a dialogue when few constraints are introduced) by specifying a minimal set of constraints for each particular topic before database retrieval occurs. In fact, such an assumption has been generally followed in most existing information systems, in which the systems collect user constraints to a certain set before interacting with the back-end databases. However, the drawbacks of such an assumption are (1) that identification of over-constrained situations may be delayed, and (2) that partial solutions will not be available to support identification of relaxation candidates in an efficient and principled manner. Our proposal is to use a caching mechanism to store portions of retrieved results, and to access this cache to reduce communication and retrieval costs while building an SS-graph. The trade-off between space demand and cost reduction of utilizing the caching mechanism requires further investigation.

Adoption of different dialogue acts can affect the dialogue efficiency and effectiveness of human-computer dialogue. For example, we hypothesize that dialogue act 1 is less efficient than dialogue acts 2 and 3 because new user constraints obtained from a user's response to dialogue act 1 may again lead to an over-constrained situation. Our ongoing work focuses on evaluating the effectiveness and efficiency of the dialogue acts with respect to factors such as problem structure, solution structure, user profiles, and constraints in domain data.

Acknowledgments

The work reported here is based on the first author's dissertation research at Language Technologies Institute, Carnegie Mellon University. We thank Nancy Green, Jaime Carbonell, Barbara Di Eugenio and three anonymous reviewers for their comments on earlier versions of this paper.

References

Abella, A.; Brown, M. K.; and Buntschuh, B. 1996. Development principles for dialogue-based interfaces. In *Proceedings of ECAI'96 Workshop on Dialogue Processing in Spoken Language Systems*, 1–7.

Beale, S. 1997. *HUNTER-GATHERER: Applying Constraint Satisfaction, Branch-and-Bound and Solution Synthesis to Computational Semantics.* Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University.

Chu-Carroll, J., and Carberry, S. 1994. A plan-based model for response generation in collaborative task-oriented dialogues. In *Proceedings of the AAAI-94*, 799–805.

Donaldson, T., and Cohen, R. 1997. A constraint satisfaction framework for managing mixed-initiative discouse. In *Proceedings of 1997 AAAI Spring Symposium on Mixed-Initiative Interaction.*

Elzer, S.; Chu, J.; and Carberry, S. 1994. Recognizing and utilizing user preferences in collaborative consulation dialogues. In *User Modeling Conference.*

Joshi, A.; Webber, B.; and Weischedel, R. M. 1984. Living up to expectations: computing expert responses. In *Proceedings of the AAAI-84*, 169–175.

Kaplan, S. J. 1979. *Cooperative responses from a portable natural language data base query system.* Ph.D. Dissertation, School of Computer and Information Science, University of Pennsylvania.

Litman, D. J.; Pan, S.; and Walker, M. A. 1998. Evaluating response strategies in a web-based spoken dialogue agent. In *COLING/ACL-98*, 780–786.

Maloney, J. H. 1991. *Using Constraints for User Interface Construction.* Ph.D. Dissertation, Department of Computer Science and Engineering, University of Washington.

Pieraccini, R.; Levin, E.; and Eckert, W. 1997. *AMICA*: the *at & t* mixed initiative conversational architecture. In *Proceedings of EUROSPEECH 97.*

Qu, Y. Forthcoming. A constraint-based model of cooperative response generation in spoken information systems.

Transcripts, S. 1992. Transcripts derived from audiotape conversations made at SRI International, Menlo Park, CA. Prepared by Jacqueline Kowtko under the direction of Patti Price.

Tsang, E., and Foster, N. 1990. Solution synthesis in the constraint satisfaction problem. Technical Report Technical Report CSM-142, Department of Computer Science, University of Essex.

van Beek, P. 1987. A model for generating better explanations. In *Proceedings of the ACL*, 215–220.