A Constraint-Based Model of Mixed-Initiative Dialogue for Information-Seeking Interactions

Yan Qu

December 2001

CMU-LTI-01-XXX

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15232

Submitted in partial fulfillment for the requirements for the degree of Doctor of Philosophy

Thesis Committee:
Jaime Carbonell, Co-chair
Nancy Green, Co-chair, University of North Carolina at Greensboro
Lori Levin
Alex Waibel
Barbara Di Eugenio, University of Illinois at Chicago

Copyright © 2001 Yan Qu

Content

Chapter 1 Introduction 9
1.1. Definitions 9
1.2. Problem Statement 10
1.3. Thesis Statement 12
1.4. Thesis Contributions 14
1.5. Overview of the Thesis 14
Chapter 2 Related Work on Mixed-Initiative Interaction 17
2.1. Initiative Tracking in Mixed-Initiative Dialogues 17
2.2. Cooperative Response Generation in Mixed-Initiative Interaction 18
Chapter 3 Constraint Processing Techniques 26
3.1. Definitions 26
3.2. Finding Solution Graphs 31
3.2.1. Walkabout strengths 31
3.2.2. DeltaBlue algorithm for finding solution graphs 33
3.3. Computing Solutions Through Solution Synthesis 34
3.3.1. The Essex algorithm 34
3.3.2. Hunter-Gatherer 35
3.4. Ordering Heuristics 37
Chapter 4 Constraint-Based Model of Problem Solving 40
4.1. Dynamic Solution Synthesis with Constraint Hierarchy 40
4.1.1. Dynamic solution synthesis 41
4.1.2. Solution synthesis with a constraint hierarchy 44
4.2. Identification of Relaxation Candidates 48
4.3. Usability of the Proposed Techniques 49
Chapter 5 Heuristics for Question Selection and Relaxation Candidate Selection 51
5.1. Heuristics for Question Selection 51
5.1.1. An example problem 52
5.1.2. Upper bound and lower bound 55
5.1.3. Attribute selection based on maximum branching 58

5.1.4. Attribute selection based on maximum information gain 58
5.1.5. Attribute selection based on the top-down walk of a constraint hierarchy 60
5.1.6. Attribute selection based on fixed ordering 61
5.1.7. Baseline attribute selection method 61
5.1.8. Comparison of the attribute selection methods 61
5.2. Heuristics for Constraint Relaxation 63
5.2.1. Relaxation candidate selection based on constraint hierarchy 63
5.2.2. Relaxation candidate selection based on solution size 66
Chapter 6 Evaluation Design: Simulated Tasks 67
6.1. Hypotheses 67
6.2. Information-Seeking Task 68
6.3. Database 68
6.4. Construction of Test Collections 68
6.4.1. Information need 69
6.4.2. Levels of task difficulty 69
6.4.3. Constraint strength distributions 71
6.4.4. Test collections 73
6.5. Design of the Evaluation 75
6.5.1. Combinations of the heuristic methods 75
6.5.2. Experimental parameters 76
6.6. Performance Measures 76
6.6.1. Task completion 76
6.6.2. Task success 77
6.6.3. Dialogue efficiency 79
6.7. Statistical Hypothesis Testing 80
6.7.1. Hypothesis testing 80
6.7.2. The paired sample *t*-test 81
6.7.3. Analysis of variance (ANOVA) 81
Chapter 7 Result Analysis of Simulated Experiments 83
7.1. Result Analysis: The Effect of Task Difficulty 83
7.1.1. Task difficulty and question selection efficiency 83

7.1.2. Task difficulty and task success ... 94
7.1.3. Task difficulty and relaxation efficiency ... 98
7.1.4. Task difficulty: summary ... 104
7.2. Result Analysis: The Effect of Goal-State Size ... 105
7.2.1. Goal-state sizes and question selection efficiency ... 105
7.2.2. Goal-state sizes and task success ... 116
7.2.3. Goal-state sizes and relaxation efficiency ... 121
7.2.4. Goal-state sizes: summary ... 129
7.3. Result Analysis: The Effect of Interval Selection ... 131
7.3.1. Interval selection and question selection efficiency ... 132
7.3.2. Interval selection and task success ... 134
7.3.3. Interval selection and relaxation efficiency ... 134
7.3.4. Interval selection: summary ... 135
7.4. Result Analysis: The Effect of Task Complexity ... 135
7.4.1. Task complexity and question selection efficiency ... 136
7.4.2. Task complexity and task success ... 138
7.4.3. Task complexity and relaxation efficiency ... 138
7.4.4. Task complexity: summary ... 141
7.5. Summary and Implications for Usability Testing ... 142
Chapter 8 Architecture of a Cooperative Mixed-Initiative Information Dialogue System ... 149
8.1. Task Description ... 149
8.2. The Airline Domain Model ... 149
8.2.1. ER diagram ... 150
8.2.2. Relational models ... 151
8.3. The Architecture of an Initiative-Taking Information Dialogue System ... 153
8.3.1. Task knowledge representation ... 155
8.3.2. Semantic interpretation ... 156
8.3.3. Problem-solving components ... 157
8.3.4. Dialogue management ... 164
8.3.5. Form-based response generation ... 176

8.4. An Illustrative Example 178
Chapter 9 Usability Study: Experimental Design and Result Analysis 184
9.1. Experimental Design and Data Collection 184
9.1.1. Hypothesis 184
9.1.2. Task scenarios 185
9.1.3. System settings 185
9.1.4. Users 188
9.1.5. Pilot runs 189
9.1.6. Performance measures 189
9.1.7. Data collection procedure 190
9.2. Result Analysis 191
9.2.1. The effect of task complexity 191
9.2.2. Under-constrained tasks vs. over-constrained tasks 193
9.2.3. Comments from users 196
Chapter 10 Conclusions and Future Work 198
10.1. Conclusions 198
10.2. Future Work 201
10.2.1. Initiative and dialogue strategy adaptation 201
10.2.2. Applications to other domains 202
10.2.3. Scalability 203
10.2.4. Application to spoken dialogue systems 203
10.2.5. Expanded usability evaluation 203
10.2.6. Selection of initiative-taking dialogue actions 205

Appendix 1: Pre-Experiment Questionnaire 206
Appendix 2: Instructions for Usability Study 207
Appendix 3: Example Task Scenarios for Training Sessions 209
Appendix 4: Post-Task User Questionnaire 211

Bibliography 213

List of Figures

Figure 1: An example solution graph. An arrow indicates the output of each constraint. 32

Figure 2: Solution synthesis using the Essex algorithm. The CSP has four variables A, B, C, and D, and three constraints A<B, A=C, and C>D. Adjacent nodes at each level are combined together for computing all partial solutions. The final assignment for the CSP is A=2, B=3, C=2, D=1. 35

Figure 3: Solution synthesis using HG. The CSP has four variables A, B, C, and D, and three constraints A<B, A=C, and C>D. Closely-constrained variables or subgraphs are ordered in a way to constrain each other to reduce the sizes of higher-level nodes. The final assignment for the CSP is A=2, B=3, C=2, D=1. 36

Figure 4: Procedure for updating a constraint in a solution synthesis graph. 43

Figure 5: Correspondence between a constraint graph and a solution synthesis graph: (a) is a CSP with two variables v_1 and v_2, and a constraint between the two variables Cv_1v_2. (b) represents how the binary CSP can be solved using solution synthesis. Variables v_1 and v_2 are placed at the base level of the SS-graph. The higher-level node N_{v1v2} is constructed by computing all legal tuples of v_1 and v_2 satisfying the constraint Cv_1v_2. The solution synthesis process is treated as a special constraint SS, as represented in (c). 45

Figure 6: Solution synthesis with walkabout strength. The CSP in (a) has three variables V_1, V_2 and V_3. The three variables have stay constraints with strengths required, weak, and strong respectively. The three binary constraints C_{V1V2}, C_{V2V3}, and C_{V1V3} all have strengths required. (b) represents how the CSP can be solved using solution synthesis with variables in the ordering of V_1, V_2 ,V_3 at the base level. The nodes in the solution synthesis graph are annotated with their respective walkabout strengths. 47

Figure 7: Solution synthesis with walkabout strength. The CSP in (a) has three variables V_1, V_2 and V_3. The three variables have stay constraints with strengths required, weak, and strong respectively. The three binary constraints C_{V1V2}, C_{V2V3}, and C_{V1V3} all have strengths required. (b) represents how the CSP can be solved using solution synthesis with variables in the ordering of V_1,V_3,V_2 at the base level. The nodes in the solution synthesis graph are annotated with their respective walkabout strengths. 48

Figure 8: Example partial solution set with six solutions for further disambiguation in the airline domain. 52

Figure 9: Decision tree when departure city (DeptCity) is requested first. 53

Figure 10: Decision tree when departure time (DeptTime) is requested first. 54

Figure 11: Use of walkabout strengths for relaxation. 65

Figure 12: Relaxation in a solution synthesis graph with walkabout strength. The dotted line indicates how walkabout strengths guide relaxation. 65

Figure 13: Relaxation in a solution synthesis graph with walkabout strength. The dotted line indicates how walkabout strengths guide relaxation. 65

Figure 14: Using dynamic solution size information for breaking ties. 66

Figure 15: Sample distributions of constraint strengths for attributes in the airline domain. 73

Figure 16: Average numbers of questions elicited by the system across different task difficulty levels with different question selection methods. 85

Figure 17: Sorted flight distributions with respect to the attributes Dow, DeptTime, and DeptCity in the Northwest flight database. 91

Figure 18: Average system times used for question selection with different question selection methods. 92

Figure 19 : Average kappa scores across task difficulty levels with different question selection methods**.** 95

Figure 20: Average numbers of relaxation requests with different relaxation selection methods. The question selection method is fixed as the random selection method. 100

Figure 21: Average numbers of relaxation requests with different selection methods. The question selection method is fixed as the constraint hierarchy based method in (a) and as the maximum branching method in (b). 101

Figure 22: Average numbers of relaxation requests with the relaxation methods fixed as the simple backtracking method in (a), as the constraint hierarchy based backtracking in (b), and the minimum solution size based backtracking in (c). 102

Figure 23: Average system times for selecting relaxation requests with the question selection fixed as the random selection method. 103

Figure 24: Average numbers of question elicited by the system at different goal-state sizes for the Easy task difficulty problems. 108

Figure 25: Average numbers of questions elicited by the system at different goal-state sizes for the Medium task difficulty problems. 109

Figure 26: Average numbers of questions elicited by the system at different goal-state sizes for the Hard task difficulty problems. 112

Figure 27: Average system times used for question selection for the Easy task difficulty problems. 113

Figure 28: Average system times used for question selection for the Medium task difficulty problems. 115

Figure 29: Average system times used for question selection for the Hard task difficulty problems. 115

Figure 30: Average kappa scores at different goal-state sizes for the Easy task difficulty problems. 117

Figure 31: Average kappa scores at different goal-state sizes for the Medium task difficulty problems. 119

Figure 32: Average kappa scores at different goal-state sizes for the Hard task difficulty problems. 120

Figure 33: Average numbers of relaxation candidates for the Easy task difficulty problems. 122

Figure 34: Average numbers of relaxation candidates for the Medium task difficulty problems. 124

Figure 35: Average numbers of relaxation candidates for the Hard task difficulty problems. 125

Figure 36: Average system time used for relaxation candidate selection for the Easy task difficulty problems. 126

Figure 37: Average system time used for relaxation candidate selection for the Medium task difficulty problems. 127

Figure 38: Average system time used for relaxation candidate selection for the Hard task difficulty problems. 129

Figure 39: ER diagram for the airline domain. Each rectangle represents an entity and each link represents a relation between entities. The numbers on the links specify the cardinality of the relations. 150

Figure 40: Architecture of the COMIX information system. The rectangles represent modules and the ovals represent stored data. The components within the dashed rectangle are concerned with constraint-based problem solving. The solid lines with arrows represent data flow and the dashed lines with arrows represent control flow. 154

Figure 41: The task knowledge hierarchy for retrieving one-way flights. Each node represents an object in the knowledge hierarchy. 156

Figure 42: Screen shot of the top-level form for retrieving one-way flights, with some of the form-based objects annotated with their names following the arrows. 158

Figure 43: Screen shot of an instantiated one-way flight request. 159

Figure 44: SQLs required for computing solutions for the example airline retrieval problem. 161

Figure 45: An example solution synthesis graph in the airline domain. 161

Figure 46: A complete set of SQLs recorded as views in the example solution synthesis graph. 163

Figure 47: Dialogue manager algorithm over an object *c*. 169

Figure 48: How knowledge sources support generation of DA1-3 in under-constrained situations. 172

Figure 49: How knowledge sources support generation of DA4-5 in over-constrained situations. 174

Figure 50: Flight query form from COMIX-UI. 186

Figure 51: System response form for over-constrained queries by COMIX-UI. 186

Figure 52: Flight query form from COMIX-MI. 187

Figure 53: System response form for over-constrained queries by COMIX-MI. 187

Figure 54: Result form when flights are found for both COMIX-UI and COMIX-MI. 188

Figure 55: Distributions of total completion time for the over-constrained round-trip tasks. 194

Figure 56: Distributions of system turns and user turns for the over-constrained round-trip tasks. 195

Figure 57: Kappa score distributions for the over-constrained round-trip tasks. 196

Chapter 1 Introduction

Information seeking and giving is essential in many applications involving human-computer interaction. With the advent of massive online information databases, the ability for a computer to provide information in a cooperative and efficient manner is beneficial to users of online information, and is imperative to the commercial success of a wide variety of applications of information systems. Information seeking is also important to many other applications, such as online commerce. For example, information seeking is an integral part of hotel or flight reservation systems and online stores. Designing and developing a cooperative and efficient dialogue between the user and information systems has been an important topic in research on cooperative dialogue systems, information systems, and human-computer interaction.

In practical applications of information systems, the user and the system do not have perfect and detailed models of each other. Since the user does not possess a detailed model of the domain knowledge and information in the system, the user's request for information may be under-constrained or over-constrained. As a result, under-constrained and over-constrained situations are abundant in information-seeking dialogues. It is important, therefore, that the system be able to inform the user of the under-constrained and over-constrained situations, initiate appropriate clarification questions in under-constrained situations, propose relaxation suggestions in over-constrained situation, and communicate such initiative-taking actions in a cooperative and efficient manner.

1.1. Definitions

In this work, we make the following definitions, which structure the information dialogue problem:

- *An information need* consists of a set of attributes and appropriate values for these attributes. We model the attribute value pairs as unary constraints over the attributes and the dependencies between the attributes as binary constraints. Without the loss of generality, we restrict our analysis and evaluation to unary and binary constraints.
- *A human-computer information seeking dialogue* is the process during which the user provides the system with his/her information need and the system provides solutions that satisfy the user's information need. The information exchange between the system and the user can be achieved through one single dialogue turn or through a sequence of dialogue turns.

- An *under-constrained problem* is defined as an information need with many solutions, due to the lack of adequate constraints to constrain the problem. In information-seeking dialogues, under-constrained situations occur when the information seeker's needs have yet to be presented (i.e., to be presented in later turns), or the information seeker's needs are under-specified.
- An *over-constrained problem* is defined as an information need with no solution, caused by failure to satisfy some individual constraint or failure to satisfy combinations of some constraints. In information-seeking dialogues, over-constrained situations occur when the restrictions and preferences of an information seeker cannot be satisfied at the same time. For example, suppose the information seeker is interested in knowing AA night flights to Dallas (e.g., *Does American Airlines have a night flight to Dallas?*), when there is no such flight, the information request is over-constrained within the domain of flight information.

1.2. Problem Statement

Many existing information dialogue systems adopt a two-phase approach: problem construction followed by solution construction and presentation, with the former concerned with the acquisition of the preferences and restrictions in a user's information needs, and the latter concerned with presenting solutions that satisfy the user's information needs (e.g., Danieli and Gerbino, 1995; Abella, et al., 1996; Litman, et al., 1998; Chu-Carroll, 2000). When we look at human-human information-seeking dialogues, however, we observe that the two phases are very much interleaved:

Dialogue excerpt 1[1]:

C1: I need an airfare for a proposal

A2: ok from where to where

C3: SFO to Kwalalampour

A4: ok and let's see

C5: say mid July

A6: mid July ok.

A7: now let's see if we get a quote here. ah full coach?

C8: yeah

A9: all right round trip it's twenty three thirty eight

C10: twenty two thirty eight that's coach

A11: that's coach

C12: now they can go business that far can't

A13: uh huh

C14: ok

A15: business is going to be round trip twenty six twenty six

C16: twenty six twenty six ok thank you very much

For example, consider dialogue excerpt 1 between a travel agent (A) and a customer (C) who are working together to obtain quotes for the customer. At first sight, the dialogue might be roughly divided into two parts - the problem construction part, where the dialogue participants work together to specify the information needs (turns C1 to A6), and the solution presentation part, where the travel agent presents solutions to the customer (turns A7 to C16). A closer analysis of the solution presentation part shows, however, that the information request is continuously refined before and during the presentation of solutions. For example, before offering a quote, the travel agent takes the initiative to refine the problem definition by clarifying whether the class is full coach (turn A7). While offering the quote, the travel agent again takes the initiative to refine the problem definition to include the travel type as round trip (turn A9). In turn C12, the customer takes the initiative to change the problem definition by changing the class to business, and the travel agent presents the solution to this new problem definition in turn A15.

As can be seen from dialogue excerpt 1, even during solution presentation, problem definition can be continuously refined. In fact, we have observed that assuming a two-phase dichotomy can lead to inefficient dialogue behaviors even for the human agent interacting with back-end databases, such as delayed identification of over-constrained problems.

Dialogue excerpt 2:

A1: ok what flights do you want to have him on?

C2: ok he would like to be on United flight one one one seven

A3: first of all he's going all the way through to Copenhagen?

C4: oh, that's right

---

[1] Dialogue excerpts 1 and 2 are taken from the SRI Transcripts (1992), which is a collection of transcripts of naturally occurring human-human travel reservation dialogues.

A5: and this is a connection in what city?

C6: ah Los Angeles

A7: and what time does that United flight leave?

C8: it's ah going out at uh two o'clock today, SFO to Los Angeles

A9: and then connecting to SAS flight nine thirty two?

C10: nine thirty two that's right

A11: actually the eleven seventeen I'm showing sold out

Consider dialogue excerpt 2, which is a dialogue between a travel agent (A) and a customer (C) who are constructing a travel plan for another traveler. Dialogue excerpt 2 is an example of an over-constrained problem where no solution can be found to satisfy the traveler's travel needs. The over-constraining constraint - requesting the flight to be United 1117 - occurred in turn C2, but was not detected until turn A11. Instead, the travel agent focuses on collecting more constraints to complete the information need specification only to find during the solution presentation phase that the problem is over-constrained.

Another limitation of existing research on mixed-initiative interaction and cooperative response generation in dialogue systems is that although various knowledge sources have been exploited to support generation of cooperative initiative-taking dialogue actions, e.g., domain models (Abella et al., 1996; Denecke and Waibel, 1997; Litman et al., 1998; Chu-Carroll, 2000), belief models (Chu-Carroll and Carberry, 1995a; Chu-Carroll and Carberry, 1995b; Green and Carberry, 1999), missing axioms (Smith, 1992), user models (Guinn, 1995), problem-solving states (Jordan and Di Eugenio, 1997), there has been little empirical work demonstrating how different initiative-taking strategies lead to more efficient, natural, or successful dialogues. Little evaluation has been done to compare the effectiveness and efficiency of different knowledge sources in supporting these strategies. In addition, the possible interactions between the clarification strategies in under-constrained situations and conflict resolution strategies in over-constrained situations have never been explored. Thus it is unclear under what conditions the use of knowledge sources result in more efficient or effective dialogues.

1.3. Thesis Statement

We claim that for modeling information systems dialogues in which both the system and the user have incomplete information of each other, a different model (other than the two-phase model) is necessary. In such a model, problem definition and solution presentation are interleaved and

incremental. In our work, we implement such a model using various constraint-based techniques such as constraint satisfaction, solution synthesis and constraint hierarchy. Using the constraint-based model, both the system's task specific knowledge base and the user's information needs are modeled as a network of constraints. There can be an arbitrary number of strengths reflecting varying degrees of preferences. Constraints and their strengths constitute a *constraint hierarchy*. *Solution synthesis* is a technique through which all solutions to a constraint satisfaction problem (CSP) can be computed by iteratively combining partial answers to arrive at a complete list of all correct answers. The solution synthesis technique supports easy computation and maintenance of partial parallel solutions, straightforward comparison between partial solutions, and easy incorporation of constraint modifications.

The framework of incremental problem formulation and solution construction and refinement consists of *cycles of constraint acquisition, solution construction, solution evaluation, and solution modification*. The stage of constraint acquisition relies on interaction with the user, thus requiring dialogue management. The stages of solution construction, solution evaluation and solution modification are conducted by a constraint-based problem solver. Through solution evaluation, solution status (over-constrained or under-constrained situations) can be evaluated immediately. Through solution modification, the system can use its knowledge about the problem solving state to guide the adoption of cooperative strategies for the acquisition of new constraints for under-constrained situations, or the modification of existing constraints for over-constrained situations. Repeating the cycles allows the system to improve its picture of the user's problem until a satisfying solution is found, through incremental acquisition of constraints to update the problem definition and incremental construction of partial solutions. The model is capable of providing solutions while the problem is still being defined.

We claim also that this constraint-based framework of incremental problem formulation and solution construction enables investigation of the effectiveness and efficiency of different heuristics for generating initiative-taking actions in both under-constrained and over-constrained situations and enables exploration of the interaction between these heuristics in both types of dialogue situations.

To summarize, we claim that:

- A constraint-based framework that models incremental problem formulation and solution construction supports the generation of initiative-taking behaviors which lead to efficient and cooperative human-computer dialogues;
- The constraint-based framework allows the exploration of principled ways for resolving over-constrained situations and under-constrained situations in information-seeking dialogues.

1.4. Thesis Contributions

The main contributions of this thesis can be summarized as follows:

- Developing a constraint-based framework of problem solving that supports incremental problem formulation and solution construction;
- Integrating and extending existing AI techniques such as solution synthesis, constraint hierarchy and constraint satisfaction in the constraint-based problem solver;
- Investigating, within the constraint-based framework, heuristics for selecting questions to ask the user in under-constrained situations and heuristics for identifying relaxation candidates in over-constrained situations;
- Evaluating the effectiveness and efficiency of the different heuristics in simulated information-seeking tasks. The performance of these different heuristics is analyzed with respect to several problem dimensions: e.g., task difficulty, interval size selection for interval attributes, and task complexity;
- Investigating the interaction between the question selection heuristics and the relaxation candidate selection heuristics;
- Evaluating the usability of initiative-taking actions supported by the constraint-based framework in a form-based information dialogue system.

1.5. Overview of the Thesis

The remainder of this thesis is organized as follows:

Chapter 2 reviews relevant work on mixed-initiative dialogues and cooperative response generation.

Chapter 3 reviews relevant work on constraint satisfaction that serves as the foundation of the proposed constraint-based model for supporting the generation of initiative-taking dialogue behaviors.

Chapter 4 elaborates on the constraint-based model to support the generation of initiative-taking actions. We describe our extensions to the traditional solution synthesis techniques to dynamic CSPs and the incorporation of constraint hierarchy into the model. We describe how relaxation is carried out within such a model and enumerate the features of problems for which the proposed framework is beneficial.

In Chapter 5, we present several heuristics that support question selection in under-constrained problems and constraint relaxation in over-constrained problems. These heuristics explore the usage of the knowledge sources of partial solution states and constraint hierarchy in the constraint-based model.

In Chapter 6, we describe the methodology for investigating the effectiveness and efficiency of the proposed heuristics in resolving the under-constrained and over-constrained situations in information-seeking tasks. We present the test collections for simulating information-seeking dialogues in the flight reservation domain and discuss the parameter settings in the evaluation. The statistics for measuring dialogue efficiency and task success are described.

Chapter 7 presents the evaluation results of the effectiveness and efficiency of the heuristics for solving simulated flight reservation tasks. We examine the effect of several factors for the evaluation, including task difficulty level, goal-state size, interval size, and task complexity.

In Chapter 8, we present COMIX, a form-based collaborative mixed-initiative information dialogue system that incorporates the constraint-based problem-solving model and the best heuristics we have obtained from the simulated evaluations. We first present the system architecture, which includes a dialogue manager, a constraint-based problem solver, task knowledge objects, and a form-based interface for the user, and describe the interaction between the system components. For the dialogue manager, we describe the principles for dialogue management, enumerate the initiative-taking dialogue actions motivated by an analysis of naturally occurring information-seeking dialogues, and describe the realization of these dialogue actions in a form-based dialogue environment.

Chapter 9 describes a limited-scope usability study of the COMIX system in different initiative settings – user-initiative vs. mixed-initiative. We present the design of the usability study and analyze the evaluation results.

Finally, Chapter 10 presents our conclusions and outlines several possible future directions for expanding this work.

Chapter 2 Related Work on Mixed-Initiative Interaction

In information-seeking and task-executing domains, since the information and abilities needed to solve a task are distributed among the participating agents, a system that is collaborating with users to solve a problem must have the flexibility to take over or give up initiative. Mixed-initiative interactions allow direction and control of the interaction shifts among the participants. A cooperative mixed-initiative system needs to model (1) *when* an agent takes or relinquish initiative and (2) *what* appropriate dialogue actions to perform when it takes initiative. In section 2.1, we review related work on the *when* aspect, focusing on initiative tracking. In section 2.2, we review related work on the *what* aspect, focusing on cooperative response generation in mixed-initiative interaction.

2.1. Initiative Tracking in Mixed-Initiative Dialogues

In naturally occurring mixed-initiative dialogues, initiative of interaction shifts among participants in a natural and coherent fashion. Recently, researchers argue for a two-thread model of initiative, which distinguishes whether the shifts occur at the discourse level or at the task level (Chu-Carroll and Brown, 1997a; Jordan and Di Eugenio, 1997). *Task initiative* tracks the lead in the development of the participants' domain plan in the problem solving process, while *dialogue initiative* tracks the lead in determining the current discourse focus. An agent has task initiative if his utterance directly contributes to the progress of the problem solving process. An agent has dialogue initiative if he takes conversational lead in order to establish mutual beliefs, such as resolving ambiguity or squaring away invalid beliefs. The dialogue initiative is subordinate to the task initiative: when an agent takes the task initiative, he also takes the dialogue initiative, but when the agent takes dialogue initiative, however, the agent does not necessarily take the task initiative. In fact, the agent may choose to pass the task initiative due to reasons such as lack of expertise in domain reasoning.

Chu-Carroll and Brown (1997a) further identify three classes of cues for tracking initiative in naturally occurring dialogues. The classification is based on the types of knowledge that are needed to recognize them. The first cue class, *explicit cues*, includes cues that can be recognized using linguistic information, i.e., explicit requests by the speaker to give up or take over the initiative. The second cue class, *discourse cues*, includes cues that can be recognized using linguistic and discourse information, such as the surface form of an utterance (e.g., questions) or the discourse

relationship between the current and prior utterances (e.g., repetitions). The third cue class, *analytical cues*, includes cues that can be recognized by performing an evaluation on the speaker's proposal using the hearer's private knowledge. For initiative tracking, Chu-Carroll and Brown (1997b) uses the Dempster-Shafer theory of evidence to predict the initiative holders in the next dialogue turn based on the current initiative holders and the effect that the observed cues have on changing them. The algorithm is demonstrated to be effective in predicting both dialogue initiative and task initiative.

In contrast to the explicit modeling of initiative proposed by Chu-Carroll and Brown (1997b), Heeman and Strayer (2001) stipulate that initiative is subordinate to the intentional structure of the discourse theory of Grosz and Sidner (1986). Following their argument, a dialogue manager does not need to have an explicit model of initiative. A dialogue manager only needs to model the intentional structure of discourse, and the initiative model follows, as initiative is held by the initiator of a discourse segment of the intentional structure.

In this work, we follow the distinction of task initiative and dialogue initiative proposed by Chu-Carroll and Brown, focusing on task initiative in the problem solving process. For initiative tracking, we have adopted a simple fixed initiative assignment policy. For example, in the usability study in Chapter 9, we allow the system the possibility to take dialogue initiative at every dialogue turn in the user-initiative setting, and allow the system the possibility to take both dialogue initiative and task initiative at every dialogue turn in the mixed-initiative setting. The more sophisticated models for initiative tracking by Chu-Carroll and Brown (1997b) and Heeman and Strayer (2001) could be incorporated into such a system in order to take into account dialogue context or the structure of the discourse.

2.2. Cooperative Response Generation in Mixed-Initiative Interaction

There has been much work on designing systems that take initiative in providing cooperative responses, following Cooperative Conversational Principles proposed by Grice (1975). Example earlier work in this area includes (Kaplan, 1979) and (Di Eugenio, 1987). Both have developed mechanisms that could identify query failures, and provide indirect answers to inform the user of the reasons for the failures.

Kaplan (1979) presents a system that provides cooperative responses to questions for database retrieval. His system can provide the user with cooperative indirect answers instead of the un-

cooperative “none” when a query fails to bring back any solutions. To achieve this, the system maintains a connected graph structure where the initial query is decomposed into sub-graphs. Whenever a query fails, the system goes back to check on the non-emptiness of these sub-graphs, and identifies the sub-graphs that causes no solution. A cooperative response is then generated to inform the user of the cause instead of just responding with “none found”. When no answers are available, the system can also make suggestive indirect responses that supply an answer to a slightly modified question. The modification relies on varying or eliminating the focus of the original question, with focus representing the aspect of the question that is most likely to shift in a follow-up question. The correct determination of the focus, however, is a not an easy problem. Once the right focus is chosen, it is generally more useful to vary the focus in a meaningful way than to simply eliminate the focus which can result in uncooperative responses.

Di Eugenio (1987) studies cooperative response generation in information retrieval from databases that addresses several issues: detection and correction of user’s misconceptions and answer generation that take into account the user’s expectations on the cardinality of the result. She maintains a graph that represents the structure of a query that follows the logical schemes in a relational database. When a query failure is found, the graph is traversed and all presuppositions that cause retrieval failure are identified. Her system also generates summary responses when the list of items in an answer is long. The summary response is based on the computation of the cardinality of the retrieval result. By providing the user with a summary response, the system gives the user the option to continue requesting details without overwhelming him with details in the first place.

The above two database query systems processed queries in isolation (i.e., one turn at a time). The structure that organized queries over several turns was not of much concern. In practice, however, user's information needs are seldom specified and satisfied within a single turn. Work has been done to incorporate discourse processing and dialogue management into design of more cooperative and user-friendly database interfaces. Within such a framework, queries are interpreted and responses are generated with respect to the previous discourse. For example, discourse phenomena that occur frequently in task-oriented man-machine dialogues such as anaphora and ellipsis can be complemented using discourse context (e.g., Grosz, 1977; Carbonell, 1983; Carberry, 1990). This enables users to ask brief, fragmentary, and under-specified questions, which has been identified to be preferred by users (Carbonell, 1983). The dialogue models can be driven by some pre-defined information templates, which are collections of properties that describe objects or transactions following a domain-dependent structure for executing a particular task.

These pre-defined properties can be used as the basis for the system to formulate queries to help the user accomplish the task (e.g., Abella et al., 1996; Denecke and Waibel 1997; Litman et al., 1998; Chu-Carroll, 2000). Or the dialogue manager can adopt a more general and sophisticated plan inference model, with plan inference at possibly separated multi-levels – discourse, problem-solving, and domain levels (e.g., Allen and Perrault, 1980; Chu-Carroll and Carberry, 1995a; Chu-Carroll and Carberry, 1995b; Green and Carberry, 1999; Smith, 1992; Guinn, 1995; Raskutti and Zukerman, 1997).

Abella and colleagues (1996; 1999) adopt an object-oriented approach to dialogue management. A major component of the dialogue manager is the task knowledge representation, which consists of application-dependent objects organized in an inheritance hierarchy. Each object has a name, a list of variables (or properties) with associated values that are specific to the object, and methods associated with the object. The dialogue manager exploits this hierarchy to determine what queries to ask the user. In (Abella et al., 1996), a property in each object is assigned a weight to convey the importance of the property. This weight defines the global measure of the importance of the property. In addition, properties are also assigned a state: either static (defined as part of the application domain) or dynamic (based on the current context of the dialogue). Combination of the properties and states give rise to several property categories. The membership of a property in a certain category categorizes the local importance of the property. The final weight of a property is the sum of the weight associated with the property and the weight associated with a category to which the property belongs.

In ambiguous situations in which there are too many responses from the application, the dialogue manager will select a question to ask the user in the hope that an answer to that question will resolve the ambiguity. Which property to ask in a question is computed based on the minimum expected number of questions (the computation of which will be discussed in more details in section 5.1.2). This computation takes into account the branching factor of solutions, property weights, and solution partitions by the domain values of properties. In situations where no solutions are found, the system begins by dropping the constraints with low weights. If solution tuples result from this relaxation, then the results are presented to the user. If the system runs out of properties it can drop, then the system resorts to confirming the constraints with high weights.

Denecke and Waibel (1997) propose using under-specified representations to represent relevant domain-dependent data that serve as the basis for generating clarification questions. The under-specified feature structures are a generalization of typed feature structures (Carpenter, 1992)

capable of representing descriptions of more than one object. The goal of a clarification question is to obtain information to disambiguate a representation. Asking a clarification question is tantamount to eliciting one option out of a list of several possible options. The user's response to the question provides information for disambiguating the under-specified representations. This model is information driven in that the information needed for disambiguating an under-specified feature structure is determined by the under-specified representation itself. The system chooses a feature path in the under-specified representation such that its value is not yet specified.

When multiple feature paths are available, the system chooses the path with the maximum entropy, with the entropy of a feature path defined as the information that is necessary to disambiguate the path. To calculate the entropy of a path, the system assumes that the typed hierarchy stores probabilities between typed objects expressing evidence of the subsumption relationship. When there is more than one path with the maximum entropy, the system selects the one with the shortest path.

Allen and Perrault (1980) were among the first to propose a plan-based model for generating cooperative responses and interpreting indirect speech acts. Recognizing that dialogue participants are autonomous and that negotiation and conflict resolution are inevitable, they model domain plans and problem solving plans for negotiation and conflict resolution.

Chu-Carroll and Carberry (1995a; 1995b) present a plan-based framework for collaborative problem solving by a *Propose-Evaluate-Modify* cycle of collaboration. Their theory views the collaborative planning process as a sequence of proposals, evaluations, and modifications, which may result in a constructed plan fully agreed upon by both participants. Through the evaluation process, infeasible actions, ill-formed plans, sub-optimal plans, and information sharing needs can be identified. Negotiation, clarification, and information-sharing dialogues can be subsequently initiated to correct an invalid proposal, to suggest better alternatives, and to provide supporting evidence to establish mutual beliefs. Within this framework, Chu-Carroll and Carberry focus on modeling information-sharing sub-dialogues to resolve uncertainty in beliefs and collaborative negotiation sub-dialogues to resolve conflicts in beliefs. Information-sharing sub-dialogues are initiated by an agent when she has to evaluate the proposal made by the other agent, but she realizes that she does not have sufficient information to do so. Collaborative negotiation sub-dialogues are initiated when an agent detects a conflict between the agents' beliefs that would cause her to reject it. Their model relies on detailed and complete system and user models with ranking heuristics for detecting and resolving conflicts and ambiguities.

Green and Carberry (1999) present an approach within a plan inference framework for a system to take the initiative in providing unrequested extra relevant information to Yes-No questions. In the model, stimulus conditions are introduced into the discourse plan operators to model a speaker's motivation for providing extra information. Stimulus conditions, based on linguistic analysis, can be considered as pre-compiled results of *deep* planning based on discourse goals; the use of these conditions reduces the reasoning required for content determination.

Smith (1992) is interested in deciding when an agent directs initiative to another agent in a collaborative problem-solving domain for effective collaboration. He proposes the *missing axiom theory of discourse*; that is, the role of generating an utterance in a problem-solving task is to supply "missing axioms". Smith uses a top-down problem-solver for collaborative problem solving. The problem-solver can check whether a goal has been satisfied, decompose the goal into subgoals and solve the subgoals. When a particular goal cannot be satisfied through the above means, one option of the system is to request another agent to satisfy the goal. Utterances are generated only when needed, i.e., when the problem solver cannot solve the goal using its own knowledge.

Guinn (1995) extends Smith's problem solver and presents a model of collaborative discourse that integrates problem solving, dialogue initiative, conflict resolution and negotiation. He studies how these dialogue phenomena and knowledge requirements contribute to the effectiveness of collaboration. Simulations have been conducted to demonstrate the effectiveness of discourse initiative in collaborative problem solving. The model attaches an initiative level to each task level. A competency evaluation, based on user model information, is used to decide who should be given the initiative for a given task goal.

Raskutti and Zukerman (1997) develop a system that generates disambiguating and information-seeking queries during collaborative planning activities. In situations where their system infers more than one plausible goal from the user's utterances, it generates disambiguating queries to identify the user's intended goal. In cases where a single goal is recognized, but contains insufficient details for the system to construct a plan to achieve this goal, their system generates information-seeking queries to elicit additional information from the user in order to further constrain the user's goal. The system makes use of a hierarchical representation of goals possibly intended by the user, and probabilities of these goals for generating the disambiguation queries and information-seeking queries.

Recently, some researchers have used constraint-based framework for modeling cooperative human-computer interaction. For example, in analyzing collaborative problem solving dialogues, Jordan and Di Eugenio (1997) model problem solving of a collaborative task as a constraint satisfaction problem (CSP). The problem space consists of variables that must have a single value or a set of values of certain cardinality assigned to them. In their recent work on the agreement process in collaborative dialogues, Di Eugenio et al. (2000) have again modeled problem solving space as a set of constraints and applied problem solving features as one dimension affecting negotiation. The solution size can be characterizes as determinate, when there exist solutions that satisfy a set of constraints, or indeterminate, when there exists no solution or when the constraints of some parameters are left open. Focusing on empirical analysis of human-human interaction, neither work has presented a detailed computational model of incorporating problem-solving features for generating cooperative human-computer interaction.

Donaldson and Cohen (1997) propose a three-part goal-oriented model of turn taking. The three parts of the model are *why* to take a turn (motivation), *how* to take a turn (which goal to address in the dialogue), and *when* to take a turn (at relevant points in the dialogue). In particular, in their model, each utterance that a speaker makes will cause the listener to adopt multiple potential turn-taking goals; the listener then must choose the most appropriate goal to pursue. A constraint satisfaction based framework is used to decide which goal to pursue. Specifically, turn-taking goals are represented in a turn-taking CSP by variables $V_1,...,V_n$, with each variable representing one turn-taking goal. The CSP solver examines the constraints between the variables, and attempts goal assignments that break as few constraints as possible. Constraints on the variables are mostly ordering constraints[2]. The algorithms proposed for solving turn-taking CSPs are *heuristic repair* methods. To use the heuristic repair methods, one needs first to have a scoring mechanism to measure the quality of the solutions. The methods start with an instantiated solution, either randomly or through some procedure, then (1) select a variable that participates in a conflict, and (2) assign a value to the variable that improve the quality score of the solution. Such methods are appropriate for solving turn-taking CSPs, since it provides an anytime algorithm. Given that an agent may be required to speak at any time, this feature is important.

Freuder and Wallace (1997) introduce a model of constraint acquisition and satisfaction for Customer and Matchmaker interaction. The model consists of two modes: *suggestion*, made by the

---

[2] One example constraint from (Donaldson and Cohen, 1997) is that clarification goals should be accomplished before answering goals. The rationale for this constraint is that ambiguity should be resolved before an answer is given.

Matchmaker to the Customer and, *correction*, made by the Customer to the Matchmaker, indicating how the suggestion fails to meets the customer's needs. Repeating the cycles of suggestion and correction allows the Matchmaker to improve its picture of the Customer's problem until a final suggestion constitutes a satisfactory solution. The goal of the CSP solver is to find suggestions that reduce the length of the dialogue between the Customer and the Matchmaker. The first proposed approach is to find solutions that are most likely to satisfy constraints between variables, even for constraints that have not yet been presented to the Matchmaker. To achieve this, domain values that are less likely to be in conflict with values in other variables are preferred. The second approach is to maximize constraint violations, in the hope that the Matchmaker can find the solutions that violate as many constraints as possible and gather these constraints early on as part of the problem formulation. To achieve this, values that are mostly likely to be in conflict with values in other variables are preferred. During the iterations, solutions from the previous iteration can be re-used by resetting values to certain variables.

While cooperative response generation has been the focus of considerable research in cooperative human-computer interaction, there is little empirical work that compares different response generation strategies and demonstrates how some strategies lead to more efficient and successful dialogues than others. A notable exception is the recent work by Litman et al. (1998). Litman et al. empirically evaluated response strategies within a system called TOOT that allows users to access online AMTRAK train schedules via a telephone dialogue. TOOT has two versions distinguished by different response strategies: literal TOOT (LT) and cooperative TOOT (CT).

In under-constrained situations, i.e., where there are too many trains to present in a single utterance, the LT strategy groups the information into units of 3 trains, then the system presents each unit until the user tells the system to stop. In contrast, the CT strategy first summarizes the ranges of trains available, then gives the user the option to hear the trains in units of 3 or to further constrain the query. In over-constrained situations, i.e., where there is no train that matches a query, the LT strategy only reports the failure of retrieving trains for the query, while the CT strategy first relaxes the user’s time constraint and then prompts the user to perform other types of relaxation. Their empirical evaluation, based on hypothesis testing and PARADISE (Walker et al., 1997), examines how evaluation measures differ as a function of response strategy and task and elaborates on the conditions under which TOOT response strategies lead to better performance.

This thesis addresses two limitations of the existing research work on cooperative response generation in information-seeking dialogue applications. First, among the work on database query

systems (e.g., Abella et al., 1996; Denecke and Waibel, 1997; Litman et al., 1998; Chu-Carroll, 2000), the systems often adopt application-dependent templates as the guide for formulating queries. In the process of query formulation, the systems try to elicit as much information about the user's information need as possible to fill in the templates, without much consideration of the computation of solutions. Once the templates are filled in, the systems then send the queries to the database to retrieve solutions that satisfy the user's information needs. As a result, the query formulation stage and the solution computation stage are separated. We have observed through the human-human information-seeking dialogues in Chapter 1 that such a dichotomy between query formulation and solution computation can result in interaction problems such as delayed detection of over-constrained situations. The first focus of this thesis, therefore, is to propose a computational model of *incremental problem formulation and solution construction that interleaves the two aspects of problem solving*. This model supports immediate detection of under-constrained and over-constrained situations even during the problem formulation stage. The model is based on a combination of constraint processing techniques (e.g., solution synthesis, constraint hierarchy, walkabout strengths) that have not been explored much in information-seeking dialogue applications.

Second, although various knowledge sources (e.g., domain models, belief models, missing axioms, user models, problem-solving states) have been exploited to support generation of cooperative initiative-taking dialogue actions, and although these knowledge sources have been demonstrated to be useful in generating clarification or conflict resolution sub-dialogues, most systems adopt a single strategy for generating these sub-dialogues. The effectiveness of strategies based on different knowledge sources is seldom compared. In addition, the possible interaction between the clarification strategies and conflict resolution strategies has never been explored. The constraint-based model we propose allows us to explore these strategies and the interaction between them in a single uniform framework. The second focus of this work, therefore, is (1) to investigate the effectiveness and efficiency in employing different knowledge sources in generating initiative-taking actions in both under-constrained and over-constrained situations, (2) to explore the interaction between the actions taken in both types of dialogue situations, and (3) to provide an empirical evaluation of the different initiative-taking strategies in human-computer information-seeking systems.

Chapter 3 Constraint Processing Techniques

A *constraint* describes a relation that should be satisfied. Constraints have been used in a variety of research areas in AI, e.g., geometric layout, user interface support, general-purpose programming languages, planning, scheduling, computer vision and natural language processing. In this chapter, I review the definitions and several techniques for constraint processing: constraint hierarchy, walkabout strength, solution synthesis, and variable or value ordering. Some of the definitions presented in this chapter are adapted from Tsang (1993). I discuss how to extend and combine these techniques for incremental construction of partial solutions for cooperative problem solving in the next chapter.

3.1. Definitions

Definition 3-1:

> **CSP**: A constraint satisfaction problem (CSP) is typically specified by a set of variables $V = \{v_1,..., v_n\}$ and a set of constraints C on subsets of V limiting the values that may be assigned in a consistent manner. Each variable v_i has an associated domain $D_{v_i} = \{d_{v_i 1},..., d_{v_i n}\}$, which identifies its set of possible values.

The constraint satisfaction task is to find assignments of values for v_i in V that simultaneously satisfy all the constraints C.

Constraints can be categorized based on the number of variables that participate in them. *Unary constraints* specify the possible values that a variable can take without reference to other variables. For example, A>3 is a unary constraint that says the values for the variable A need to be greater than 3. *Binary constraints* specify the relationship between two variables. For example, the binary constraints A>B says that any value assigned to variable A needs to be greater than the value assigned to B. *N-nary constraints* involve the relationship between *n* variables. Without loss of generality, in this thesis, we restrict most of our discussion to unary and binary constraints. In this thesis, constraints can be represented as relations between variables (e.g., $A<B$), can be represented as solution tuples (Definition 3-4), or can be represented by the letter C with the variables as subscripts (e.g., C_A is a unary constraint for A and C_{AB} is a binary constraint between A and B).

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 constitute a user's information need as the variables in a CSP, and the user's preferences and restrictions over these attributes as constraints. Typically, a *information request* (or a *query*) is a request for information for a set of target attributes based on some value specification for given attributes and between attributes. For instance, in the travel domain, attributes such as the arrival city, the departure city, the departure date, and the carrier can be treated as variables. The domains for these variables are the possible values found in the database (e.g., US Airways, United Airlines, etc., for the carrier attribute). The variables are constrained by domain relations in the domain databases and the user preferences and restrictions on these variables.

Definition 3-2:

> **Dynamic CSP (DCSP):** In CSPs, the sets *V*, D_{v_i} , and *C* are *fixed* and known beforehand. Each solution must contain one or more sets of assignments for every variable in *V* and every constraint in *C*. In many problems, the set of variables and the set of constraints can change during the problem solving process. Such problems are modeled as dynamic constraint satisfaction problems.

A DCSP 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 (in which case it is considered a *restriction*) or increase (i.e., a *relaxation*).

In human-computer interaction of information-seeking tasks, the set of variables that are relevant to a solution and the values that can be assigned to them change dynamically in response to user input and the decisions made during the course of the problem solving process. Therefore, constraint-based information CSP can be readily modeled as a DCSP.

Definition 3-3:

> **Compound labels**: A compound label is the simultaneous assignment of values to a (possibly empty) set of variables $\{v_1,...v_n\}$. We use $\{(d_1,...,d_n)\}$ to denote the compound label of assigning $d_1,...,d_n$ to $v_1,...v_n$, respectively.

Definition 3-4:

Solution tuples: A solution tuple of a CSP is a compound label for all the variables in the CSP that satisfies all the constraints.

We define a CSP as *satisfiable* if there exist solution tuples for the CSP. In some applications, we are interested in obtaining a set of possible solutions of certain size *k*. For instance, in an information-seeking application through a form-based interface, we may want the system to provide a fixed-sized set of possible solutions for the user to browse from. Depending on the size of the solution tuples, we can classify CSPs into the following categories with respect to the pre-determined number *k*:

Under-constrained: the size of the set of possible solution tuples exceeds *k*

Over-constrained: no solution tuples are possible

Under-constrained situations occur due to lack of enough constraints to constrain the problem. To solve under-constrained problems, new constraints are generally required to further restrict the solution space.

Over-constrained situations are the result of failure to satisfy some individual constraints or failure to satisfy combinations of some constraints. In order to get to solutions for over-constrained problems, enlarging or removing constraints is generally explored. For example, for the over-constrained information need "*Does American Airlines have a night flight to Dallas?*" discussed in section 1.1, we can expand the constraint domain for airline being AA to include other airlines. Removing the constraint airline being AA is tantamount to expanding the constraint domain to include all possible airlines in the flight information-seeking task.

In some applications, some solutions are better than others. In other cases, the assignment of different values to the same variable incurs different costs. The task in such problems is to find optimal solutions, where optimality is defined in terms of some application-specific functions. For example, in many scheduling applications, we may be interested in getting materials from one location to another in the shortest route possible (e.g., getting materials from New York to Boston directly instead of routing through Denver) or in maximizing the utility of the cargo vehicles (e.g., having the cargo vehicles full thus with higher utility than having the cargo vehicle empty on certain trips). In seeking flight information, we may want to find the cheapest price for a trip or to minimize the total travel time. These problems are called Constraint Satisfaction Optimization

Problem (CSOP) to be distinguished from the standard CSP. A CSOP is defined as the standard constraint satisfaction problem (CSP) plus the requirement of finding optimal solutions.

Definition 3-5

> **CSOP:** A CSOP is defined as a CSP (Definition 3-1) together with an optimization function f that maps every solution tuple Γ in the set of solution tuples S to a numerical value:
>
> $f: S \rightarrow$ numerical value.
>
> We call $f(\Gamma)$ the f-value of Γ.

The task of a CSOP is to find the solution tuple with the optimal (minimal or maximal) f-value with respect to the function f. The optimization function f is generally application-dependent.

To find the optimal solutions, we need to find all possible solution tuples first, and then compare their f-values. Therefore, techniques that compute all possible solutions (such as solution synthesis to be discussed in section 3.3) are more relevant for such problems than techniques for finding single solutions. Although pruning of search space can potentially reduce computation complexity, one has to ensure that the search space that is being pruned does not contain the optimal solution. This requires knowledge about solutions and f-values to guarantee that no solution exists in the pruned search space or that the f-values to the solutions in the pruned search space are sub-optimal.

There are often a large number of ways to satisfy the constraints in CSPs. Heuristics are often used to arrive at desired solutions. A more predictable and declarative way to control the solution process is using constraints of various strengths to guide the process of selecting a solution (Borning et al., 1996; Maloney 1991). The *strength* of a constraint specifies the extent that the constraint is preferred to be satisfied. A *required* constraint must be satisfied, while a *preferential* constraint is not required to be satisfied, but is preferred to be satisfied if possible. Since preferential constraints are not required, they can be overridden by stronger constraints when necessary. There can be an arbitrary number of strengths reflecting varying degrees of preferences. Constraints and their strengths constitute a *constraint hierarchy.*

We use *labeled constraints*, constraints that are labeled with their respective strengths, to represent constraints in a constraint hierarchy. For better readability, in this thesis, we give symbolic names to the different strengths of constraints (e.g., required, strong, weak, etc.), even though in implementation the strengths are mapped into numerical values. For example, we represent the

required constraint that the arrival city be Dallas as the labeled constraint ArriveCity=Dallas, required.

Definition 3-6:

> **Constraint hierarchy**: A constraint hierarchy consists of labeled constraints that are organized into different levels based on their strengths.

We expect the set of solutions to a constraint hierarchy to demonstrate the following characteristics: first, all the required constraints should hold. Second, the solutions should satisfy the non-required constraints as well as possible, respecting their relative strengths.

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 to be a certain carrier (e.g., USAir) is *preferred* but not required. The task of the system is to provide users with information satisfying their information needs as much as possible. A constraint hierarchy can be exploited in various ways to achieve such a goal. First, the preferential information encoded in a hierarchy can be used to guide constraint or variable ordering in constructing solutions. Second, the constraint hierarchy controls how a solution graph should be updated in order to accommodate any possible changes to its original CSP. For example, when a user request is under-constrained, the system may take the initiative to ask the user for clarifying questions to obtain more of the user's preferences. Which particular question to ask could be based on the strengths of the corresponding constraints participating in the problem. A potential useful heuristic is to select the constraints with higher strengths first. When the user's request is over-constrained, or when the user's later preferences are in conflict with the previous ones, the system may suggest constraints for relaxing the over-constrained request by relaxing the constraints with the weakest strengths. The constraint strengths used for describing examples in this thesis include required, strong prefer, medium prefer, and weak prefer.

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. An example domain constraint in the travel domain is that the time between two connecting flights should be greater than 30 minutes. Domain constraints generally have required strengths as well. *User constraints*

represent the restrictions and preferences in the user's information needs. For example, a user may prefer the cheapest flights, or a flight with a certain airline, or connections with shorter connecting times. 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.

3.2. Finding Solution Graphs

Constraints can be satisfied in multiple ways or *methods*. Methods are procedures that, if executed, will cause the constraint to be satisfied. Each method determines a value for one or more variables (*outputs*) from its other variables (*inputs*). For example, the plus constraint in $A + B = C$ can be satisfied by the following three methods: $A \leftarrow C - B$, $B \leftarrow C - A$, and $C \leftarrow A + B$. A set of multi-way constraints typically has many potential propagation paths. Thus the constraint problem solver must in general decide which path to use, and perhaps execute a method for each constraint. In the case of a constraint hierarchy solver, the selected path should compute a ``best" solution according to the constraint hierarchy. The propagation path can be represented as a directed graph, which we call the *solution graph*.

The problem of finding a solution graph for a constraint graph is in general NP-complete (Maloney 1991). Very efficient algorithms for solving restricted forms of constraint problems, however, have been proposed by researchers. Here I review a restricted form of constraint problems and an algorithm that can be used in finding solution graphs efficiently for such kind of problems.

The CSP problems that are relevant to this thesis observe two restrictions on a constraint graph: 1) that the constraint graph is free of cycles, and 2) that no constraint method has more than one output variable. This type of CSP problems can be solved efficiently by the DeltaBlue algorithm, introduced by Freeman-Benson and Maloney (Freeman-Benson and Maloney, 1989; Freeman-Benson and Maloney, 1990; Maloney 1991). The key idea to this algorithm is to only use information local to a constraint when deciding how to enforce it. DeltaBlue does this by annotating every variable with an incrementally maintained value called the *walkabout strength*.

3.2.1. Walkabout strengths

The *walkabout strength* of a variable *V* is defined in (Maloney 1991: p57) and is repeated here as follows:

If V is determined by method M of constraint C in the current solution graph, its walkabout strength is the weaker of C's strength and the weakest walkabout strength among all potential outputs[3] of C except V itself.

If V is not determined by any other constraint, then its walkabout strength is determined by its system-supplied stay constraint, so its walkabout strength is *weakest*.

We illustrate the computation of walkabout strengths using the CSP example in Figure 1. The figure shows a solution graph for a CSP with five variables A, B, C, D, and E, three stay constraints on A, B, and D, with strengths being $S_A = strong$, $S_B = required$, and $S_D = strong$, respectively, and two constraints C_1 and C_2 with strengths $S_{C_1} = required$ and $S_{C_2} = weak$, respectively. An arrow indicates the output of each constraint.

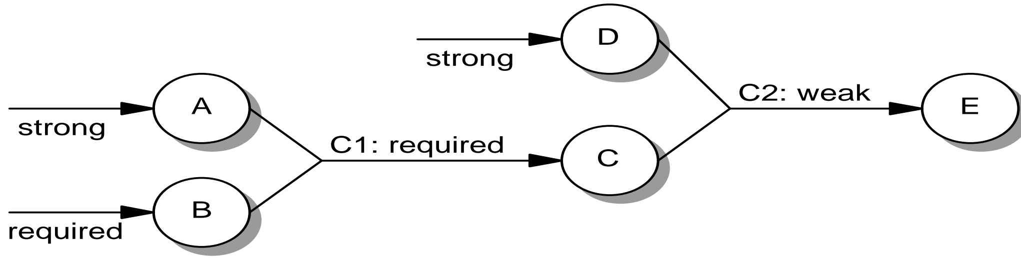


Figure 1: An example solution graph. An arrow indicates the output of each constraint.

The walkabout strengths (WS) of variables A, B, and D are determined by their stay constraints. The walkabout strength of variable C is the minimum of A's walkabout strength, B's walkabout strength, and C_1's strength. The walkabout strength of variable E is the minimum of C_2's strength, C's walkabout strength, and D's walkabout strength. Note that weak walkabout strengths propagate through stronger constraints (A to C), but strong walkabout strengths do not propagate through weaker ones (C to E). Specifically,

$$WS_A = S_A = strong$$

$$WS_B = S_B = required$$

$$WS_D = S_D = strong$$

$$WS_C = \min(S_{C_1}, WS_A, WS_B) = \{required, strong, required\} = strong$$

[3] A variable is a *potential output* of a constraint C if it is the output of any method of C.

$$WS_E = \min(S_{C_2}, WS_C, WS_D) = \{weak, strong, strong\} = weak$$

As summarized in (Maloney 1991), walkabout strengths have two useful characteristics: first, because weaker walkabout strengths propagate through stronger constraints in a solution graph, and stronger walkabout strengths do not propagate through such a graph, walkabout strength of a variable, thus, indicates the strength of the weakest constraint in the current solution graph. Such a constraint can be relaxed to allow other constraints to be enforced. Second, the walkabout strength of a variable may reflect the existence of a constraint quite far away in the solution graph. This allows the DeltaBlue algorithm to efficiently identify the constraint to relax without the need to traverse the graph.

3.2.2. DeltaBlue algorithm for finding solution graphs

The DeltaBlue algorithm maintains two data structures: the current constraint hierarchy *H* and the current solution graph *S*. Upon any change, the solution graph is incrementally modified by two operations: AddConstraint and RemoveConstraint. A detailed description of the DeltaBlue algorithm can be found in (Maloney, 1991: p57-63).

In general, when adding a constraint whose strength is stronger than the walkabout strength of its output variable *V* in the solution graph, the algorithm enforces the constraint and updates the walkabout strength of *V* and incrementally updates the walkabout strength of all other variables. When removing a constraint whose output variable is *V*, the algorithm sets the walkabout strength of *V* to weakest and incrementally updates the walkabout strengths of all other variables. For a constraint hierarchy with *N* existing constraints and *M* methods for enforcing each constraint, the DeltaBlue algorithm spends *O(M)* time deciding how to enforce a given constraint, and considers each constraint at most once during each incremental operation. In the best case, the constraint can be enforced and the cost is just *O(M)* to consider its *M* possible methods. In the worst case, every constraint in *H* is retracted and reconsidered at a cost of *O(M)* each, resulting in a total cost of *O(MN)*. An average case would require a combination of retracting constraints and re-computing walkabout strengths for part of the solution graph. Since *M*, the methods available to enforce a constraint, is usually bounded by a smaller number, the incremental running time for DeltaBlue is in effect *O(N)*, i.e., the cost of the actual running time grows linearly with the number of constraints in the constraint hierarchy.

3.3. Computing Solutions Through Solution Synthesis

Finding a solution graph constitutes the first stage for computing solutions for a set of constraints. Once a solution graph is found, variables in the solution graph need to be instantiated so that the values for these variables satisfy the constraints. Sometimes the goal is to find one such assignment, sometimes the goal is to find all such assignments. Since our interests concern finding all solutions during problem solving process, only techniques for generating all solutions are relevant. In this section, I review solution synthesis, a technique through which all solutions to a CSP can be computed.

Solution synthesis is a technique to generate *all* solutions to a CSP by iteratively combining partial answers to arrive at a complete list of all correct answers. In general, variables of a CSP are represented as nodes at the base level (order 1) in a solution synthesis graph. Nodes of higher levels are incrementally constructed, representing all legal compound tuples of the set of lower level variables, until the node for solution tuples is synthesized. Through solution synthesis, all assignments of values to variables that satisfy the problem's constraints are produced. Solution synthesis for CSP was first introduced by Freuder (1978). Freuder's original algorithm requires maintaining a lattice resulting from exhaustive combinations of lower level nodes into higher-level nodes. Upward and downward constraint propagation is used to eliminate illegitimate compound labels in the nodes. Freuder's algorithm is intractable, and is generally not suitable for generating practical solutions (Tsang and Foster, 1990). The solution synthesis technique was later refined in the Essex Algorithm by Tsang and Foster (1990) and recently in Hunter-Gatherer by Beale (1997).

3.3.1. The Essex algorithm

In the Essex algorithm, Tsang and Foster (1990) propose an arbitrary ordering of input variables at the base level. Second-order solutions are constructed only from *adjacent* input nodes, in contrast to the exhaustive combinations of nodes as found in Freuder's algorithm. Third-order solutions are constructed only from *adjacent* second-order solutions, etc. The root of the tree is the node of solution tuples. Figure 2 presents the solution synthesis tree for a CSP with four variables A, B, C and D, and three constraints $A<B$, $A=C$, $C>D$.

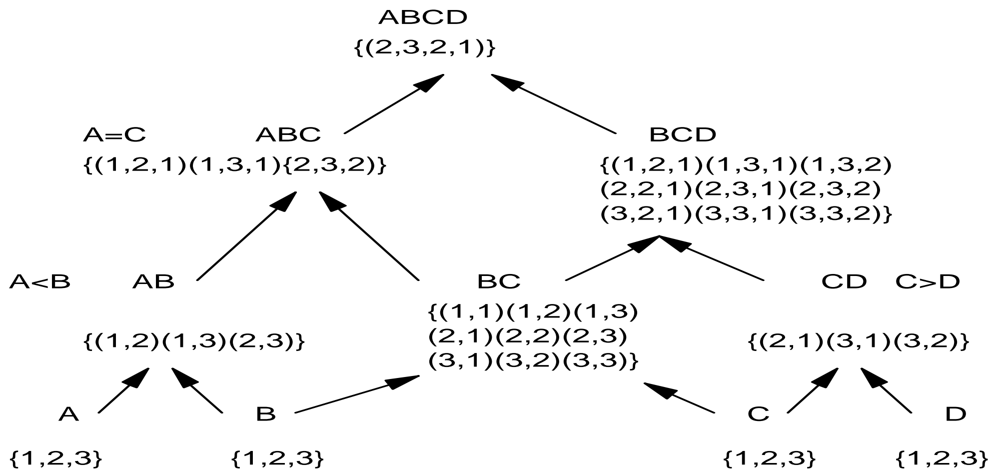


Figure 2: Solution synthesis using the Essex algorithm. The CSP has four variables A, B, C, and D, and three constraints A<B, A=C, and C>D. Adjacent nodes at each level are combined together for computing all partial solutions. The final assignment for the CSP is A=2, B=3, C=2, D=1.

The efficiency of the basic Essex algorithm can be improved (Tsang and Foster 1990). First, the efficiency can be improved by giving the base level nodes a specific ordering. In general, the smaller the size of the nodes is, the less computation is needed for constructing nodes of higher levels. Even though the sizes of nodes at the base level, which are determined by the problem specification, cannot be controlled, the sizes of the higher nodes could be reduced through grouping the lower-level nodes in a way so as to maximally constrain each other. Minimum bandwidth ordering (MBO, to be discussed in section 3.4) has been proposed at initialization to maximize the constraining effect.

Second, constraints can be partially propagated to maximize the effect of early constraint pruning. Doing so could reduce the size of nodes, thus make later synthesis less expensive.

3.3.2. Hunter-Gatherer

Beale (1997) advances the work on solution synthesis in several directions in the Hunter-Gatherer (HG) approach. First, Beale generalizes the idea of synthesis to synthesis of subgraphs, thus eliminating the limitations of previous work, in which synthesis happens only to pair of nodes at the same level. Solution synthesis in HG is used to combine results from subgraphs or to add individual nodes onto subgraphs.

Second, HG extends the idea of MBO ordering of variables (to be discussed in section 3.4), first proposed in the Essex algorithm, to subgraphs to maximize the benefits of early constraint pruning. In HG, solution synthesis is used to combine smaller sub-solutions into larger solutions. Variables

(and later subgraphs) are grouped in such a way as to maximally constrain each other so that the solution sizes can be reduced as early as possible, and to minimize the amount of interaction across subgraphs. Interactions outside the subgraphs being combined are not noticed. The search space is organized into maximally independent subgroups of any size using graph decomposition techniques.

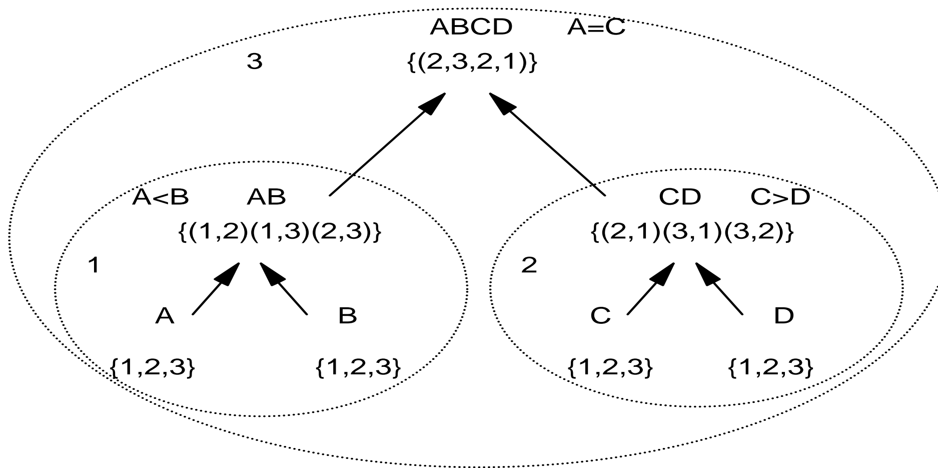


Figure 3: Solution synthesis using HG. The CSP has four variables A, B, C, and D, and three constraints A<B, A=C, and C>D. Closely-constrained variables or subgraphs are ordered in a way to constrain each other to reduce the sizes of higher-level nodes. The final assignment for the CSP is A=2, B=3, C=2, D=1.

The savings of HG are apparent from Figure 3, which represents a more efficient solution space as compared to the solution space in Figure 2 for the same CSP problem. In this figure, variables A and B are grouped together into subgraph 1 so that the constraint A<B can be effective in reducing the solution size at node AB. Variable C and D are grouped together into subgraph 2 so that the constraint C>D can be effective in reducing the solution size at node CD. Note the combination between variable B and C as found in Figure 2 is not necessary in HG, because there is no interaction between B and C.

This thesis extends the Hunter-Gatherer approach to dynamic constraint satisfaction problems with preferential information. In the next chapter, I describe how constraint hierarchy, walkabout strength, and solution synthesis can be combined together for dynamically constructing and maintaining parallel partial solutions during interaction.

3.4. Ordering Heuristics

Ordering heuristics are general principles that can be used to optimally order the instantiations of variables and their values. Many ordering heuristics have been proposed. These types of strategies are closely connected to constraint analysis; a good discussion of them can be found in (Tsang, 1993). Decisions about ordering could affect the pruning of search spaces and the number of backtracks required in a search, thus affecting the efficiency of the search process significantly.

The "Minimal Width Ordering" (MWO) heuristic aims to minimize the need for backtracking. It applies to CSP problems where some variables are more constrained by more variables than others. The general strategy is to assign values to the variables that are more constrained first and label the less constrained variables last, in the hope that backtracking will be reduced. Take as an example a simple CSP with three variables A, B, and C, and two binary constraints A=B and A=C. It would be beneficial to label A first. This will reduce the choices in B and C. If B and C are labeled first with different values before A, then when A is labeled, its value could conflict with one of the values in B or C, thus requiring backtracking. For MWO, the variables are ordered before the search starts.

The "Minimal Bandwidth Ordering" (MBO) heuristic seeks to reduce the distance that will have to be backtracked in case of failure. It achieves this by placing the constrained variables close together in the hope that when backtracking is necessary, only a small distance will have to be backtracked. For example, in a CSP with five variables A, B, C, D, and E, and a binary constraint between A and B, if A is labeled immediately after B, then in case of failure, backtracking would proceed immediately to B, without intervening variables. If after B is labeled, C, D, and E are labeled first before A, then when backtracking is necessary, the search will need to backtracking all the way back to B, and then re-label the values for C, D, and E. Obviously, the distance for backtracking is smaller in the former case than the latter. For MWO, the variables are also ordered before the search starts.

The fail first principle (FFP) heuristic aims at recognizing failure as soon as possible. It first selects the variable that is most difficult to assign a value, because it is most likely to fail. One way to measure the difficulty in assigning a value to a variable is using the size of the domain: the variable that has the smallest domain should be tried first. For example, if variable A has domain size 6 and variable B has domain size 4, then variable B should be instantiated first.

In contrast to the MWO and MBO heuristics that give the variables a fixed ordering before search starts, the ordering with FFP can be either static or dynamic. When FFP is used to order variables before search starts, it sorts the variables in ascending order based on their domain sizes. When the FFP is used together with lookahead algorithms, where constraints are propagated after a variable is labeled, values are possibly removed from the domains of unlabeled variables. Again, at each stage of the search, the domains of all the unlabeled variables are compared and the variable that has the smallest domain will be selected. Because the domain sizes of the unlabelled variables could change dynamically after constraints are applied, the ordering could change dynamically.

These ordering heuristics exploit various features of the constraint graph. The MWO and MBO consider the topology of the constraint graph, but do not take into account the domain of the variables. The FFP, on the other hand, considers the domain of the variables, but does not consider the topology of the constraint graph. The success or failure of instantiating one variable is independent of another variable's success or failure. The MWO and MBO heuristics can be combined with FFP. One approach is to employ FFP to order the variables dynamically, and when several variables having the same domain size, use the principles in the MWO or MBO heuristics to break the ties. Another approach is to employ the MWO or MBO heuristic to order the variables before labeling, and in case of ties, select the variable which has a smaller domain size. Which heuristics are more efficient and whether it is worthwhile to combine the heuristics together for a particular problem is obviously problem- or domain-dependent.

Efficiency of a search algorithm for solving CSPs can also be affected by the ordering of values selected for each variable. While variable ordering techniques select the most constrained variables first to reduce backtracking or to identify backtracking early so that if one needs to backtrack one does not first waste effort that will later thrown away, value ordering techniques seek to minimize backtracking by selecting the most promising values first. Value ordering is only useful for finding single solutions, however. For problems that require all possible solutions, heuristics for ordering of values are not helpful. All values that result in solutions must be attempted; regardless of the order in which they are found.

The effectiveness of variable and value ordering heuristics has been investigated in constraint-based work. For example, Sadeh and Fox (1995) investigate variable and value ordering heuristics in the job scheduling domain. Examples of job shop scheduling problems include factory scheduling problems, space mission scheduling problems, and factory rescheduling problems. Their evaluation experiments, designed to reduce the effective size of the search space, have shown

that generic variable and value heuristics such as MWO do not perform well in this domain. This is because the tightness of the constraints and the connectivity of the constraint graphs resulting from the interactions between resource demand and resource contention in the job scheduling domain cannot be captured properly by the general heuristics. They introduce a probabilistic framework to better capture the key aspects of the scheduling search space and demonstrate empirically that variable and value ordering heuristics derived within such a probabilistic framework often yield significant improvements in search efficiency and significant reductions in the search time required to obtain a satisfactory solution.

Therefore, it is worth remembering that heuristics such as MWO, MBO, and FFP are general strategies only. Given a particular application, domain specific features and knowledge should always be examined. Sometimes, the available domain-specific heuristics (e.g., a constraint hierarchy that specifies the importance of variables and constraints in a particular application or the measure of resource demand and contention in the job scheduling domain) may be very effective.

Chapter 4 Constraint-Based Model of Problem Solving

In interactive information seeking systems, a user's requests may be under-constrained or over-constrained. A cooperative system should have the capability of asking constraining questions during under-constrained situations or suggesting constraint relaxation candidates in over-constrained situations to facilitate cooperative problem solving. This requires the ability of the system to identify restriction and relaxation candidates to resolve under-constrained and over-constrained problems. In this thesis, we propose a framework of incremental solution construction and refinement, and propose using a solution network as the knowledge source for identifying restriction or relaxation candidates. In this chapter, we present the integration of the AI techniques of constraint hierarchy, walkabout strength, and solution synthesis for incremental solution construction and refinement.

In section 4.1, we discuss how to construct solutions dynamically and incrementally by extending the traditional solution synthesis techniques. The first extension extends the solution synthesis techniques to dynamic CSPs. The second extension integrates preferential information encoded in a constraint hierarchy to the solution synthesis graph. In section 4.2, we describe how backtracking is handled within the solution synthesis framework when problems are over-constrained. While we highlight the places within this framework where heuristics can be used for efficiently identifying restriction or relaxation candidates in under-constrained or over-constrained situations, we leave the details of these heuristics until the next chapter. In section 4.3, we identify features of problems that make the proposed framework applicable and beneficial.

4.1. Dynamic Solution Synthesis with Constraint Hierarchy

Solution synthesis is applicable for problems when all possible solutions are required and for optimization problems. We use solution synthesis techniques (Freuder, 1978; Tsang and 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 solution synthesis graph (SS-graph). Subsets of base-level nodes are combined yielding higher-level nodes that 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.

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

4.1.1. Dynamic solution synthesis

Applications that utilize solution synthesis are typically static constraint satisfaction problems. We have argued earlier that human-computer interaction is a *dynamic* CSP, because the user's information needs can be refined and modified during the interaction process.

In general, solution synthesis can be extended for DCSPs through operations for adding/removing variables and adding/removing/modifying constraints. Adding or removing variables affects the structure of the SS-graph. For example, in the Essex algorithm, adding a variable to the tail of the base nodes of the existing N variables involves constructing $N+1$ extra nodes, of order $1, 2, ..., N+1$. Other ways of adding variables to the SS-graph are possible. For instance, a variable can be placed in between the ordered nodes at the base level, so that the closely constrained variables can be grouped together. This latter method, however, changes the existing structure of the SS-graph dramatically. Our way to handle added variables is to simply append them to the tail of the ordered nodes at the base level, but synthesize them 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 and the other at level $N+1$ for the top-level node. We treat the operation AddVariables as a special case of UpdateConstraint to be discussed later.

Removing variables from an SS-graph is in general complicated. Basically, when a node that represents the domain of the deleted variable is removed, all the nodes that are children to the node must be either deleted or re-constructed. The location of the deleted variable in the base level ordering can greatly affect the complexity. In the Essex algorithm, for example, removing variables at both ends of the initialization ordering results in the pruning of the left or right frontier branches, which involves N nodes. The closer the removed variable is to the middle of the base level ordering, the more children nodes are to be either deleted or re-constructed. Therefore, in DCSPs, if we know in advance certain variables are more likely to be removed than others, putting these variables at the ends of the initialization ordering can make updating the graph more efficient. 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 information agent and the user;

thus variables never are deleted. Deleting a variable can also be seen as relaxing the constraints on the variable to all possible domain values, in effect, removing any restriction on the variable.

Adding, relaxing or modifying a constraint affects the size of nodes in the solution synthesis graph. For instance, adding a constraint can possibly reduce the sizes of some nodes, while removing a constraint can possibly enlarge the sizes of some nodes. In the Essex algorithm, if there are k variables between variables A and B in the base level ordering, and the constraint between A and B is being updated, then the first node that needs to be re-constructed is of order $k+2$, whose children nodes may also need to be updated if the node is changed.

Adding, relaxing or modifying a constraint in the solution synthesis graph is implemented by the UpdateConstraint procedure illustrated in Figure 4.

UpdateConstraint (Constraint, VofCSP, Nodes)

1. V ← the set of variables from Constraint;
2. NewVars ← NIL;
3. for each variable v in V, if not yet in VofCSP, NewVars ← NewVars + v;
4. if NewVars not empty,
    4.1 create a set of nodes Q for NewVars;
    4.2 Nodes ← Nodes + Q;
    4.3 TopNode ← top node from Nodes;
    4.4 TopNode ← Compose(TopNode,Q);
    4.5 Nodes ← Nodes + TopNode;

/* apply constraint */
5 CurNode ← the node that minimally covers V;
6 update CurNode by applying Constraint;
/* propagate the changes to children nodes */
7 CurNodes ←{CurNode};
8 while CurNodes not empty
    8.1 for each CurNode in CurNodes
        8.1.1 ParentNodes ← children nodes of CurNode;
        8.1.2 CurNodes ← CurNodes - {CurNode};
        8.1.3 while ParentNodes not empty
            for each ParentNode in ParentNodes,

OtherInput $\leftarrow$ other input nodes from ChildNode;

ChildNode $\leftarrow$ Compose(CurNode,OtherInputs);

CurNodes <- CurNodes + {ChildNode};

9 return Nodes;

Figure 4: Procedure for updating a constraint in a solution synthesis graph.

The *Compose* procedure is where solution synthesis happens. It produces all combinations of value assignments for the input nodes that satisfy the constraints in effect for the combination. For adding a variable to the solution synthesis graph, the complexity is $O(a^{c+x})$, in which $O(a^{c})$ is the complexity for the current SS-graph, while $O(a^{x})$ is the complexity for input complexity of the new variables in the constraint, where *x* is the number of new variables in the constraint, and *a* is the maximum number of values in the domain of a variable. In effect, *x* is usually a small number (e.g., less than 3) due to the incremental nature in acquiring constraints in human-computer interaction.

In applying solution synthesis to adaptive solution refinement using data from structured databases (e.g., a relational database), I adopt an incremental approach to solution synthesis, which is able to incrementally construct solution combinations in $O(n^2)$ time and $O(n^2)$ memory, in which *n* represents the maximal number of records in a database. Let us suppose *n* is the maximal number of records in a database, *f* is the number of fields in the database, and *a* is the number of possible domain values for each attribute. At level one, an attribute is combined with its key attribute to form partial solutions. The time complexity of this operation is determined by the query operation of a database. The resulting partial solutions recorded at level two nodes have a memory complexity *O(n).* When level two nodes are combined together using solution synthesis, first product combinations of all possible values are calculated, which requires *intermediate* $O(n^2)$ memory and $O(n^2)$ running time. Illegal value combinations are then filtered using constraints, which requires $O(n^2)$ searching time. Since *n* is the maximal number of records, the result of the filtering will create level two nodes with memory requirement *O(n).* Solution synthesis at level three happens in exactly the same way as at level two.

Note I have chosen to leave out the effects of *f* and *a* so far. In general, *f* increases the memory requirements by a factor of *f* additional attribute values. The factor *a* generally reduces the memory

requirement by eliminating records from consideration. Since the values of f and a are generally small compared to the number of records n, their effects are negligible.

Note that the incorporation of new constraints into the solution synthesis graph can be achieved in different orders. Some orderings may be more efficient than others depending on the structure of the graph and the constraints. For example, Beale (1997) points out that when using solution synthesis to solve static CSPs, a good heuristic is to construct the graph in such a manner as to maximize the benefits of early constraint pruning within sub-graphs, and to minimize the interaction between sub-graphs. We study a number of ordering heuristics in the next chapter.

4.1.2. Solution synthesis with a constraint hierarchy

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 *walkabout strength*, as we have reviewed in section 3.2.1. An SS-graph is different from a constraint graph, but can be transformed into a constraint graph by representing constraints as nodes as opposed to links, and by introducing projection constraints (Freuder, 1978; Freuder, 1995). We adapt the walkabout strength annotation to SS-graphs based on such a transformation.

4.1.2.1. Transforming Solution Synthesis Graph into Constraint Graph

Recall that the annotation of walkabout strengths on variables operates on constraint graphs (section 3.2.1). The nodes in such graphs represent variables, while the arcs represent constraints. A directed arc represents the method chosen in the solution graph in enforcing a given constraint. Walkabout strengths of variables are calculated by looking at the strengths of constraints in which the variable participates, and the strengths of potential output variables of these constraints.

SS-graphs are different from constraint graphs. In an SS-graph, the base level nodes represent the variables in a CSP. Subsets of base-level nodes are combined yielding higher-level nodes that 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.

In order to use walkabout strengths in SS-graphs, SS-graphs need to be interpreted as equivalent constraint graphs. This is possible if an SS-graph undergoes two transformations:

Nodes representing compound labels above the base levels represent variables participating in certain constraints. *K*-ary constraints of the base level variables are translated into *k*-constraints for respective compound variables.

A special constraint (or operation) SS is introduced to represent the relation between the nodes at level *k* and the higher nodes at level *k+1*. This operator is named as *projection constraint* in (Freuder, 1995). The SS operator has one method to enforce the relationship between the nodes:

$N_{v1v2} = SS(v_1, v_2)$ (computing N_{v1v2} from v_1 and v_2)

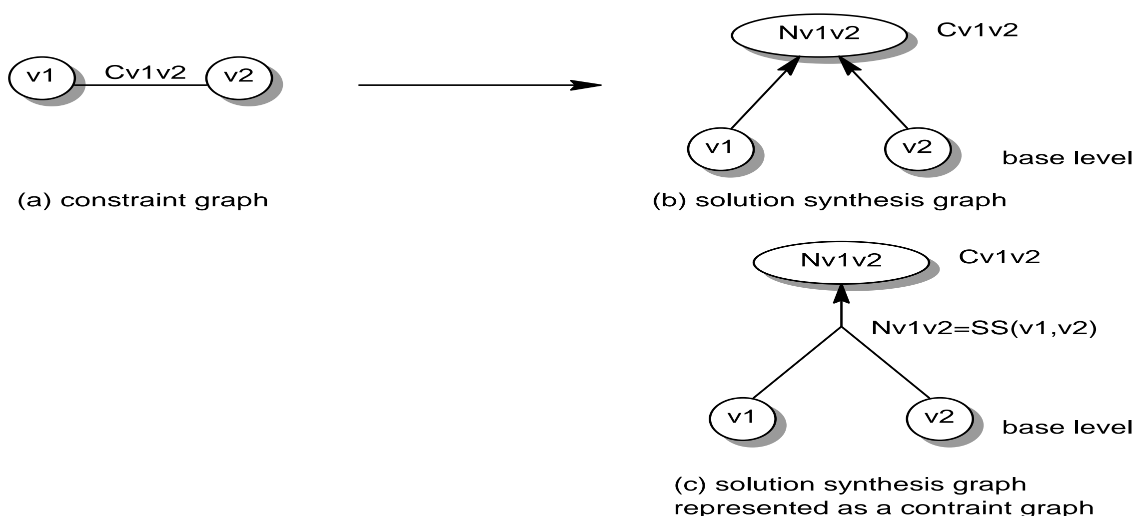


Figure 5: Correspondence between a constraint graph and a solution synthesis graph: (a) is a CSP with two variables v_1 and v_2, and a constraint between the two variables Cv_1v_2. (b) represents how the binary CSP can be solved using solution synthesis. Variables v_1 and v_2 are placed at the base level of the SS-graph. The higher-level node N_{v1v2} is constructed by computing all legal tuples of v_1 and v_2 satisfying the constraint Cv_1v_2. The solution synthesis process is treated as a special constraint SS, as represented in (c).

Figure 5 illustrates the SS-graph for a binary constraint problem and the correspondence between an SS-graph and a constraint graph. The binary constraint Cv_1v_2 between v_1 and v_2 is translated into a binary constraint at node $N_{v1,v2}$. This constraint restricts the set of possible compound labels for instantiation of variables v_1 and v_2. Graph (c) satisfies the definition of a constraint graph, i.e., the nodes represent variables v_1, v_2 and N_{v1v2}, and the arcs represent the projection constraint $N_{v1v2} = SS(v_1, v_2)$.

4.1.2.2. Applying Walkabout Strengths to SS-graphs

Once a solution synthesis graph is transformed into an equivalent constraint graph, walkabout strengths of the nodes in a solution graph can be computed according to the definitions in section 3.2.1. We assume the projection constraint SS has a strength required, so that other weaker strengths can propagate through the projection constraints.

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.

The assignment of the required strength to the base level nodes ensures that constraints over the variables (base level nodes) are candidates for relaxing constraints, but the variables themselves will never be. The default value is set to highest weight required to ensure that weaker constraints can propagate through the SS-graph.

- 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 example, in Figure 6(a) and Figure 7(a), a CSP has three variables V_1, V_2, and V_3. The three variables have stay constraints with strengths $S_{V_1} = required$, $S_{V_2} = weak$, and $S_{V_3} = strong$ respectively. There are three binary constraints C_{V1V2}, C_{V2V3}, and C_{V1V3}, with strengths all being required. Figure 6(b) and Figure 7(b) represent how the CSP can be solved using solution

synthesis. The difference between Figure 6(b) and Figure 7(b) is the ordering of variables V_1, V_2, and V_3 at the base levels. In Figure 6(b), walkabout strengths (*WS*) for the nodes in the solution synthesis graph of this example are calculated as follows:

$$WS_{V_1} = S_{V_1} = required$$

$$WS_{V_2} = S_{V_2} = weak$$

$$WS_{V_3} = S_{V_3} = strong$$

$$WS_{Nv_1v_2} = \min(WS_{V_1}, WS_{V_2}, S_{CV_1V_2}) = weak$$

$$WS_{Nv_2v_3} = \min(WS_{V_2}, WS_{V_3}, S_{CV_2V_3}) = weak$$

$$WS_{Nv_1v_2v_3} = \min(WS_{Nv_1v_2}, WS_{Nv_2v_3}, S_{CV_1V_3}) = weak$$

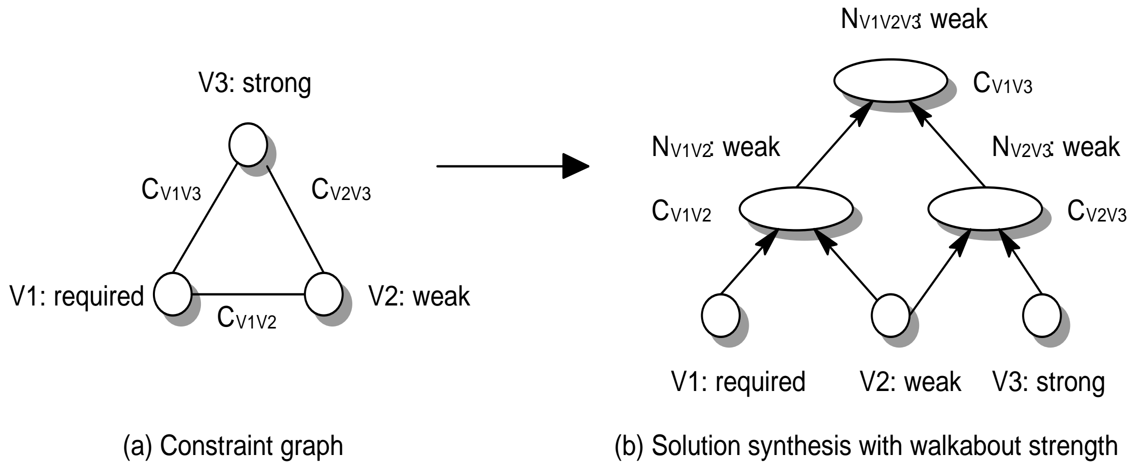


Figure 6: Solution synthesis with walkabout strength. The CSP in (a) has three variables V_1, V_2 and V_3. The three variables have stay constraints with strengths required, weak, and strong respectively. The three binary constraints C_{V1V2}, C_{V2V3}, and C_{V1V3} all have strengths required. (b) represents how the CSP can be solved using solution synthesis with variables in the ordering of V_1, V_2 ,V_3 at the base level. The nodes in the solution synthesis graph are annotated with their respective walkabout strengths.

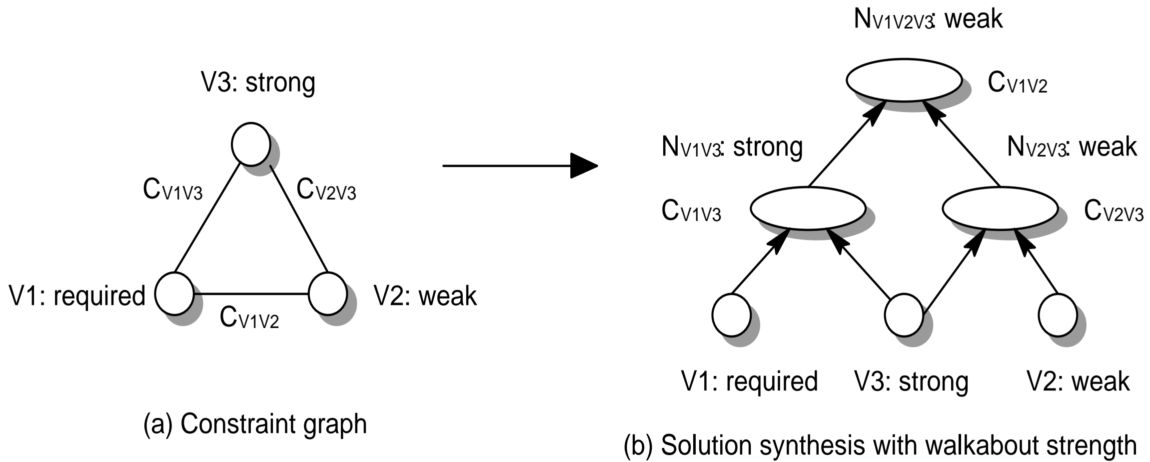


Figure 7: Solution synthesis with walkabout strength. The CSP in (a) has three variables V_1, V_2 and V_3. The three variables have stay constraints with strengths required, weak, and strong respectively. The three binary constraints C_{V1V2}, C_{V2V3}, and C_{V1V3} all have strengths required. (b) represents how the CSP can be solved using solution synthesis with variables in the ordering of V_1,V_3,V_2 at the base level. The nodes in the solution synthesis graph are annotated with their respective walkabout strengths.

Walkabout strengths for the nodes in Figure 7 can be calculated similarly.

4.2. Identification of Relaxation Candidates

When a problem is over-constrained, solution synthesis will yield empty solution tuples when constructing partial solutions. If the application domain permits relaxation of certain constraints (e.g., the user will accept another airline if the preferred carrier is not available), then a necessary step in solving over-constrained problems is to identify what constraints to relax. A basic strategy is to backtrack to the last constraint that has been applied during solution synthesis and re-compute the partial solutions. If the relaxation of the constraint results in solutions, then the problem is solved. If no solutions are found, then more constraints are to be relaxed until either a solution is found or all the constraints are relaxed and no solution is found. Since the algorithm always backtracks to the last constraint when it becomes impossible to proceed, we call this relaxation based on chronological backtracking. The standard procedure for simple chronological backtracking can be found in (Tsang, 1993: p37).

For example, in the solution synthesis graph in Figure 6(b), according to chronological backtracking, the node sequence for backtracking traversal is $N_{v1v2\,v3}$, N_{v2v3}, N_{v1v2}, V_3, V_2, and V_1. The sequencing for constraint relaxation selection is C_{v1v3}, C_{v2v3}, and C_{v1v2}.

In the worst case, the standard chronological backtracking algorithm has a time complexity of $O(a^n e)$ and space complexity of *O(na)*, where *n* is the number of variables, *e* the number of constraints, and *a* is the domain sizes of the variables in a CSP. In the next chapter, we discuss two approaches that make use of the knowledge encoded in the solution synthesis graph to guide backtracking that could potentially increase search efficiency: constraint hierarchy and solution size.

4.3. Usability of the Proposed Techniques

Here we identify the different features in a problem that must come together for the proposed approach to be useful, and briefly illustrate why the proposed approach can be applied naturally to the design of cooperative information systems.

- Need for parallel partial solutions. Many real-world problems cannot be solved in one shot, but rather problem definitions evolve in multiple steps through interaction with the user. Early commitment to a single solution may result in the need for backtracking at later steps, which can be very inefficient for some problems. Solution synthesis allows partial solutions to be constructed and maintained; the partial solutions can be refined by later constraints.
- Relatively independent sub-graphs. Identifying tightly constrained sub-graphs allows us to use constraint satisfaction to reduce the complexity of sub-graphs and reduce the complexity for combination at higher levels. Moreover, sub-graph groupings partition a larger problem into smaller problems. Information about the solution sets of the smaller problems provides important information that can be used by the system for generating appropriate strategies (e.g., deciding on which sub-problems to constrain or relax). Relative independence between sub-graphs allows the system to update solution sets for some partial problems, while maximally reusing the solution sets of other partial problems.
- Constraints of different strengths. In many real-world problems, satisfaction of some constraints is required, while satisfaction of others is only preferred. A constraint hierarchy records the degree of requirements and preferences in terms of numerical values. Walkabout strengths allows the strengths of variables and constraints to propagate through the solution synthesis graph, providing an efficient way for identifying partial solutions that need to be updated during under-constrained and over-constrained situations.

Cooperative information giving is a natural problem that possesses the above features. First, in information giving systems, the user's information needs are seldom specified at one go; thus

solutions generated by the system are at best partial. Solution synthesis allows the system to construct and maintain parallel partial solutions, which can be refined during interaction. Second, usable and practical information systems store their domain data in a structured way. In the commonly used relational database paradigm, data is usually partitioned into multiple entities, which are relatively independent of each other, and are connected by relations. An entity contains a primary key that identifies the entity, together with a set of zero or more mutually independent attribute values that describe the entity in some way. The primary key constrains the other attribute values in that it uniquely identifies the values for these attributes. Knowledge of the structure of a database will enable the system to group attributes into closely-constrained sub-graphs, following their residence within certain entities. Relations between entities allow sub-graphs to be combined into larger problems. Third, a user's information needs usually come with different strengths. A constraint hierarchy is thus necessary for recording the degrees of preferences. Walkabout strengths combined with solution synthesis of sub-graphs provide an efficient way of identifying sub-problems that need to be updated during interaction.

In this chapter, we have shown how partial solutions can be constructed and maintained by the solution synthesis technique, how preferential information can be encoded in such a framework, and how relaxation is carried out within such a framework. In the next chapter, we study a number of heuristics that can potentially improve the efficiency of computation. We hypothesize that systems with the capabilities discussed above support effective and cooperative behaviors in human-computer interaction, which we evaluate using simulation experiments in Chapter 7 and usability experiments in Chapter 9.

Chapter 5 Heuristics for Question Selection and Relaxation Candidate Selection

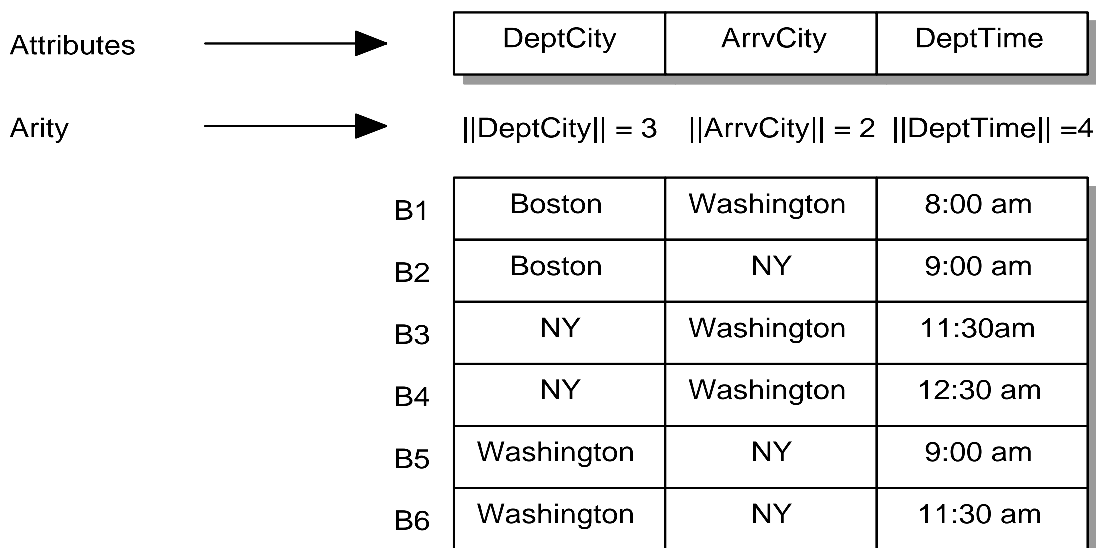
In human-human information-seeking dialogues, the information need of an information seeker is often specified incrementally during the information-seeking process. This is partly due to the interactive nature of dialogue, where the participants take turns to acknowledge and respond. It is also partly due to the fact that both the information seeker and the information provider do not have complete information of each other's knowledge or goals. Thus, interaction is required for proper communication of information and collaboration on problem solving. As a result of these factors, during interaction between collaborators, an information need is only partially specified and the possible solutions to satisfy the information need may be quite large. A cooperative system should take the initiative to guide the information seeker to efficiently locate the solution that maximally satisfies the information need. The order of the questions that a cooperative system chooses to ask can adversely affect the number of questions that it will have to ask. Sometimes, the information need is over-constrained, and a cooperative system should help the information seeker find appropriate task constraints to relax. Different relaxation decisions can also affect dialogue efficiency and task success. In Chapter 4, we have presented a constraint-based framework that supports incremental solution construction and refinement. There, we have left out the details of two issues: (1) how the variables (representing attributes in information needs) should be ordered in building the solution synthesis graph, and (2) how to use the knowledge sources within the solution synthesis graph for identification of relaxation candidates. Maximizing the efficiency in question selection is tantamount to ordering variables in such a way that the solutions can be computed in a minimal number of steps. Therefore, efficient attribute selection addresses variable ordering in the construction of the solution synthesis tree. Maximizing the efficiency and effectiveness in relaxation candidate selection is tantamount to searching for optimal solutions for a constraint satisfaction optimization problem.

5.1. Heuristics for Question Selection

We first discuss the upper and lower bounds for the question selection problem. Then we present a baseline method and propose four heuristic methods for possibly efficient question selection. We compare these heuristic methods in terms of their bias and complexity. Quantitative comparison of these methods will be presented in Chapter 7.

5.1.1. An example problem

Let us look at the question selection problem with an example from the travel domain in Figure 8. Suppose the system has detected a potential solution set of six flights with three attributes – departure city (DeptCity), arrival city (ArrvCity), and departure time (DeptTime) – along with the values of the attributes for each of the branches. The system needs to continue to request constraints on the three attributes to further reduce the solution set. Ideally, the system should ask the question that yields the fewest remaining viable alternatives.



	DeptCity	ArrvCity	DeptTime
B1	Boston	Washington	8:00 am
B2	Boston	NY	9:00 am
B3	NY	Washington	11:30am
B4	NY	Washington	12:30 am
B5	Washington	NY	9:00 am
B6	Washington	NY	11:30 am

Figure 8: Example partial solution set with six solutions for further disambiguation in the airline domain.

If the system is first to request the value for a departure city as in Figure 9, the user may respond with Boston, NY, or Washington[4]. Assuming there is no prior information from user models or domain models, there is an equal probability (33%) that the user will say Boston, NY, or Washington.

If the user responds with Boston, the system must differentiate between the branches B1 and B2 in Figure 8. If the user responds with NY, the system must differentiate between the branches B3 and B4. Similarly, if the user responds with Washington, then the system must differentiate between the branches B5 and B6.

[4] To simplify the illustration, we restrict all responses of the questions to single value answers. In real world situations, it is possible that the responses consist of multiple (i.e., disjunction of) possible values.

If the user responds with Boston as the departure city, the system has two remaining attributes to determine. If the system chooses to request the arrival city next, then either the value NY or Washington as a response will differentiate between branch B1 and branch B2. If the system chooses the departure time instead, branches B1 and B2 can be differentiated in a similar fashion.

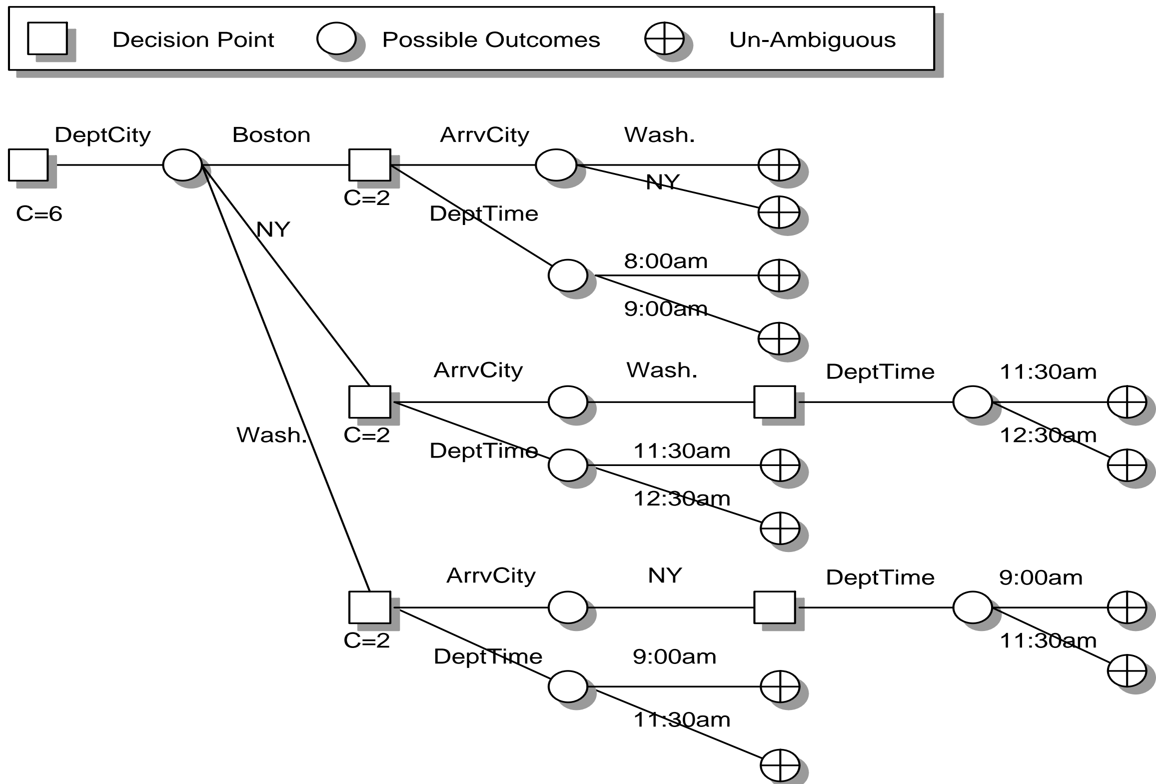


Figure 9: Decision tree when departure city (DeptCity) is requested first.

However, if the user responds with NY or Washington, then it is more efficient for the system to ask about departure time rather than arrival city, since for both departure cities the arrival city is the same, and, hence requesting values for the arrival city would not be sufficient to distinguish between the two flights. If the system chooses to ask about departure time, then branch B3 can be differentiated from B4 and branch B5 can be differentiated from B6, respectively.

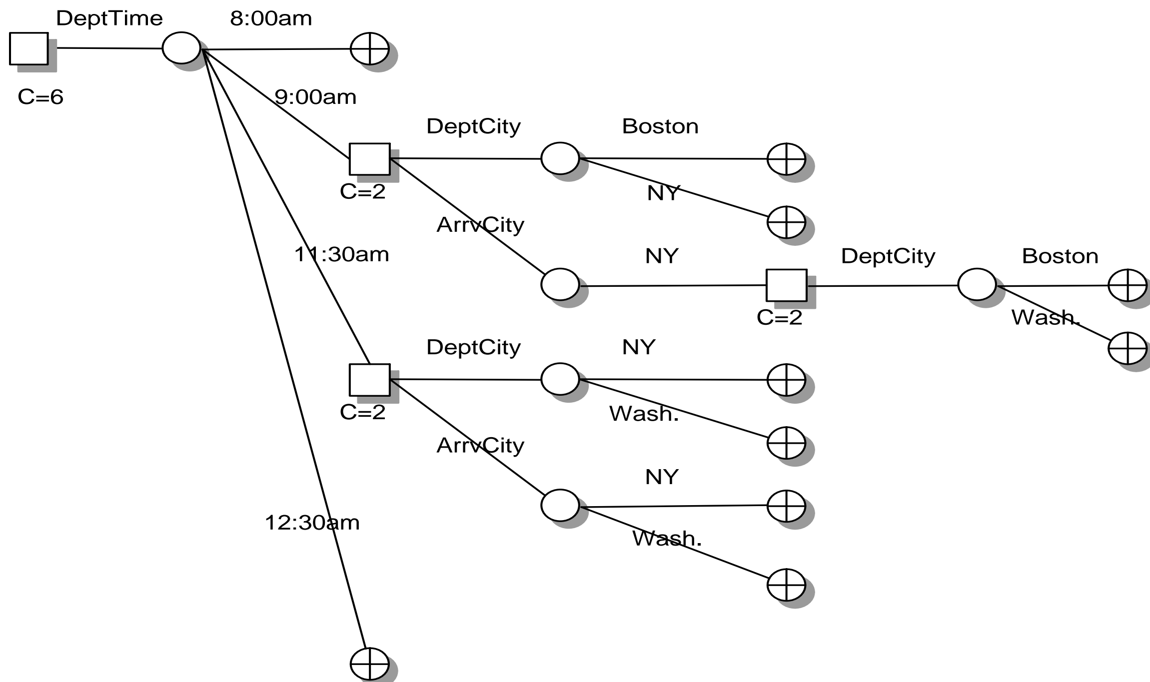


Figure 10: Decision tree when departure time (DeptTime) is requested first.

If the system chooses to ask first about departure time, instead of departure city as in Figure 10, the response of either 8:00am or 12:30am will uniquely differentiate branches B1 from B4 in Figure 8. For the rest of the branches, the system needs to ask additional questions about either the departure city or the arrival city. If the departure time is 9:00am and the value for the departure city is requested, then branches B2 or B5 can be unambiguously differentiated. If the departure time is 9:00am but the value for the arrival city is asked instead, the system has to continue to ask about the departure city because the response of NY as the arrival city is not enough to differentiate branches B2 from B5. If the departure time is 11:30am, then the system can ask about either the departure city or the arrival city, since both questions are equally effective in differentiating branch B3 from B6.

As we can see from Figure 9 and Figure 10, with an ambiguous solution set, requesting values for attributes in essence reduces the number of branches and fewer branches result in reduced ambiguity. Furthermore, selection of an attribute for a question affects the number of potential subsequent questions that need to be asked.

5.1.2. Upper bound and lower bound

For comparing the question selection methods to be introduced later in this Chapter, we first estimate the performance bounds for the question selection problem. The upper bound establishes the maximum number of questions required or available for resolving the ambiguity in the solution set, while the lower bound establishes the minimum number of questions required or available for resolving the ambiguity.

Given a set of *N* attributes $A = \{a_1, ..., a_n\}$ and a solution set *R*, the *upper bound* – the maximum number of questions that can be asked to reduce the ambiguity of the solution set *R* – is bounded by *N-1*.

For example, suppose an attribute set *A* consists of seven attributes: departure city (DeptCity), arrival city (ArriveCity), departure time (DeptTime), arrival time (ArriveTime), day of the week or date (Dow), price (Price), and reward (Reward). The maximum number of questions that can be asked for resolving solution ambiguity is bounded by 7.

Note that this computation of upper bound is based on the assumption that the attributes are independent. In reality, however, many attributes correlate with each other. For example, departure times are in general closely correlated with arrival times after departure cities and arrival cities are determined. Different attribute selection methods account for such dependencies in variable ways. The methods that better account for such dependencies are generally more efficient than those that do not. This will become clear in Chapter 7.

The (expected) *lower bound* is based on the computation of the expected number of questions for each attribute, defined previously by Abella et al. (1996). Our description below is consistent with the computation by Abella et al., but we offer more details on the computation step by step through the example in Figure 8. Let us define the following:

R: a potential solution set
||R||: the number of records in potential solution set *R*
A: a set of attributes
a_i: an attribute in *A*
$A \setminus \{a_i\}$: the set of attributes with attribute a_i removed from A

V: the set of different values of attribute a_i in *R* (domain of a_i in *R*)

S: the set of records that have a given value v for a given attribute a_i (S is a subset of R)

$||S(v, a_i, R)||/||R||$: the probability that the attribute a_i has value v in R

$E(R, a_i)$: the expected number of questions after selecting attribute a_i

$C(R,A)$: the expected number of questions with respect to potential solution set R and attribute set A

We use the following recursive criterion to compute the minimum expected number of questions C with respect to potential solution set R and attribute set A:

$$C(R,A) = 1 + \min_{a_i \in A}\left\{ \sum_{v \in V(a_i,R)} \frac{\| S(v,a_i,R) \|}{\| R \|} C(S(v,a_i,R), A \setminus \{a_i\}) \right\} \quad (1)$$

$$C(R,A) = 0, if \ \| R \| <= k \quad (2)$$

where k is a pre-determined constant that specifies when the recursion stops.

Formula (1) and (2) together determine the selection of an attribute $a_i \in A$ as the first attribute and recursively calls itself on the remaining attributes computing how many expected questions remain to be asked. It computes C for all the attributes in A and chooses the attribute that yields the minimum expected number of questions. The system must ask at least one question. Formula (2) specifies the stop condition for the recursion. When $k=1$, recursion stops when a unique record is identified. When $k>1$, recursion stops when the potential solution set is reduced to a certain size equal or smaller than k.

For example, for the solution set in Figure 1, the lower bound is the minimal expected number of selecting DeptCity (DC), ArrvCity (AC), or DeptTime (DT) as the first question. If DeptCity (DC) is selected as the first question as illustrated in Figure 1, then the subsequent expected number of questions is:

$$E(R,DC) = \frac{1}{3}C(R_{DC=Boston}, \{AC,DT\}) + \frac{1}{3}C(R_{DC=NY}, \{AC,DT\}) + \frac{1}{3}C(R_{DC=Wash}, \{AC,DT\})$$

After asking the first question there is an equal 33% probability that the system will have to ask a second question should the user respond with either Boston, NY, or Washington. For example, if the user responds with Boston, then the system needs to continue to choose between ArrvCity (AC) and DeptTime (DT) for the next question. If ArrvCity is selected as the next question, then $E(R_{DC=Boston}, \{DT\}) = 1$ as either the response Washington or NY can uniquely differentiate the

two branches. Similarly, if DeptTime is selected as the next question, then $E(R_{DC=Boston},\{AC\})=1$ as either the response of 8:00am or 9:00am can uniquely differentiate the two branches. $C(R_{DC=Boston},\{AC,DT\})$ is the minimum of $E(R_{DC=Boston},\{DT\})=1$ and $E(R_{DC=Boston},\{AC\})=1$, i.e., 1.

Similarly, we can compute $C(R_{DC=NY},\{AC,DT\})$ or $C(R_{DC=Wash.},\{AC,DT\})$. The expected number of questions for *E(R,DC)* is then:

$$E(R,DC)=\frac{1}{3}C(R_{DC=Boston},\{AC,DT\})+\frac{1}{3}C(R_{DC=NY},\{AC,DT\})+\frac{1}{3}C(R_{DC=Wash},\{AC,DT\})$$

$$=\frac{1}{3}*1+\frac{1}{3}*1+\frac{1}{3}*1=1$$

Therefore, the total expected number of questions is 1 after the departure city is chosen first.

If the departure time is requested first instead of the departure city as illustrated in Figure 3, the total expected number of questions would be 0.67. This is computed as follows:

$$E(R,DT)=\frac{1}{6}*0+\frac{1}{3}C(R_{DT=900am},\{DC,AC\})+\frac{1}{3}C(R_{DT=1130am},\{DC,AC\}+\frac{1}{6}*0)$$

$$=0+\frac{1}{3}*1+\frac{1}{3}*1+0=0.67$$

When the response is 8:00am or 12:30am, a unique branch can be identified; therefore, no further questions is necessary, i.e., expected number of subsequent questions is 0. When the response is 9:00am or 11:30am, the system needs to continue to request values for departure city or arrival city. The minimal expected number of subsequent questions in both cases is 1.

Similarly, we compute the expected number of subsequent questions to be 1.33 if arrival city is chosen first. The lower bound for this question selection problem is the minimum of *E(R,AC)*, *E(R,DC)*, and *E(R,DT)*. I.e.,

$$C(R,\{AC,DC,DT\})=1+\min(E(R,AC),E(R,DC),E(R,DT))$$

$$=1+\min(1.33,1,0.67)=1.67$$

In summary, given a collection of potential solutions consisting of multiple attribute-value pairs, there are typically many ways of ordering questions to guide the user to a reduced set of sub-solutions that better satisfy the user's information need. We would like to select the attribute that is most efficient for such reduction. The candidate attribute can be selected based on different criteria. In the following, we propose four heuristic methods for selecting attributes with the goal of minimizing the total number of candidates required to reach the goal solution set. These methods are: (1) attribute selection based on maximum branching, (2) attribute selection based on maximum information gain, (3) attribute selection based on constraint hierarchy, and (4) attribute selection based on a fixed ordering. We also describe a baseline method used for evaluating the heuristic methods: random attribute selection.

5.1.3. Attribute selection based on maximum branching

This method selects an attribute as a candidate that splits a potential solution set into the highest number of sub solution sets. This means that the attribute with the highest number of domain values within the solution set is the candidate attribute. If an attribute is an interval type such as departure time or price, the domain values can be grouped together into intervals. The method is motivated by the intuition that widely splitting the solution set scatters the records into much smaller sets, which in turn could lead to the solution state faster with fewer number of subsequent candidates.

For example, in Figure 1, among the attributes DeptCity, ArrvCity and DeptTime, the attribute DeptTime has a branching number of 4, which is greater than either that of the DeptCity (3) or that of the ArrvCity (2). Therefore, DeptTime will be selected as the best candidate with respect to the current solution set using this method.

5.1.4. Attribute selection based on maximum information gain

The second heuristic method uses a statistical property, *information gain*, that measures how well a given attribute separates a solution set according to some target classification. To compute information gain, we need to first define *entropy*, a measure commonly used in information theory, which measures the uncertainty in a data collection.

Given a collection (or potential solution set) *R*, we can think of the target attribute *A* as a random value that can take one of several values *V (A)* and has a probability distribution *P*. Then the entropy of *R* of the random variable *A* is defined as:

$$Entropy(A) = \sum_{v \in V(A)} - p(v) \log_2 p(v)$$

where *p(v)* is the proportion of *R* when *A=v*. The logarithm is base 2, because in information theory, entropy is a measure of the expected encoding length measured in bits. Note that if the target attribute can take on *N* possible values, the entropy can be as large as $log_2 N$.

Given that entropy is a measure of the uncertainty in a collection of data, information gain is simply the expected reduction in entropy caused by partitioning the collection according to an attribute. More precisely, the information gain, *Gain(R,A)* of an attribute *A*, relative to a collection of data (or potential solution set) *R*, is defined as:

$$Gain(R, A) = Entropy(A) - \sum_{v \in Values(A)} \frac{|R_v|}{|R|} Entropy(R_v) \qquad (3)$$

where *Values(A)* is the set of all possible values for attribute *A*, and R_v is the subset of *R* for which attribute *A* has value *v* (i.e., $Rv = \{r \in R \mid A(r) = v\}$). The first term in equation (3) is the entropy of the original collection *R*, and the second term is the expected value of the entropy after *R* is partitioned using attribute *A*. The expected entropy described by this second term is simply the sum of the entropy of each subset R_v, weighted by the fraction of examples $\frac{|R_v|}{|R|}$ that belong to R_v. *Gain(R,A)* is, therefore, the expected reduction in entropy caused by knowing the value of attribute *A*.

Let us look at how information gain is used for identifying an attribute candidate for the example in Figure 1. We first calculate the information gain *Gain(R,DC)* for the attribute DeptCity:

$$Entropy(R) = (-\frac{1}{6} \log_2 \frac{1}{6}) * 6 = \log_2 6$$

$$Entropy(R_{DC=Boston}) = (-\frac{1}{2} \log_2 \frac{1}{2}) * 2 = \log_2 2 = 1$$

$$Entropy(R_{DC=NY}) = (-\frac{1}{2}\log_2 \frac{1}{2}) * 2 = \log_2 2 = 1$$

$$Entropy(R_{DC=Wash.}) = (-\frac{1}{2}\log_2 \frac{1}{2}) * 2 = \log_2 2 = 1$$

$$Gain(R, DC) = Entropy(R) - (\frac{1}{3}Entropy(R_{DC=Boston}) + \frac{1}{3}Entropy(R_{DC=NY}) + \frac{1}{3}Entropy(R_{DC=Wash}))$$
$$= \log_2 6 - 1 = 2.58\text{-}1 = 1.58$$

Similarly, we calculate the information gains for the attributes ArrvCity and DeptTime:

$$Gain(R, AC) = Entropy(R) - (\frac{1}{2}Entropy(R_{AC=Wash.}) + \frac{1}{2}Entropy(R_{AC=NY}))$$

$$= \log_2 6 - (\frac{1}{2} * \log_2 3 + \frac{1}{2} * \log_2 3) = \log_2 6 - \log_2 3 = 2.58 - 1.58 = 1$$

$$Gain(R, DT) = Entropy(R) - (\frac{1}{6}Entropy(R_{DT=800am}) + \frac{1}{3}Entropy(R_{DT=900am})$$

$$+ \frac{1}{3}Entropy(R_{DT=1130am}) + \frac{1}{6}Entropy(R_{DT=1230am}))$$

$$= \log_2 6 - (\frac{1}{6} * 0 + \frac{1}{3} * 1 + \frac{1}{3} * 1 + \frac{1}{6} * 0) = \log_2 6 - \frac{2}{3} = 2.58 - 0.67 = 1.91$$

Since $Gain(R, DT) > Gain(R, DC) > Gain(R, AC)$, or the selection of the attribute DeptTime yields the most reduction in information gain, DeptTime is selected as the candidate.

5.1.5. Attribute selection based on the top-down walk of a constraint hierarchy

The constraint hierarchy is another source of knowledge that a system can employ in question selection. The constraint hierarchy represents the strengths of the user's preferences and restrictions. We distinguish between constraints with the required strength, which must be satisfied to meet the user's information needs, and constraints with preferred strengths (strong, medium or weak strengths for the examples in this thesis), which are preferred to be satisfied as much as possible in a decreasing order of preferences. The heuristic based on constraint hierarchy for question selection is to simply select attributes in the decreasing order of their constraint strengths. The motivation for this heuristic is to ensure that strongly preferred constraints are satisfied first before the problem is solved or becomes over-constrained.

5.1.6. Attribute selection based on fixed ordering

The fixed ordering method takes into account the branching factor. Specifically, it creates an ordering of attributes in decreasing order of the branching factor, i.e., a_i precedes a_j when $branches(a_i) > branches(a_j)$. In contrast to the maximum branching heuristic where the branching factor is dynamically computed with respect to $R\,|_{A - Candidate(A)}$, the fixed ordering method computes the attribute ordering with respect to *R* before any interaction (i.e., attribute selection) begins.

5.1.7. Baseline attribute selection method

We use a random attribute selection method to serve as the baseline for evaluating the question selection heuristics. Given a set of potential attributes *A*, the random attribute selection method randomly chooses an attribute from *A* at each step (i.e., $Candidate(A) = random(A)$). After a candidate is selected, the set *A* is then updated with the chosen attribute removed, i.e., $A = A - \{Candidate(A)\}$.

5.1.8. Comparison of the attribute selection methods

In this section, we compare the attribute selection methods, focusing on their bias and complexity.

We have defined the question selection problem as searching the key solution from a potential solution set *R* with the goal of minimizing the number of attributes required. Which path does the minimal expected path method (i.e., the lower bound) choose? It chooses an optimal ordering of the attributes that minimizes the expected numbers of steps to the goal states, which is to arrive to a subset of solutions of a predetermined size. The search space of all attribute orderings is a complete space of finite numbers of attribute orderings. The search process maintains all possible search paths through the space of attribute orderings before a final selection is made. Therefore, the selection is optimal at each step with respect to the expected number of subsequent steps.

The maximum branching heuristic is a greedy algorithm and is biased towards selecting the attribute that maximally spreads the branches. It is biased towards shorter paths than longer ones, even though it does not guarantee shortest path.

Because of the use of the information gain statistic, the behavior of the maximum information gain heuristic is hard to characterize in an exact way. However, in approximation, it is very similar to the maximum branching heuristic in that it is also a greedy algorithm and it is biased towards shorter paths than longer ones.

The attribute selection method based on calculating the minimal expected number of questions is a very expensive process. Assume that we have *n* attributes with an average branching factor of *b*, the time required for selecting an attribute would be $O(b^n)$ in the worst case.

The random attribute selection method randomly selects an attribute from the attribute set *A* with *n* attributes. The time complexity for selecting each attribute is *O(n)*.

The fixed order method sorts the attributes in *A* based on the branching factor in the static mode. The ordering is fixed with respect to *R*, and does not change as the selection proceeds. The complexity of preparing the ordering is two-fold. First, for all attributes, we calculate the branches of the attributes with respect to *R*, which has a complexity of *O(nR)* for *n* attriubtes. Second, we sort the attributes based on the branching factor in decreasing order, which has a worst-case complexity of $O(n^2)$ depending on the sorting algorithm. At run time, attribute selection is simply to choose the highest ranked attribute from the pre-ordered attribute set, which takes constant time. The attribute set is updated after an attribute is selected by removing the selected attribute.

In contrast to the fixed ordering method, the complexity of the maximum branching method comes mostly from run time: the ordering of attributes is created at run time as the potential solution set is being updated. At each step of attribute selection, we compute the branching factors for each attribute, which has a time complexity of $O(nR)$ for *n* attributes[5]. Subsequently after an attribute is selected, the potential solution set is filtered based on the constraint on the selected attribute and the attribute set is updated by removing the selected attribute. Thus, the sizes of the potential solution set and the attribute set are reduced to $\frac{R}{b}$ and *n-1*, respectively. Note that the time complexity can be further reduced by increasing memory usage to store immediate results in order to reduce the number of times required for looping through the potential solution set for counting the number of branches. We leave the exploitation of the memory usage as future work.

[5] Note that after the numbers of branches are calculated for the attributes, the maximal number needs to be identified, which has a time complexity of *O(n-1)*. Since this is generally much smaller compared to *O(nR)*, we ignore further discussion of this factor for both the maximum branching method and the maximum information gain method.

The maximum information gain method employs a more complicated statistic than the maximum branching method. Nevertheless, the dominating factor in computation is still to calculate the branching factors for the attributes, which has a complexity similar to that of the maximum branching method.

The attribute ordering based on constraint hierarchy sorts the attributes in *A* based on the strength of the constraints in the static mode. The ordering is fixed with respect to *R*, and does not change as the selection proceeds. The complexity of preparing the ordering is simply based on the sorting of the attributes based on the strength values in decreasing order, which has a worst-case complexity of $O(n^2)$ depending on the sorting algorithm. At run time, attribute selection is simply to choose the highest ranked attribute from the pre-ordered attribute set, which takes constant time. The attribute set is updated after an attribute is selected by removing the selected attribute.

	Time complexity	
	Pre-time	Run-time
Minimal expected question	N/A	$O(b^n)$
Maximum branching	N/A	$O(nR)$
Maximum information gain	N/A	$O(nR)$
Constraint strength	$O(n^2)$	Constant
Random selection	N/A	$O(n)$
Fixed ordering	$O(nR)+O(n^2)$	Constant

Table 1: Summary of complexity of the attribute selection methods.

5.2. Heuristics for Constraint Relaxation

In the constraint-based problem-solving framework introduced in Chapter 4, once we construct and maintain solutions in the solution synthesis graph, we can explore the knowledge sources in such a graph for resolving over-constrained situations. In this section, we explore the knowledge sources for identifying relaxation candidates when a problem is over-constrained. For selecting relaxation candidates, we introduce two heuristics: one based on the constraint hierarchy and the other one based on solution size. The baseline relaxation method is the chronological backtracking method described in section 4.2.

5.2.1. Relaxation candidate selection based on constraint hierarchy

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 information agent tries to satisfy the constraints with higher strengths in the constraint hierarchy, while relaxing constraints with lower strengths first. Figure 11 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. Such performance is possible because the algorithm only uses information local to a node, i.e., the walkabout strength, when deciding relaxation choices.

Once an SS-graph is annotated with walkabout strengths to its nodes, the algorithm in Figure 11 can be applied to identify which variable needs to have its values updated upon changes. For example, for the simple CSP with three variables V_1, V_2, and V_3 described earlier in section 4.1.2, suppose that the solution sets at node $N_{V_1V_2V_3}$ are empty in Figure 12 and Figure 13, and that a request for relaxation is posted at node $N_{V_1V_2V_3}$. We want to relax sub-graphs that have the weakest walkabout strengths, and subsequently variables or constraints within such sub-graphs. The algorithm in Figure 11 identifies V_2 as the candidate for relaxation.

```
PROCEDURE LocateRelaxCandidate (SS-Graph)
RelaxCandidate ← NIL;
TopNode ← top level node from SS-Graph
While RelaxCandidate not found,
InputNodes ← input nodes of TopNode
        for each input node InputNode of TopNode
                if walkabout strength of TopNode resulted from a constraint C,
then RelaxCandidate ← constraint C;
        else if walkabout strength resulted from an input node InputNode,
                        then TopNode ← InputNode;
                else if there is a tie between the candidate InputNodes or constraints,
                        then TopNode ← a randomly selected InputNode;
                end if
        next InputNode
Loop
```

Return RelaxCandidate

End Procedure

Figure 11: Use of walkabout strengths for relaxation.

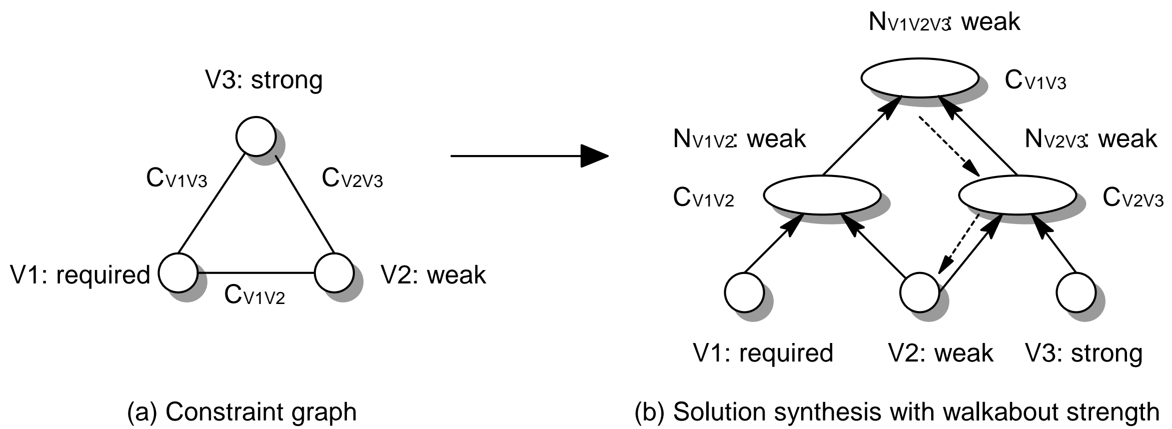


Figure 12: Relaxation in a solution synthesis graph with walkabout strength. The dotted line indicates how walkabout strengths guide relaxation.

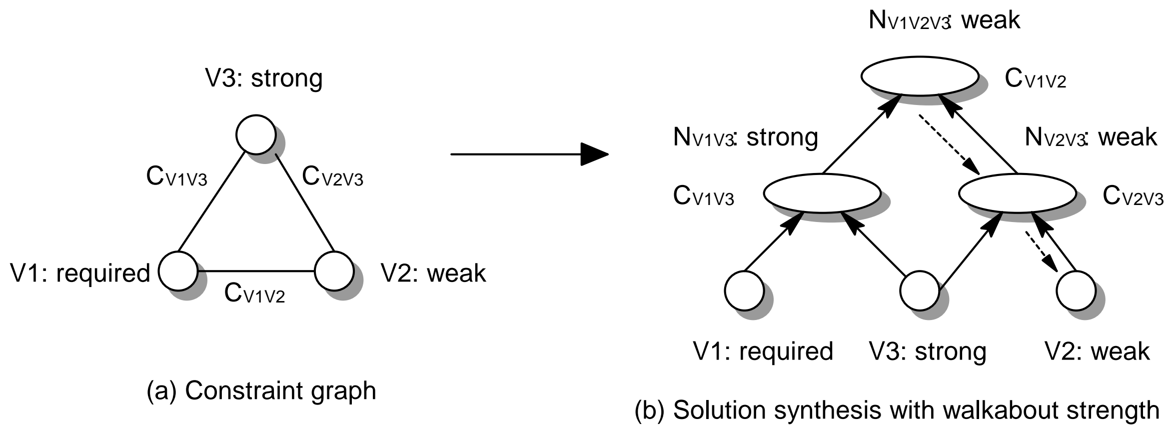


Figure 13: Relaxation in a solution synthesis graph with walkabout strength. The dotted line indicates how walkabout strengths guide relaxation.

Figure 12 and Figure 13 also demonstrate that some orderings of variables at the base level are more efficient for solution update than others. For example, in Figure 12, when V_2 is chosen to be updated (relaxed) to accommodate the changes, $N_{V_1V_2}$ and $N_{V_2V_3}$ at the upper level have to be updated, and then nodes at higher levels. In Figure 13, only the path through V_2, $N_{V_2V_3}$, and $N_{V_1V_2V_3}$ needs to be updated. In this case, partial solutions returned by the sub-graph composed of V_1, V_3, and $N_{V_1V_3}$ can be reused. In general, when constructing a solution synthesis graph, a good heuristic

would be ordering variables in a descending order of their strengths. This will maximize the reuse of previous partial solutions in solution re-construction.

5.2.2. Relaxation candidate selection based on solution size

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 look at solution size to estimate which node is more constrained, which is more likely to be the cause for the over-constrained situations. The solution size information at a node reflects the combined efforts of all the constraints effective over the compound labels of a node. We can annotate the nodes with the solution size information, and employ the information in the solution synthesis graph in the same way as the walkabout strength information.

The solution size information can also be combined with the walkabout strengths in identifying candidates for relaxation. For example, we can give priority to the walkabout strengths over solution size information, and use the latter only for breaking the ties among candidates. Using the combined knowledge sources, we could 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 smallest solution size;

Figure 14: Using dynamic solution size information for breaking ties.

Chapter 6 Evaluation Design: Simulated Tasks

In this chapter, we describe the design for evaluating the effectiveness and efficiency of the different heuristic methods in support of question selection and relaxation selection for simulated information-seeking tasks. Since our focus is to identify the most effective and efficient heuristic methods used by the system to support mixed-initiative information-seeking dialogues, all the experiments are designed to be system-initiative dialogues, emphasizing the system-initiative aspect. These methods aim to maximize mixed-initiative dialogue efficiency by minimizing the number of dialogue turns required to resolve over- and under-constrained situations in information-seeking dialogues. In the following sections, we describe the hypotheses we intend to validate through the experiments, the data collection, the simulated tasks, and the evaluation measures.

6.1. Hypotheses

In the evaluation experiments based on simulated tasks, we intend to identify the most effective and most efficient heuristic methods that can be utilized by the system to support the its initiative taking effort. We compare the performance of the different methods on the grounds of effectiveness and efficiency. Since these experiments are simulated, we do not have a way to actually measure the usability of these methods. Usability is evaluated in part through a limited-scope user study in Chapter 9. Specifically, we evaluate the following hypotheses with respect to question selection and relaxation candidate selection:

- Methods that employ knowledge of problem-solving states produce a more effective and more efficient dialogue in question selection than the random question selection method produces.
- Methods that employ knowledge of users' preferences produce more effective and more efficient dialogue in question selection than the random question selection method produces.
- Methods that employ knowledge of problem-solving states produce a more effective and more efficient dialogue in relaxation candidate selection than the random question selection method produces.
- Methods that employ knowledge of users' preferences produce a more effective and more efficient dialogue in relaxation candidate selection than the random question selection method produces.

6.2. Information-Seeking Task

In the simulated experimental environment, the information-seeking task is defined as follows: a user has a particular information need, possibly involving varying strengths of preferences that are unknown to the system. The system, having the initiative, requests information from the user in order to produce solutions that will satisfy the user's information need. The goal of the system is to produce solutions that satisfy as much as possible the user's information need, while at the same time, minimizing the number of interactions (dialogue turns) required. To achieve this goal, the system iteratively identifies attributes and requests values for these attributes from the user's information request. This corresponds to system-user interaction where the system requests values of attributes from the user and the user provides the system with a value. In our evaluation, we simulate this request-response behavior by obtaining the value of a candidate attribute from the target information need. We assume that not all the attributes describing a flight are useful for the user's information needs. For example, in the airline reservation domain, we generally regard the attributes DeptCity (departure city), ArriveCity (arrive city), DepartTime (departure time), ArriveTime (arrival time), Dow (day of week), and Reward (reward miles) to be useful attributes for locating flights, while discarding all other information such as Equipment (equipment type) or MealCode (meal code) as non-essential.

6.3. Database

All the information-seeking tasks are based on the retrieval of information from a NorthWest Airlines flight schedule database (downloaded from www.nwa.com). The database consists of flights originating from 405 cities to 399 destination cities, with a total of 8457 flights. Each flight record specifies the flight number, the departure city code, the arrival city code, the departure time, the arrival time, the equipment type, flight frequency during the week, the number of stops, the meal service type, and the mileage between the departure city and the arrival city. A more detailed description of the airline database can be found later in Chapter 8.

6.4. Construction of Test Collections

A test collection consists of a set of flights, each of which is considered to represent a particular information need of a user. In the following sections, we will define a user's information needs and describe how we construct different test collections for the evaluation experiments.

6.4.1. Information need

A user's information need is specified as a set of attribute-value pairs. For example, a user's travel constraints include the departure city, the arrival city, the departure time, and price. In the simulated evaluation experiments, a user's information requests are generated based on flight information from the Northwest flight schedule database, user preference strength distributions, task difficulty levels, and task complexity. We will discuss the generation of the test collections in section 6.4.4.

6.4.2. Levels of task difficulty

Based on the availability of flights, information needs are classified into three task difficulty levels. Many factors contribute to the availability of flights, e.g., the departure city, the arrival city, the day of week, or the time of day. For example, it is relatively easy to book a flight from or to a hub city, while it is relatively difficult to book a flight to a city with fewer flights. Flights are more frequent during weekdays than on weekends, and more frequent during the day than at night. In our design, we only assume two of these factors – departure city and arrival city – as the determining factors of task difficulty.

For departure cities, we distinguish three levels of difficulty based on the number of flights departing from that city per week. The three levels Hard, Medium, and Easy are represented by numerical values 3, 2, and 1, respectively (Table 2). Cities with fewer than 40 flights per week are given the Hard level because they are likely to cause over-constrained situations when additional constraints are introduced to the information needs, while cities with more than 101 are given the Easy level because they are unlikely to cause over-constrained situations even when additional constraints are introduced to the information needs.

Number of flights Per week	Task difficulty Level	Number of Such cities	Example cities
1-40	Hard (3)	212	KWI (Kuwait, Kuwait) FRA (Frankfurt, Germany)
41-100	Medium (2)	115	MRY (Monterey, CA) MSN (Madison, WI)
101 and up (max 4486)	Easy (1)	78	MSP (Minneapolis/St. Paul, MN) ATL (Atlanta, GA)

Table 2 : Types of departure city and their task difficulty levels.

Similarly, we distinguish three levels of difficulty for arrival cities based on the number of flights arriving at that city per week. Again, these levels are represented by the same numerical values as

for the departure city (Table 3). Similar to the level assignments to the departure cities, the difficulty levels for the departure cities are based on availability of flights per week that signifies how likely additional constraints will result in over-constrained situations.

Number of flights per week	Task difficulty Level	Number of such cities	Example cities
1-40	Hard (3)	202	FRA (Frankfurt, Germany) ELM (Elmiral/Corning, NY)
41-100	Medium (2)	84	MRY (Monterey, CA) COS (Colorado Springs, CO)
101 and up (max 4518)	Easy (1)	82	MSN (Madison, WI) MSP (Minneapolis/St. Paul, MN)

Table 3: Types of arrival city and their task difficulty levels.

Departure city: task difficulty level	Arrival city: task difficulty level	Flight: task difficulty level
Hard (3)	Hard (3)	Hard (6)
Hard (3)	Medium (2)	Hard (5)
Hard (3)	Easy (1)	Medium (4)
Medium (2)	Hard (3)	Hard (5)
Medium (2)	Medium (2)	Medium (4)
Medium (2)	Easy (1)	Easy (3)
Easy (1)	Hard (3)	Medium (4)
Easy (1)	Medium (2)	Easy (3)
Easy (1)	Easy (1)	Easy (2)

Table 4: Task difficulty levels assigned to flights.

The task difficulty level for an information need is determined by combining the task difficulty factors of the departure city and the arrival city as illustrated in Table 4. For example, if a flight has a departure city of the type Hard (3) and an arrival city of the type Hard (3), then the task difficulty level for the flight is 6, which is defined to be Hard for the flight. We categorize flights with a sum of 5 or 6 as Hard flights, a sum of 4 as Medium flights, and a sum of 2 or 3 as Easy flights. For each level, there can be three types of departure city and arrival city combinations (Table 4). Table 5 provides some sample information needs with task difficulty levels assigned to them.

Task	Flight	DeptC	Arrive	Depart	Arrive	Equip	Dow	Frequ	Stops	MealC	Rewar
3	5235	MDT	CLE	9:45:0	11:15:	AIR	7	X6	0	0	280
4	3964	BGO	OSL	11:00:	11:50:	AIR	3	all	0	0	195
3	6868	IAH	SHV	8:50:0	9:55:0	AIR	5	X6	0	0	192
4	32	DTW	LGW	9:20:0	10:05:	D10	1	All	0	D	3786
2	7863	GSO	IAH	5:35:0	7:16:0	AIR	1	All	0	S	986

Table 5: Sample flight information needs with task difficulty levels assigned**.**

6.4.3. Constraint strength distributions

When we generate an information need, we assign a constraint strength to each attribute by randomly selecting a constraint strength value from its respective distributions. These distributions are collected based on a corpus analysis of naturally occurring airline reservation dialogues (SRI transcripts, 1992). In the corpus analysis, we use two major types of knowledge to detect the strength of a constraint: linguistic cues and conversational circumstances. We classify linguistic cues into 4 levels of constraint strengths as illustrate by the examples in Table 6.

Strength level	Examples (with linguistic cues in bold face)	Constraints
Required	**He needs** the cheapest rate on this one.	Price=cheapest, required
	So the first **I need** to catch the first flight from ORD.	DeptTime=earliest-morning, required; DeptCity=ORD, required
Strong	**I'd like to** arrive SFO around 6am.	ArriveTime=6am, strong; ArriveCity=SFO, strong
	I do like American.	Carrier=American, strong
Medium	Let's say **mayb**e late afternoon.	DeptTime=mid-afternoon, medium
	I suspect five o'clock is **ok**.	DeptTime=5pm, medium
Weak	Let's sign him up with Pam Am or **whatever** we can.	Carrier=PamAm, weak
	Any carrier is ok.	Carrrier=Any, weak

Table 6: Linguistic cues and their corresponding constraint strength levels.

In addition, we identify 5 conversation circumstances to convey the circumstances that can potentially affect the preference strengths. They are:

- **Initiation**: the information seeker provides constraints as part of the problem description, e.g., "*I'd like to arrive SFO around six in the evening.*"
- **Response-Initiation**: the information seeker volunteers constraints when responding to the request of the information provider. E.g., after the information provider suggests San Francisco as the departure city, the information seeker responds with "*actually Oakland would be good too on that.*"
- **Response**: the information seeker responds to the information provider's request for information. Only responses with substantial information are considered responses. E.g., after being asked about the arrive city, the information seeker responds with "*Greenbay*".

- **Confirmation**: the information seeker confirms the constraints that have been presented or discussed. E.g., after exchanging information of departure city, departure time, etc., the information seeker confirms the information exchange with "*So the first I need to catch the first flight from ORD out to Greenbay Monday morning.*"
- **Acknowledgement**: the information seeker acknowledges to confirm understanding. No substantive information is provided. E.g., "*Ok*", "*Hmm*", etc.

The final strength of a constraint is determined by a function of the current linguistic cues, the current conversational circumstance, and its constraint strength in history that keeps track of the constraint strength assigned so far. If there exist linguistic cues that assign a strength level to the constraint, then this strength overrides the history strength and becomes the current strength of the constraint. If there exist no linguistic cues but the constraint has been assigned a strength level in history, then the existing (history) strength will hold. If there exist no linguistic cues and there is no strength assignment in history, then the constraint is assigned a default strength based on the conversational circumstances as follows:

Initiation: strong
Response-initiation: strong
Response: medium
Confirm: medium
Acknowledge: weak

To get the constraint strength distribution for an attribute, e.g., the departure city (DeptCity), we manually analyzed 16 dialogues in the SRI corpus and obtained the final constraint strength for the attribute at the end of each dialogue. The collection of these strength values forms the strength distribution for the attribute. For example, the distribution produced by the corpus analysis for departure city is: required, required, strong, required, required, weak, required, required, required, strong, strong, medium, required, required, required, strong, while the distribution for departure time is: strong, weak, medium, strong, strong, weak, strong, strong, weak, weak, weak, strong, strong, strong, weak, weak.

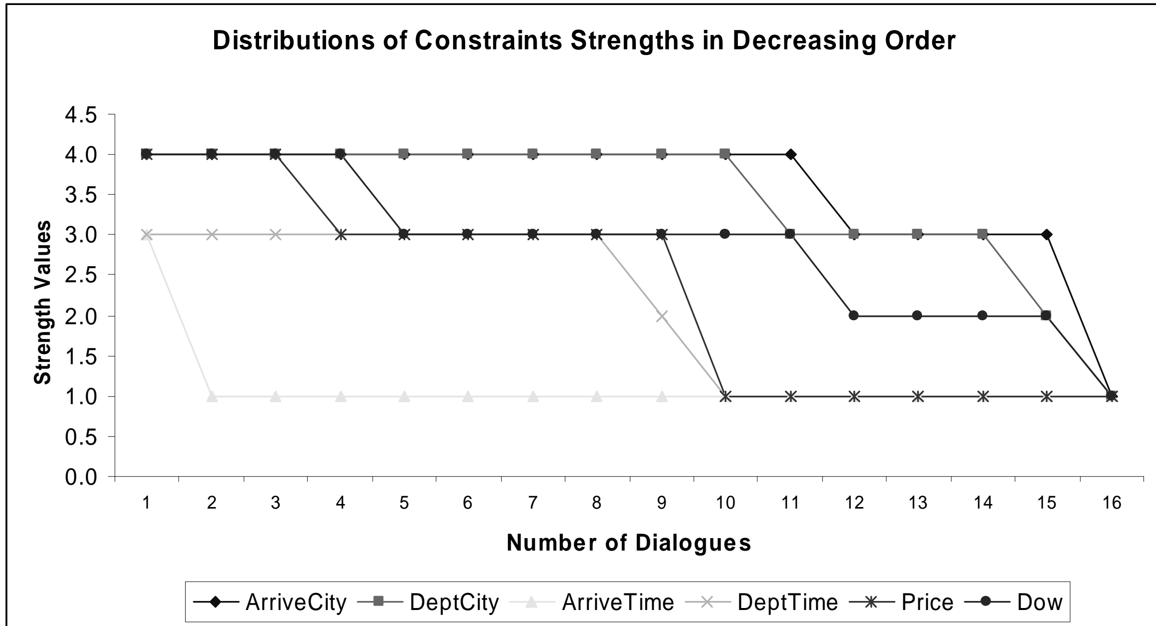


Figure 15: Sample distributions of constraint strengths for attributes in the airline domain.

Figure 15 presents the corpus-based distributions of the constraint strengths in decreasing order of the strength values for attributes ArriveCity, DeptCity, ArriveTime, DeptTime, Price and Dow. The numerical value 4 represents the required strength, 3 the strong strength, 2 the medium strength, and 1 the weak strength. The figure shows that, even though the constraints for each attribute can take values from required to weak, the probability distributions of these attributes would in general give higher weights to DeptCity or ArriveCity, but lower weights to DeptTime or ArriveTime.

6.4.4. Test collections

We have constructed test collections based on two different types of information needs of users. One type of information needs is based on one-leg flights, and the other type is based on three-leg flights.

The first collection, C_1, consists of flights randomly selected from the set of all possible flights, with the values of some attributes randomly varied, so that it is unknown whether or not the flights can be found in the database. To vary the values of some attributes, after we randomly select a flight, we choose to substitute the departure time of a flight with a randomly selected new departure time from the flight database. The arrival time is then updated based on the travel time interval of the original flight. We do not alter the values of the departure cities or the arrival cities. This

guarantees that the reward mileage between the departure city and the arrival city is valid. It also guarantees that there is a possible flight for an information need, even when the information need is over-constrained.

These simulated information needs are categorized by their task difficulty levels: Hard, Medium or Easy. For each task difficulty level, we generate sixty information needs. In addition, these information needs are expanded with constraint hierarchy. Specifically, we assign a constraint strength value to each attribute that constitutes an information need by randomly sampling the constraint strength distribution of that attribute (section 6.4.3). The attributes that are given constraint strengths include DeptCity, ArriveCity, DeptTime, ArriveTime, DOW, and Reward. Since we do not have corpus results for the Reward attribute, we use the distribution of Price instead, based on the assumption that Reward is directly proportional to Price.

The second test collection, C_2, consists of information requests involving three-leg flights. This collection is constructed for evaluating the efficiency and effectiveness of the proposed heuristic methods for tasks with increasing complexity. To construct an information need in this collection, we first randomly select the first leg as one flight from the database. Then the subsequent legs are also randomly selected from the database, observing two general constraints. First, the departure city of the next leg should be the same as the arrival city of the previous leg. Second, when the legs are traveled in the same day and the arrival time of a previous leg is required, the departure time for the next leg should be at least 45 minutes later than the previous arrival time.[6] This second constraint is aimed to ensure that ample connecting time is provided. Similar to collection C_1, the departure time of each leg is randomly modified, so it is unknown whether an information request is over-constrained or not. Each leg is extended with constraint strengths for its attributes, but the constraint hierarchies for different legs are assumed to be independent. The legs within a single complete information need can belong to different task difficulty levels; we do not categorize task difficulty levels for a single information request. Twenty 3-leg 2-destination flight information requests are generated with the final destination city being the same as the departure city. Another twenty 3-leg 3-destination flight information requests are generated with the final destination city different from the departure city.

[6] In general, international flight connections require more connecting time than domestic flight connections. In the Northwest flight database, the minimum required domestic connecting time is 30 minutes, while that for the international flight is 90 minutes.

6.5. Design of the Evaluation

With test collections C_1 and C_2, we compare different heuristic methods for varying parameter settings against given performance measures. In this section, we enumerate the heuristic methods used in the evaluation and the parameter settings.

6.5.1. Combinations of the heuristic methods

During the information-seeking process, either under-constrained situations or over-constrained situations can occur. Thus, the initiative behavior of the system relies on a combination of (1) strategies for question selection in under-constrained situations and (2) strategies for relaxation candidate selection in over-constrained situations discussed in the previous chapter. Table 7 presents combinations of question selection and relaxation candidate selection heuristics that are compared in our experiments. For example, the MaxBranch-Simple method means that the maximum branching method is used for question selection and the simple backtracking method is used for identifying relaxation candidates.

Method Abbrs.	Question selection	Relaxation candidate selection
MaxBranch-Simple	Maximum branching	Simple backtracking
MaxBranch-MinSize	Maximum branching	Minimum solution size
Const-Simple	Constraint hierarchy	Simple backtracking
Const-Const	Constraint hierarchy	Constraint hierarchy
MaxInfo-Simple	Maximum information gain	Simple backtracking
FixedOrder-Simple	Fixed ordering	Simple backtracking
Random-Simple	Random selection	Simple backtracking
Random-Const	Random selection	Constraint hierarchy
Random-MinSize	Random selection	Minimum solution size

Table 7: Combination of heuristic methods employed by the system.

The MaxBranch-Simple, Const-Simple, MaxInfo-Simple, FixedOrder-Simple, and Random-Simple methods are designed to evaluate the effectiveness and efficiency of the different question selection methods while keeping simple backtracking as the relaxation candidate selection method. The Random-Simple, Random-Const, and Random-MinSize are designed to evaluate the effectiveness and efficiency of the different relaxation candidate selection methods while keeping the random selection method as the question selection method. Additional comparisons are designed between the MaxBranch-Simple method and the MaxBranch-MinSize method and between the Const-Simple method and the Const-Const method to determine the interaction effect between the question selection methods and the relaxation candidate selection methods.

6.5.2. Experimental parameters

Many factors can influence the effectiveness and the efficiency of these heuristic methods. In this study, we investigate the effect of four factors with respect to these heuristic methods: goal state size (*G*), interval size (*I*), task difficulty level (*T*), and the number of attributes in an information request (*N*). These factors are defined below:

- *G*: the number of solutions in a set of possible solutions that is considered the goal state. For example, *G=5* means that the system reaches a goal state when five or fewer solutions are included in the solution set.
- *I*: the different interval sizes for interval variables such as DeptTime, ArriveTime, and Reward.
- *T:* the task difficulty levels of an information need. Currently, we distinguish three levels of task difficulty – Easy, Medium, and Hard – based on the task difficulty levels of departure city and arrival city.
- *N:* the number of attributes in an information need. Currently, *N* can take two values: 6 for one-leg flight retrieval problems and 18 for three-leg flight retrieval problems. Performance on this measure demonstrates the scalability of methods from simple to complex problems.

This experimental design permits insights into the relationship between problem structure, as defined by these parameters, and the performance of different question selection and relaxation candidate selection methods.

6.6. Performance Measures

6.6.1. Task completion

The efficiency and effectiveness of a method for a particular problem are computed after task completion. The *stop conditions* depend primarily on whether the system is capable to supply solutions to the user with a solution set whose size is equal or smaller than a pre-determined goal state size *k*, which satisfies the user's original or relaxed information needs. Specifically, if solutions exist and the size of the solution set is smaller than a pre-determined goal state size *k*, then question selection stops and the system is ready to provide the solutions to the user. If all the attributes in the attribute set have been selected, question selection also stops, since the system has no available information to further reduce the ambiguity of the solution set. If the solution set is

empty, then constraint relaxation should be conducted; however, if all constraints are required and there is no candidate for relaxation, then question selection stops.

6.6.2. Task success

We use the kappa statistic to measure the effectiveness of the combination methods in satisfying the user's information needs. The kappa statistic is used to calculate the degree of satisfaction of the user's information need (Carletta, 1996). Specifically, we compare the information in a system solution specified by attribute-value pairs with the attribute-pairs in a user's original information need. The kappa statistic is defined as

$$k = \frac{P(A) - P(E)}{1 - P(E)}$$

where $P(A)$ is the proportion of times that the attribute-value pairs in the system solution agree with the attribute-value pairs in the information need and $P(E)$ is the proportion of times that the attribute-value pairs in the system solution and the attribute-value pairs in the information need are expected to agree by chance. When all task information items are successfully instantiated, the agreement is then perfect and $k=1$. When agreement is only by chance, then $k=0$. k accounts for the inherent task complexity by correcting for agreement expected by chance.

Because of several characteristics of our experiment design and the data, we have adapted the computation of the kappa statistic in the following ways:

- The system response for a given information need in our experiments generally consists of a set of solutions of size n or less. To measure the kappa statistic for this set, we first compute the kappa score for each solution with respect to the information need, and then select the maximum kappa score as the final kappa score for the solution set.
- When $P(E)$ is unknown, it can generally be estimated from the distributions of values in the keys (Walker et al., 1997). In our domain, we have the complete population of the flights available; therefore, we are able to compute the prior distributions of the values for each attribute. From the prior distribution p of an attribute, we estimated the $P(E)$ of that attribute as

  $$P(E) = p^2$$

We observe that $P(E)$ is very different across attributes, ranging from 0.031 for the DeptCity attribute to 0.144 for the Dow attribute. A separate empirical attempt to estimate $P(E)$ through Monte Carlo simulations provides similar results.

- In (Walker et al., 1997), a single confusion matrix is used for all attributes. Such arrangement inflates the kappa scores when there are few cross-attribute confusions by making $P(E)$ smaller. In our evaluation experiments, no agreement occurs across attributes. Therefore, we first compute the kappa statistic for each attribute in a solution with respect to an information need, then we take the average kappa scores of all the attributes in an information need for the task.
- When the constraints over the attributes (or the attribute-value pairs) have different preference strengths, we first compute the kappa statistic for each attribute, then take the weighted average k of all the attributes for the task.

To summarize, the procedure for computing the kappa score for the system's response with respect to an information need is as follows. If the system response is an empty set of solutions even after constraint relaxation, i.e., the problem is unsolvable, then we discard the information need for the purpose of calculating kappa scores. If the size of the solution set is greater than a pre-determined goal state size n, then unique n solutions are randomly selected from the solution set. This simulates the situations when too many solutions are found and the system can only display a portion of the solutions as determined by n. For any solution set with a solution size of n or less, we calculate the weighted average kappa score for the set.

For under-constrained information needs, if the problem solver stops due to running out of attributes to select, the key solution is guaranteed to be in the solution set. It is likely, however, that the size of the solution set provided by the system is greater than the goal state size n. In these cases, the system randomly selects n unique solutions from the current solution set, and treats the new set as the final solution set for kappa computation. In such cases, task success measured by kappa is always 1. If the problem solver stops because the goal state is reached, then the information request is not always guaranteed to be satisfied by the solutions in that set. It is possible that the constraints for the unselected attributes cannot be satisfied. Consequently, the kappa score can have a value less than 1.

For over-constrained information requests, question selection could be stopped either when the problem is unsolvable or when the size of the solution set provided by the system satisfies the limit of the goal state size. In the former case, since no relaxation is possible for obtaining a solution, the

problem is unsolvable, and we discard such cases from kappa score computation. In the latter case, it is likely that he solution set results from constraint relaxation. Thus, the kappa scores are generally less than 1 and can be negative when no agreement is observed.

6.6.3. Dialogue efficiency

We use two types of measures to evaluate dialogue efficiency of a cooperative information system: *question selection efficiency measures* and *relaxation efficiency measures*.

Question selection efficiency measures are used to evaluate the efficiency of question selection methods during the process in which the system is trying to find solutions to satisfy the user's information needs. The measures for evaluating question selection efficiency include:

- Average number of questions required per problem: measuring the average number of questions necessary to be solicited by the system before the system can provide the user with solutions to satisfy her information need. In both the under-constrained and over-constrained situations, this number is the number of questions the system can ask to reach the pre-determined goal state. The maximum number of questions is bounded by the size of the attribute set. The smaller the number of questions required to be solicited, the more efficient the dialogue is.
- Average total time for question selection per problem: measuring the average system time required for selecting all questions for solving a problem.

Relaxation efficiency measures are introduced to measure the efficiency of the relaxation methods in solving over-constrained problems. The measures used for evaluating relaxation efficiency include:

- Average number of relaxation requests per problem: measuring the average number of relaxation requests that the system has to be engaged in before solutions can be obtained. We want to minimize the number of relaxation requests in system and user interaction by identifying the most effective candidates for relaxation in the user's information need.
- Average total time for relaxation candidate identification per problem: measuring the average system time required for identifying relaxation candidates to successfully solve a problem.

6.7. Statistical Hypothesis Testing

Suppose we run the Random-Simple method and the MaxBranch-Simple method (Table 7) over ten test problems and collect the number of questions required for solving each problem in Table 8. We want to determine whether the MaxBranch-Simple method is more efficient in solving the problems than the baseline Random-Simple method in terms of the numbers of questions.

	1	2	3	4	5	6	7	8	9	10	$\bar{x}$
Random-Simple (A)	5	4	3	6	5	6	6	3	4	6	4.8
Maxbranch-Simple (B)	4	3	3	4	4	3	3	3	3	3	3.3

Table 8: Hypothetical numbers of questions for a test sample with 10 problems.

For the Random-Simple method, we calculate the average number of questions for the problem set:

$$\bar{x}_1 = \frac{5+4+3+6+5+6+6+3+4+6}{10} = 4.8$$

Similarly, we calculate the average number of questions for the MaxBranch-Simple method:

$$\bar{x}_2 = \frac{4+3+3+4+4+3+3+3+3+3}{10} = 3.3$$

Therefore, on average, we observe that the MaxBranch-Simple method requires fewer questions than the Random-Simple method. The lower score is gratifying, but is the difference due to the improved efficiency of the MaxBranch-Simple method or is it due to the random noise in sampling?

In this work, we estimate the significance of the differences between different methods using two hypothesis testing methods: the paired sample *t*-tests and analysis of variance (ANOVA).

6.7.1. Hypothesis testing

In this thesis, a typical hypothesis is formulated as follows:

> A method A is better (e.g., more effective or more efficient) as compared to a method B, in the same task situation with the same parameter settings, if the paired sample *t*-test (to be described in the next section) value between the results of method A and method B is significant at $p \leq 0.05$, in the positive direction, where p is probability of error for accepting this hypothesis.

6.7.2. The paired sample *t*-test

In most of our experiments, we select a test collection and run a pair of competing methods over the collection to evaluate which method is more efficient or effective. For example, suppose we want to compare the performance of method A and method B over the same set of problems, we can formulate the null hypothesis H_0 (that the performance between A and B is equal) and the alternative hypotheses H_1 (that method A performs better or worse than method B) as follows:

$$H_o : \mu_A = \mu_B$$

$$H_1 : \mu_A > \mu_B \text{ (one-tailed test)}$$

$$H_1 : \mu_A < \mu_B \text{ (one-tailed test)}$$

where μ_A and μ_B are the population means for method A and B, respectively.

To evaluate these hypotheses, we use two-sample *t*-tests for pairwise comparisons of group means (Cohen, 1995). For instance, for the sample data in Table 8, we obtain a *t* value of 4.025, which is greater than the one-tailed *t* critical value 1.833 at $p=0.05$[7]. Therefore, we can reject the null hypothesis at the .05 level and claim that the MaxBranch-Simple method is significantly more efficient than the Random-Simple method (i.e., $H_1 : \mu_{Random-Simple} > \mu_{MaxBranch-Simple}$ holds).

6.7.3. Analysis of variance (ANOVA)

Pairwise comparisons like *t*-tests have some pitfalls, however. First, the approach has merit when only few groups of data are to be compared. If there exist a large number of groups (e.g., *N* groups with a large *N*), then comparing $N*(N-1)/2$ pairs can be quite cumbersome. Second, suppose that the *N* groups come from the same population. With a large *N*, the probably of incorrectly rejecting the null hypothesis that all groups come form the same population is quite large when $N*(N-1)/2$ pairs of *t*-tests are run. To illustrate, assuming the probability of incorrectly concluding that two groups are drawn from different populations is $p=0.05$. Then the probability of incorrectly concluding that all groups are not drawn from the same population is $1-(1-p)^{N*(N-1)/2}$, which can be very large when *N* is a larger number (Cohen, 1995: p189-192).

---

[7] All the *t*-test results and the ANOVA test results reported in this work are obtained through Microsoft Excel 2000.

Third, when multiple factors are contributing to performance differences, there might be interaction effects between these factors. For example, a question selection method may perform better than other methods only at certain task difficulty levels. In such cases, the question selection method and the task difficulty level interact together. To measure the interaction effects between factors, a pair of one-factor experiments is not sufficient. To demonstrate such interaction effects, we need two-factor or multiple factor experiments.

Analysis of variance is a technique of testing hypothesis of the population means of *several* groups. The technique also supports discovery of interactions between multiple factors in experiments. Analysis of variance can be one-factor, two-factor, three-factor, or multiple factor. The interested reader is referred to (Cohen, 1995) for details of this technique.

In analyzing the experimental results in this thesis, we first use the ANOVA approach to identify the effects of individual factors and the interaction effects between them using mostly two-factor analysis of variance of several groups. For detailed comparisons between groups, we further use pairwise comparisons of group means. Cohen (1995: p195-199) discusses two such tests for pairwise comparison of means – *Scheffé tests* and *least significant difference (LSD) tests*. For our analysis, we will simply use the standard *t*-tests. Fully aware of the potential pitfalls of the spurious differences of the *t*-tests, we conduct pairwise comparisons only for cases when ANOVA results demonstrate significance.

Chapter 7 Result Analysis of Simulated Experiments

This chapter reports the results of an experimental study comparing the question selection methods in under-constrained situations and the relaxation candidate selection methods in over-constrained situations. For under-constrained problems, we evaluate four heuristic attribute selection methods described in Chapter 5: attribute selection based on fixed ordering, attribute selection based on maximum branching, attribute selection based on maximum information gain, and attribute selection based on constraint hierarchy. The baseline for evaluating these methods is random attribute selection. For over-constrained problems, we evaluate two heuristic-based methods for identifying relaxation candidates described in Chapter 5: relaxation candidate selection based on solution size and relaxation candidate selection based on constraint hierarchy. The baseline method for identifying relaxation candidate is chronological backtracking. Compared with the baseline methods for either question selection or relaxation candidate selection, the heuristic methods take advantage of the additional information available in the partial problem solving process. In the following sections of this chapter, we investigate the relationship between the parameter settings – task difficulty, goal-state size, interval size, and task complexity – and the performance of the different methods.

7.1. Result Analysis: The Effect of Task Difficulty

We first examine the effect of task difficulty when using different heuristic combination methods for problem solving. The task difficulty levels are Easy, Medium, and Hard. The goal-state size is set at *G=5*. The experiments were conducted using collection *C1*, which consists of the one-leg flight reservation problems of three task difficulty levels (section 6.4.2). The attributes used in describing the information needs consist of attributes DeptCity, ArriveCity, DeptTime, ArriveTime, Reward, and DOW. The time interval chosen for DeptTime and ArriveTime is one hour. The fixed ordering method uses the attribute ordering: $DeptCity > ArriveCity > \mathrm{Re}ward > DeptTime > ArriveTime > DOW$, which is ordered according to the decreasing order of the cardinality of the attributes.

7.1.1. Task difficulty and question selection efficiency

To examine the effect of task difficulty on question selection efficiency, we investigate three questions:

- For a particular question selection method, what is the effect of different task difficulty levels?
- For problems of a particular task difficulty level, what is the most efficient method for question selection?
- Is there a method that outperforms other methods in terms of question selection across all task levels?

We compare five methods: Random-Simple, FixedOrder-Simple, MaxBranch-Simple, Const-Simple, and MaxInfo-Simple. In these experiments, we vary the question selection methods, while keeping the relaxation candidate selection method constant as the chronological backtracking method. The maximum number of questions that can be asked by the system is bounded by 6. We discuss the results for the two question selection efficiency measures – the number of questions required for solving a problem and the time required for identifying these questions –in the following subsections.

7.1.1.1. Task difficulty and the average number of questions elicited by the system

We first look at the effect of task difficulty on the number of questions elicited by the system. Figure 16 presents a summary of the average number of questions per problem using different heuristics for question selection. We make several observations regarding this figure:

For all the methods (except the MaxInfo-Simple method at Medium task difficulty level), the average number of questions elicited by the system decreases as the task difficulty increases.

When comparing the efficiency of these methods based on the average number of questions per problem, the MaxBranch-Simple method is the most efficient method, with Const-Simple method as the second most efficient. The FixedOrder-Simple method is the least efficient, while the MaxInfo-Simple and the Random-Simple are the middle performers

Let us examine the first observation in detail. This observation is not surprising considering how task difficulty is categorized. Recall that task difficulty levels are determined based on the number of flights available at departure-arrival city pairs during a certain time period (section 6.4.2). An easier problem means that there are more possible flights with respect to a departure-arrival city pair. Consequently, for easier problems, more constraints need to be elicited to reduce the potential solutions to a pre-determined goal size or to locate the specific flights that satisfy a user's information needs.

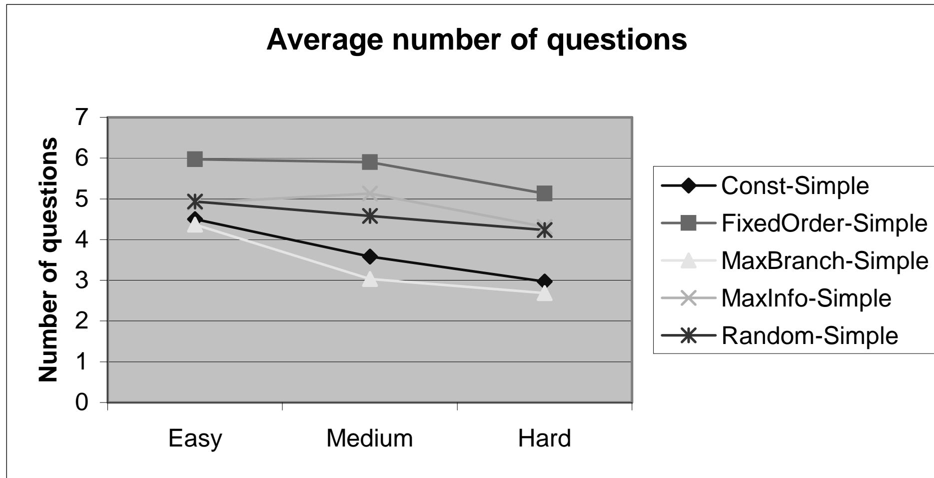


Figure 16: Average numbers of questions elicited by the system across different task difficulty levels with different question selection methods.

Table 9 summarizes the results of analysis of variance for all the methods across the three task difficulty levels. Compared to the .05 critical value, all the *F* values are fairly large. We conclude from this analysis that task difficulty levels affect the average number of questions elicited by the system. The conclusion holds true for all the question selection methods.

Methods	*F-value*	*P-value*	*F crit*
MaxBranch-Simple	35.081	<0.001	3.047
Const-Simple	29.333	<0.001	3.047
FixedOrder-Simple	11.970	<0.001	3.047
MaxInfo-Simple	12.585	<0.001	3.047
Random-Simple	9.935	<0.001	3.047

Table 9: *F*-values and *p*-values across task difficulty levels for different question selection methods.

In Table 10, we present pairwise percentage decrease numbers between Easy and Medium problems and between Medium and Hard problems. For all the paired comparisons except Easy vs Medium using the MaxInfo-Simple method, we observe decrease in the number of questions from Easy to Medium problems (ranging from 1.1% to 30.5%), and from Medium to Hard problems, respectively (ranging from 7.6% to 17.2%). An exception is with the MaxInfo-Simple method between Easy and Medium problems, where the average number of questions actually increased 4.8%.

One-tail *t*-test results in Table 10 show that for both the MaxBranch-Simple and the Const-Simple methods, the average number of system-elicited questions is significantly smaller with problems of

Medium task difficulty level compared with problems of Easy task difficulty level ($p<0.001$, one-tail for both methods). Similarly, significantly fewer number of questions is required for Hard problems compared with Medium problems ($p=0.038$, one-tail for MaxBranch-Simple and $p=0.001$, one-tail for Const-Simple, respectively).

Methods	*Easy vs Medium*		*Medium vs Hard*	
	%decrease	*p-value*	*%decrease*	*p-value*
MaxBranch-Simple	30.5%	<0.001	11.5%	0.038
Const-Simple	20.4%	<0.001	17.2%	0.001
FixedOrder-Simple	1.1%	0.207	13.0%	0.001
MaxInfo-Simple	(4.8%)	0.053	15.9%	<0.001
Random-Simple	7.1%	0.002	7.6%	0.061

Table 10: Pairwise *t*-test results between task difficulty levels for different question selection methods.

With the FixedOrder-Simple method, we cannot reject the null hypothesis that Easy and Medium problems require the same average number of questions. We can, however, claim that significantly fewer questions are required for Hard problems compared with Medium problems at $p=0.001$ level.

With the MaxInfo-Simple method, it takes more questions to solve the Medium problems than the Easy problems. Since $p=0.053$, higher than the conventional .05, we cannot reject the null hypothesis that the average number of questions of the Easy and Medium problems are equal. We can, however, claim that significantly fewer questions are required for the Hard problems compared with the Medium problems at $p<0.001$ level.

With the Random-Simple method, significantly fewer questions are needed for solving the Medium problems compared with Easy problems ($p=0.002$), but the average numbers of questions required for solving the Medium and Hard problems do not have significant difference.

The second observation from Figure 16 is that different heuristic combination methods behave differently in terms of efficiency. Across all the task difficulty levels, the MaxBranch-Simple and the Const-Simple methods performed better than the Random-Simple method, while the FixedOrder-Simple method fared worse compared with the Random-Simple method. The performance of the MaxInfo-Simple method is very close to the Random-Simple method. Paired

two sample t-tests show that the MaxBranch-Simple method is significantly more efficient than the Const-Simple method (*p<0.001*, one-tail), which is in turn significantly more efficient than the Random-Simple method (*p<0.001*, one-tail). On the other hand, the Random-Simple method is significantly better than the FixedOrder-Simple method (*p<0.001*, one-tail). However, there is no significant difference observed between the MaxInfo-Simple method and the Random-Simple method (*p=0.07*, one-tail).

Given the general trend observed above, we want to see, at a given task difficulty level, what methods perform better than the random selection method and whether there is a method that is significantly more efficient than the other methods.

Method	*Easy*		*Medium*		*Hard*	
(vs Random-Simple)	*Decrs/incrs*	*p-value*	*Decr/incr*	*p-value*	*Decr/incr*	*p-value*
MaxBranch-Simple	-11.5%	0.002	-33.8%	0.001	-36.6%	0.001
Const-Simple	-8.8%	0.015	-21.8%	0.001	-29.9%	0.001
FixedOrder-Simple	21.0%	0.001	28.7%	0.001	21.3%	0.001
MaxInfo-Simple	-0.7%	0.425	12.0%	0.003	2.0%	0.352

Table 11 : Pairewise comparisons between the different heuristic question selection methods and the random question selection method at the three task difficulty levels.

From Table 11, we can see that the MaxBranch-Simple method and the Const-Simple method perform significantly more efficiently than the Random-Simple method across all task difficulty levels. The MaxBranch-Simple method improves efficiency by 11.5% to 36.6%, while the Const-Simple method improves efficiency by 8.8% to 29.9%. The improvements in efficiency are statistically significant. On the other hand, the FixedOrder-Simple method is significant less efficient compared with the Random-Simple method, increasing the number of questions by 21.0% to 28.7%. This drop in performance is also statistically significant. The MaxInfo-Simple method is less efficient than the Random-Simple method for Medium and Hard problems, increasing the number of questions required by 2% and 12.05, respectively. For the Easy problems, it shows small improvement in efficiency (0.7%). Only the difference observed for Medium problems is statistically significant.

It is worthwhile to examine what factors influence what attributes to select during the attribute selection process. An implicit assumption in all our attribute selection methods so far is that the attributes are independent of each other. Thus, selection of one attribute has no influence over the selection of other attributes. In reality, however, the attributes are not completely independent. For

instance, once departure cities and arrival cities are selected, the reward miles between these two cities are uniquely determined for non-stop flights. Or, when the departure cities and arrival cities are selected and departure times specified, the arrival times are generally determined for non-stop flights. The random attribute selection method selects an attribute randomly from the attribute set; therefore, the dependencies between attributes are not accounted for.

Among the five methods that we have evaluated, the Random-Simple method, the Const-Simple method and the FixedOrder-Simple method all select attributes independently of each other. Compared with the MaxBranch-Simple and the MaxInfo-Simple methods, these methods do not use information of partial solutions to determine what to choose next. Both the Const-Simple and the FixedOrder-Simple methods select attributes based on a pre-determined ordering. For the Const-Simple method, the attribute ordering is determined by the user based on the user's preference strengths. For the FixedOrder-Simple method, the attribute ordering is determined by the branching factors before attribute selection begins.

The fixed ordering attribute selection method examines the data set and assigns an ordering to the attributes based on the branching factors before the attribute selection begins. For example, Reward is the third most effective differentiating attribute according to the ordering. Because of the dependencies between Reward, DeptCity and ArriveCity, after the attributes DeptCity and ArriveCity are selected, Reward ceases to be an effective differentiating attribute. Such an effect resulting from dependencies between attributes is not captured by pre-determined ordering method.

Compared with the FixedOrder-Simple method, the Const-Simple method generally selects the first two attributes as DeptCity and ArriveCity. After these two attributes are selected, the next attribute that is likely to be selected is DOW (see example constraint strength distributions in section 6.4.3). DOW is not dependent on DeptCity and ArriveCity, so selection of this attribute is still effective in reducing the set of partial solutions to the goal state.

The MaxBranch-Simple method, in contrast to the FixedOrder-Simple method, examines the branching factors of the attributes dynamically. For example, suppose the method selects DeptCity and ArriveCity as the first two attributes. Because of the dependencies between DeptCity, ArriveCity, and Reward, Reward ceases to be an effective differentiating attribute for the resulting solutions. Therefore, it is unlikely to be selected. Since the method compares and selects attributes from solution set given previously selected attributes, the dependencies between attributes are captured implicitly. This explains why while the FixedOrder-Simple method is

significantly worse than the Random-Simple method, but the dynamic version of it – the MaxBranch-Simple method – is significantly more efficient than the Random-Simple method.

Compared with the Random-Simple method, since the MaxBranch-Simple method selects an attribute based on its branching factor and since both DeptCity and ArriveCity are attributes with larger numbers of branches, these two attributes are often selected as the first candidates for question, thus quickly reducing the search space towards the solution set.

The MaxInfo-Simple method, i.e., the maximal information gain heuristic, is similar to the MaxBranch-Simple method in that it also compares and selects attributes from the solution set given previously selected attributes. Instead of counting the possible branches for each attribute, the method compares the attributes using the information gain statistic. Information gain measures reduction in uncertainty for classification problems after an attribute is selected.

It is worthwhile to examine the differences between the maximum branching heuristic and the maximum information gain heuristic. Let us look at two hypothetical attributes A and B for classifying 18 observations of *R*. Attribute A can split the data into 9 categories 1-9, and Attribute B can split the data into 5 categories a-e (Table 12).

Categories of A	A Distribution	Categories of B	B Distribution
1	1	A	3
2	1	B	4
3	1	C	4
4	4	D	4
5	4	E	3
6	4		
7	1		
8	1		
9	1		

Table 12: Classification of 18 hypothetical observations by attributes A and B.

Assuming all the observations in *R* are uniformly distributed, we calculate the entropy of *R* as:

$$Entropy(R) = (-\frac{1}{18}\log_2\frac{1}{18})*18 = \log_2 18 = 4.170$$

$$Gain(R, A) = Entropy(R) - ((-\frac{1}{18}\log_2\frac{1}{18})*6 + (-\frac{4}{18}\log_2\frac{4}{18})*3))$$

$$= 4.170 - (\frac{1}{3}\log_2 18 - \frac{2}{3}\log_2\frac{4}{18}) = 4.170 - (1.390 + 1.447) = 1.333$$

$$Gain(R,B) = Entropy(R) - ((-\frac{3}{18}\log_2\frac{3}{18})*2 + (-\frac{4}{18}\log_2\frac{4}{18})*3))$$

$$= 4.170 - (-\frac{1}{3}\log_2\frac{3}{18} - \frac{2}{3}\log_2\frac{4}{18}) = 4.170 - (2.17 + 1.447) = 1.862$$

As we can see from Table 12, attribute A splits the observations into more branches than attribute B does. The maximal branching heuristic would prefer attribute A over attribute B. The information gain heuristic, however, chooses B over A. In general, the information gain heuristic prefers attributes with uniform probability distributions because the reduction in entropy is usually larger for such attributes, while the maximum branching method only looks at the number of branches without taking into account probability distributions.

Once we are clear about the biases of the maximum branching heuristic (the MaxBranch-Simple method) and the maximum information gain heuristic (the MaxInfo-Simple method), it is easy to see why in the flights reservation domain, the maximum branching heuristic is more effective than the maximum information gain heuristic. Let us first examine the distributions for the attributes Dow, DeptTime, and DeptCity in Figure 17. Note that the distributions for DeptTime is calculated using 1-hour intervals, and that all distributions are ranked in decreasing order of the number of flights.

We can see that the flights are somewhat uniformly distributed based on Dow - the day of week attribute, while the distributions are more skewed for the DeptTime attribute, particularly the DeptCity attribute. Because of the bias toward attributes with uniform distributions, the information gain heuristic selects Dow as the first attribute for disambiguation.

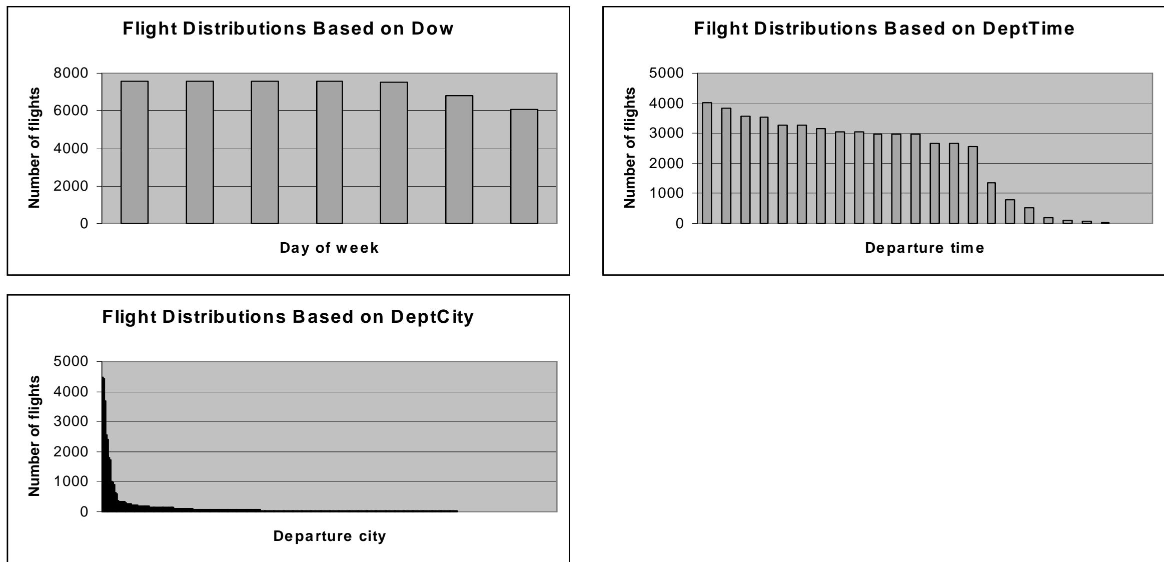


Figure 17: Sorted flight distributions with respect to the attributes Dow, DeptTime, and DeptCity in the Northwest flight database.

Since Dow has a somewhat uniform distribution, selecting this attribute does not give much differentiating power. The next step the method will probably select DeptTime, which is less skewed than DeptCity. As a result, DeptCity will be the last attribute to be selected, even though we know it is an effective attribute for narrowing down the search space. Given the distributions of the attributes in the flight information domain and bias of the information gain heuristic, we conclude that information gain is not an effective measure for this particular domain.

7.1.1.2. Task difficulty and system time for question selection

In the last section, we have observed that the heuristic methods that employ constraint strength information and solution size can significantly reduce the number of questions requested by the system to reach the goal states. This, in turn, improves the interaction efficiency between the system and the user. However, processing of knowledge such as constraint strengths and solution size requires extra computing on the part of the system. In this section, we look at the actual system time in identifying the candidates in the question selection process. All system time is measured in seconds based on Microsoft Access 97 under Microsoft Windows NT 4.0. The hardware has a single Pentium II 233Hz processor with 128 MB RAM and 256MB virtual memory. All the system times reported in this chapter were obtained from this computing environment.

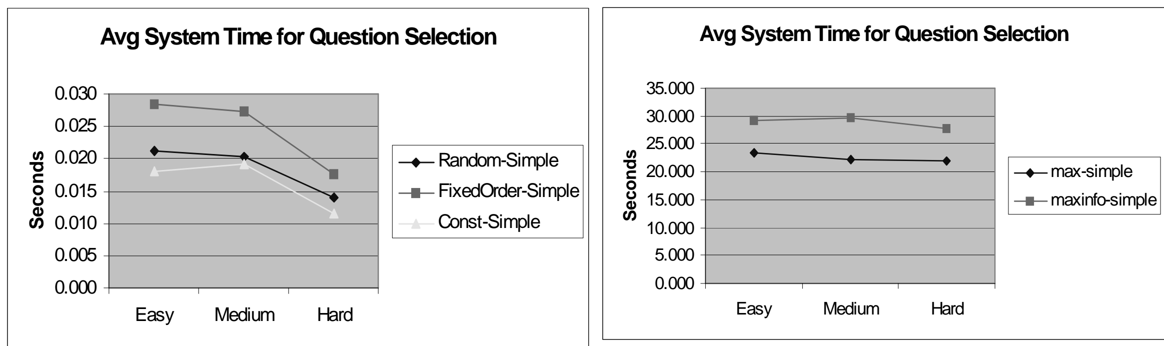


Figure 18: Average system times used for question selection with different question selection methods.

We distinguish the five methods into two groups based on the system time values. The first group consists of the Random-Simple method, the FixedOrder-Simple method, and the Const-Simple method. The average system time for selecting questions to solve a problem for this group ranges between 0 to 0.028 seconds. The second group consists of the MaxBranch-Simple method and the MaxInfo-Simple method, with the average system time for selecting questions to solve a problem ranging from 20.797 to 35.813 seconds. The system time used by the system for question selection in the second group is significantly more than that of the first group.

We compute two-factor analysis of variance for actual system time at three task difficulty levels and for different question selection methods. For the Random-Simple, FixedOrder-Simple and Const-Simple methods, the effects of question selection methods is not significant, but the task difficulty level factor is significant (*p=0.04*). The interaction effect between the two factors is not significant (*p=0.1*).

Method (vs Random-Simple)	*Easy Increase*	*Medium Increase*	*Hard Increase*
Const-Simple	-14.7%	-5.4%	-18.2%
FixedOrder-Simple	33.7%	34.0%	26.2%

Even though question selection methods does not exert a significant difference between system times among the methods, detailed comparison of the performance of the methods at each level does show the general efficiency of the methods. In general, the FixedOrder-Simple method consumes more time than the Random-Simple method, ranging from 26.2% to 34.0% increase. The Const-Simple method, on the other hand, consumes less time than the Random-Simple method, with the decrease ranging from 5.4% to 18.2%. The Const-Simple method is the most efficient in

terms of time because it takes constant time for attribute selection in the question selection process (Chapter 5), and in general requires fewer number of questions to solve a problem (Figure 16). The FixedOrder-Simple method also takes only constant time for attribute selection, but it requires more questions than the Random-Simple method to solve a problem; therefore, the total system time for solving a problem is actually more than that of the Random-Simple method.

For the two methods MaxBranch-Simple and MaxInfo-Simple, we also compute two-factor analysis of variance with the method factor and task difficulty factor. The effects of both the question selection methods and task difficulty levels are significant ($p<0.001$). The interaction effect between the two factors is also significant ($p=0.001$).

Both of these methods consume significantly more system time compared with the Random-Simple method. This is because for each question selection, in order to compute the branching factor and the information gain, the system needs to iterate through the set of partial solutions to compute the subset of solutions for individual attributes. This takes $O(nR)$ time, where R is the number records in the partial solutions. The MaxInfo-Simple method requires additional time for calculating the information gain. These experimental results are consistent with the complexity analysis in section 5.1.

In Figure 18, the trend for the performance lines of the methods are decreasing from Easy to Medium, and to Hard problems. Such trend lines of average system time correspond to the lines in Figure 16, where the lines of the average number of questions selected are presented. The correspondence suggests that as task difficulty increases, fewer questions are required, thus less system time. The only exception in both figures is with the MaxInfo-Simple method at the Medium level and with the Const-Simple method at the Medium level.

The number of questions and the system time required capture different aspects of question selection efficiency. Requesting answers from the user requires interaction between the system and the user, thus affecting dialogue efficiency. System time captures the time required for selecting these questions. System time affects the speed of the system in coming up with good questions during system and user dialogue. Compared with the Random-Simple (i.e., the random attribute selection) method, the Const-Simple method and the MaxBranch-Simple method requires fewer numbers of questions, thus having better dialogue efficiency. The MaxBranch-Simple method reduces the number of questions by 11.% to 36.6%, while the Const-Simple method reduces the number of questions by 8.8% to 29.9% (Table 11). Taking into account of system time, however,

the MaxBranch-Simple method is very expensive compared with the random selection method (1098, 1088 and 1581 more expensive for the Easy, Medium, and Hard problems, respectively), while the Const-Simple is about 5.4% to 18.2% less expensive than the Random-Simple method. Therefore, when the two question selection efficiency measures are considered together, the Const-Simple method is overall the most efficient method among all methods.

7.1.2. Task difficulty and task success

In this section, we look at how the five different heuristic methods perform in terms of task success. Again, we examine these questions:

- For a particular question selection method, what is the effect of different task difficulty levels on task success?
- For problems of a particular task difficulty level, which method produces the best task success rate?
- Is there a method that outperforms other methods in terms of task success across all task levels?

Figure 19 summarizes the average kappa score per problem for all the methods at Easy, Medium, and Hard levels. Table 13 is a two-way analysis for the task success scores at three task difficulty levels and with different question selection methods. First, we observe that both the task difficulty factor and the question selection method factor have significant effects. On average, for all the methods, the kappa scores decrease as task difficulty increase ($p=0.001$); the Const-Simple method and the FixedOrder-Simple method are better than the MaxBranch-Simple method or the MaxInfo-Simple method ($p<0.001$). The interaction effect between the task difficulty factor and the method factor, however, is not significant.

Looking at the pairwise comparisons in Table 14, we observe that the drop in task success rate ranges from 0.5% to 5.0% between the Easy problems and the Medium problems, and from 2.7% to 6.6% between the Medium problems and the Hard problems. While the drops in task success rates are observable, pairwise two-sample *t*-test results show that when comparing the kappa scores between Easy problems and Medium problems, the differences are not significant for all the methods. When comparing the kappa scores between Medium problems and Hard problems, the differences are again not significant for all methods except for the MaxInfo-Simple method. When comparing the kappa scores for the easy problems and the hard problems, we observe that the difference in task success scores are significant for all the methods with the exception of the

Random-Simple method where the difference is not significant. Overall, the test results suggest that the task difficulty levels defined in this work do incrementally increase task difficulty as far as task success is considered. However, the incremental change in task difficulty between levels does not exert significant change in the task success measure (i.e., Easy vs Medium and Medium vs Hard), but the differences do accumulate as task difficulty increases, resulting in significant differences (in the case of Easy vs Hard problems).

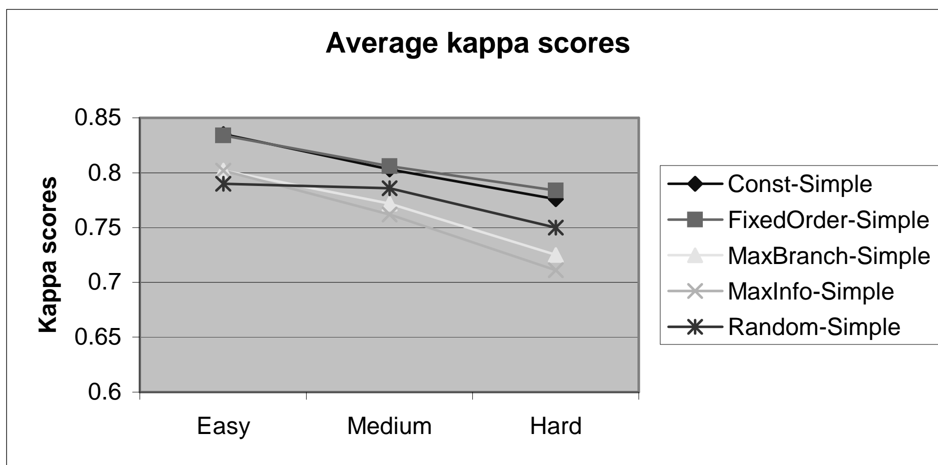


Figure 19 : Average kappa scores across task difficulty levels with different question selection methods**.**

Figure 19 shows that compared with the Random-Simple method, the Const-Simple and the FixedOrder-Simple methods achieved higher average task success rates for all the problems (a combination of Easy, Medium, and Hard problems), while the MaxInfo-Simple and the MaxBranch-Simple methods demonstrated lower success rates for Medium and Hard problems. ANOVA computation of the task success scores with different question selection methods over all the problems shows that the kappa scores are significantly different ($p<0.001$). In particular, pairwise *t*-tests in Table 15 show that both the Const-Simple method and the FixedOrder-Simple method outperform the Random-Simple method significantly ($p<0.001$), while the difference between the Const-Simple method and the FixedOrder-Simple method is not significant. Compared with the Random-Simple method, the MaxBranch-Simple method and the MaxInfo-Simple method are not significantly different. The MaxBranch-Simple method and the MaxInfo-Simple method produce similar kappa scores, with the differences not significant.

Source	*F-value*	*P-value*	*F crit*
Method	4.938	0.001	2.382
Task Difficulty	16.326	<0.001	3.006
Method X Task Difficulty	0.391	0.925	1.949

Table 13: Two-way analysis of variance for the kappa scores at different task difficulty levels for the question selection methods.

Methods	*Easy vs Medium*		*Medium vs Hard*		*Easy vs Hard*	
	%decrease	*p-value*	%decrease	*p-value*	%decrease	*p-value*
MaxBranch-Simple	3.8%	0.136	6.1%	0.058	9.6%	0.004
Const-Simple	3.8%	0.053	3.3%	0.115	7.0%	0.002
FixedOrder-Simple	3.4%	0.060	2.7%	0.125	6.0%	0.004
MaxInfo-Simple	5.0%	0.060	6.6%	0.045	11.3%	0.002
Random-Simple	0.5%	0.440	4.6%	0.073	5.1%	0.071

Table 14 : Pairwise comparisons of kappa scores between task difficulty levels for different question selection methods.

Table 16 summarizes the ANOVA results for all the methods at the three task difficulty levels for the kappa statistic. Compared to the .05 critical value, only the results of the Hard problems demonstrate significant difference. We conclude from this analysis that even though in general we observed the Const-Simple and the FixedOrder-Simple methods achieve higher task success rate, such difference is not significant for the Easy and Medium problems. However, such difference is significant for the Hard problems.

It is straightforward to see that the heuristic method using constraint hierarchy (i.e., the Const-Simple method) generally performs well compared with other methods. In the computation of the weighted kappa scores, constraints with higher weights contribute more to the final kappa score. Thus such a measure favors the Const-Simple method, which attempts to satisfy first the constraints with higher weights during problem solving.

It is worthwhile to examine why the fixed ordering method (i.e., the FixedOrder-Simple method) demonstrated similar or even better performance in task success scores compared with the Const-Simple method. Two features of the FixedOrder-Simple method account for such behavior. First, it turns out that the fixed ordering used in the FixedOrder-Simple method corresponds closely to the constraint ordering we have generated using the constraint strength distributions. Recall that when we generate a test problem with constraint hierarchy, we assign the constraint strengths to the attributes by randomly selecting a constraint strength value from their respective distributions. The sample distributions in the flight domain presented in section 6.4.3 shows that, even though the constraints for each attribute can take values from required to weak, the probability distributions of these attributes would in general give higher weights to DeptCity and ArriveCity, but lower weights to DeptTime or ArriveTime. Such an ordering corresponds roughly to the preference ordering used by the fixed ordering method. Recall that the fixed ordering of attributes we have used for the FixedOrder-Simple method starts with DeptCity and ArriveCity before the attributes DeptTime or ArriveTime.

Method Pairs	*P-values*
Const-Simple vs FixedOrder-Simple	0.135
Const-Simple vs Random-Simple	<0.001
FixedOrder-Simple vs Random-Simple	<0.001
MaxBranch-Simple vs MaxInfo-Simple	0.218
MaxBranch-Simple vs Random-Simple	0.219
MaxInfo-Simple vs Random-Simple	0.075

Table 15: Pairwise *t*-test results for kappa scores between different question selection methods.

Task Difficulty Level	*F-value*	*P-value*	*F crit*
Easy	1.485	0.207	2.402
Medium	1.307	0.267	2.402
Hard	2.727	0.030	2.402

Table 16: *F*-values and *p*-values across task difficulty levels for the kappa scores.

Second, with the FixedOrder-Simple method, whenever the DeptCity and ArriveCity attributes are selected, the Reward attribute will be satisfied since it is dependent on the DeptCity and ArriveCity. Thus, it will guarantee a match of the Reward attribute regardless of whether the attribute is discussed in the dialogue or not. With the constraint preference method, however, the

DeptCity and ArriveCity can get lower preference strengths compared with other attributes. When the preference strengths for these two attributes are lower than those of the other attributes, satisfaction of other attributes will come first. Therefore, the dependency relationship between Reward and DeptCity and ArriveCity are less likely to contribute to the matching. In fact, this is exactly what happens with the cases when the FixedOrder-Simple method produces higher task success score than the Const-Simple method.

7.1.3. Task difficulty and relaxation efficiency

To examine the effect of task difficulty on identifying relaxation candidates, we investigate three questions:

- For a particular relaxation method, what is the effect of different task difficulty levels?
- For problems of a particular task difficulty level, what is the most efficient method in identifying relaxation candidates?
- Is there a method that outperforms other methods in terms of identifying relaxation candidates across all task levels?

We compare three methods – Random-Simple, Random-Const, and Random-MinSize – to examine the effect on task difficulty on relaxation efficiency. In these experiments, we vary the relaxation candidate selection methods, while keeping the question selection method constant as the random attribute selection method. The Random-Simple method, which uses chronological backtracking as the method for identifying relaxation candidates, is regarded as the baseline. The other two methods, the Random-Const and the Random-MinSize methods, use walkabout strengths and solution size as additional knowledge in the solution synthesis graph (Chapter 5). We discuss results using the two relaxation efficiency measures – the number of relaxation candidates required and the actual time required for identifying the relaxation candidates – respectively in the following subsections.

7.1.3.1. Task difficulty and the number of relaxation candidates suggested by the system

We first look at the effect of task difficulty on the number of relaxation candidates that the system needs to elicit. The maximum number of relaxation candidates that can be requested by the system is bounded by 6, which is the maximum number of attributes that can be included in a problem.

Table 17 is a two-way analysis of the number of relaxation candidates with three task difficulty levels and different heuristic methods on relaxation efficiency. First, note that the task difficulty factor does not have significant effects at the conventional .05 level. This can be seen from Figure 20, which presents the average number of relaxation requests by the system to resolve the over-constrained problems. We observe that in Figure 20, the lines are relatively flat for each method for all Easy, Medium, and Hard problems. For each method, we further compute single-factor ANOVA of the task difficulty factor. The ANOVA for all the three methods show that the differences in performance are not significant at the conventional .05 level. Thus, we conclude that for these methods, task difficulty does not exert significant effect on the efficiency of the average number of relaxation requests by the system.

Source	*F-value*	*P-value*	*F crit*
Method	3.396	0.034	3.013
Task Difficulty	0.008	0.992	3.013
Method X Task Difficulty	0.845	0.497	2.389

Table 17: Two-factor ANOVA of the number of relaxation requests with different relaxation selection methods at different task difficulty levels.

From Table 17, we also observe that relaxation method is a significant factor (*p=0.034*). From Figure 20, we observe that when the question selection method is fixed as the random selection method, the Random-Const method is the most efficient method by requesting the fewest number of relaxation candidates for all Easy, Medium, and Hard problems. On the other hand, the Random-MinSize method is not as efficient as the baseline Random-Simple method. When we compare the Random-Const method with the baseline Random-Simple method, we observe that two lines are very close. Paired two-sample t-tests show that the differences between the two lines are not significant. However, when we compare the Random-MinSize method with the baseline Random-Simple method, the difference is significant (*p=0.005*). This difference results mainly from the difference in performance with the Easy problems.

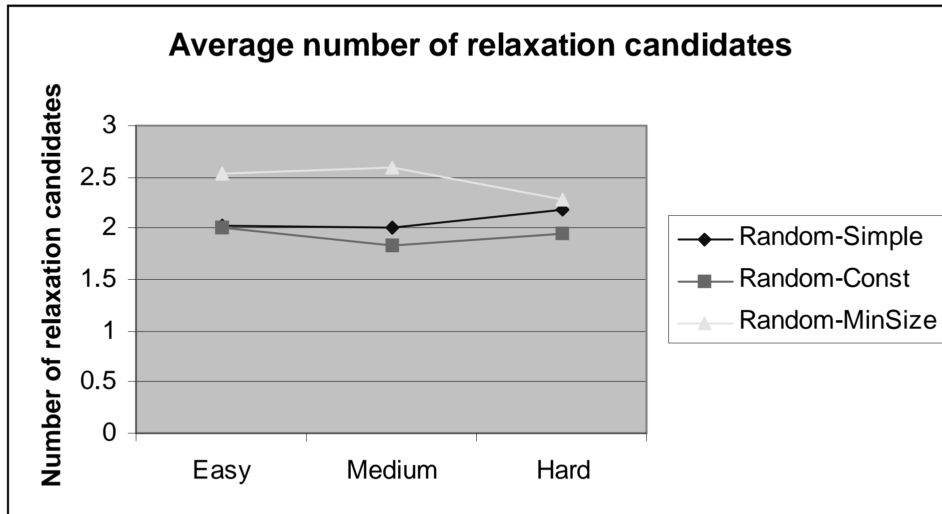


Figure 20: Average numbers of relaxation requests with different relaxation selection methods. The question selection method is fixed as the random selection method.

The interaction between task difficulty factor and the method factor, however, is not significant at the conventional .05 level.

Now, to what extent is the relaxation method dependent on the question selection method? To answer this question, we examine two aspects of the dependency. The first aspect is whether, when the question selection method is fixed, the observations with the random attribute selection hold true for other attribute selection methods during the question selection process. To evaluate this, we conducted two additional sets of experiments. In one setting, we compare the performance between the Const-Simple method and the Const-Const method, where the question selection method is fixed as the heuristic method based on constraint hierarchy, and the relaxation methods are simple backtracking and constraint-hierarchy based, respectively. In another setting, we compare the performance between the MaxBranch-Simple method and the MaxBranch-MinSize method, where the questions selection method is fixed as the maximum branching heuristic, and the relaxation methods are simple backtracking and minimum solution size based, respectively.

Figure 21a presents the performance results for the Const-Simple method and the Const-Const method. We see that the two lines are very close together, with Const-Simple method outperforms the Const-Const method for the Easy and Hard problems, but it underperforms the Const-Const method for the Medium problems. Single-factor ANOVA (with method as the factor) over all the problems, however, shows that the difference between the two lines are not significant.

Figure 21b presents the performance results for the MaxBranch-Simple method and the MaxBranch-MinSize method. The two lines are observably in parallel, but with the MaxBranch-

Simple method much more efficient, requiring an average of 31.4% fewer relaxation requests to solve the same set of problems when compared with those of the MaxBranch-MinSize method. ANOVA result, however, shows that the difference between the two lines is not significant.

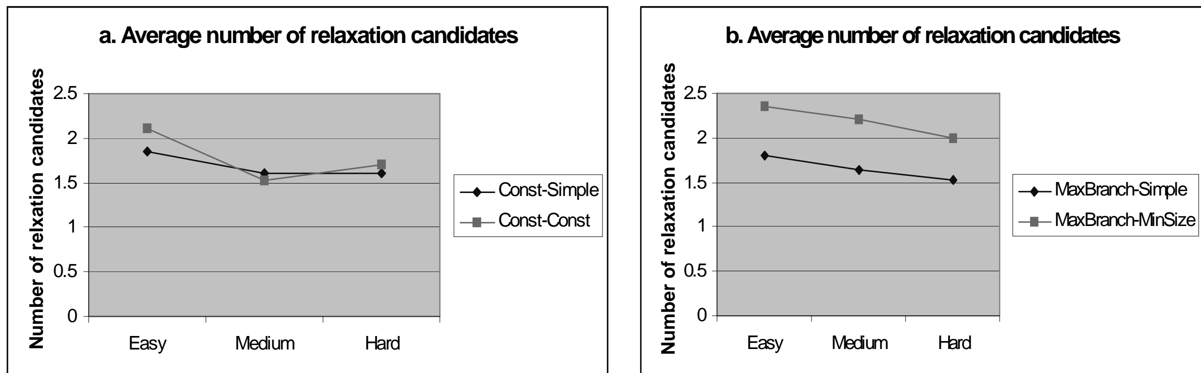


Figure 21: Average numbers of relaxation requests with different selection methods. The question selection method is fixed as the constraint hierarchy based method in (a) and as the maximum branching method in (b).

The observations from Figure 20 and Figure 21 suggest that when the question selection method is fixed, the simple chronological backtracking method and the method using constraint hierarchy are equivalent in identifying relaxation candidates. Compared with the simple chronological backtracking method, the method using the minimum solution size heuristic is equivalent or worse compared with the simple chronological method. These observations hold true regardless of the question selection method used.

The second aspect to our questions is how, when the relaxation candidate selection method is fixed, the question selection methods affect relaxation efficiency. Since difference question selection methods select different attributes during the problem solving process, the solution structures are different, which could subsequently affect the behavior of the relaxation selection method.

Figure 22 presents three group comparisons with the relaxation method fixed as a certain method. The relaxation methods are set as the chronological relaxation method, the constraint hierarchy based method and the minimum solution size based method in figure a, b, and c, respectively. We make the following observations of the comparisons:

First, when the relaxation method is fixed, the more efficient question selection methods result in more efficient relaxation. As we can see from Figure 22, the more efficient question selection

methods – the constraint hierarchy based method and the maximum branching based method – generally outperform the random attribute selection method. The only exception to this general observation is with the Const-Const method at the Easy task difficulty level, when it actually underperforms the Random-Const method. Still, the general observation suggests that partial solution structure influences relaxation efficiency. The most efficient question selection methods require fewer numbers of questions for solving a problem, thus having a simpler solution structure, which in turn makes relaxation efficient. Analysis of variance of these three groups shows that differences between the more efficient question selection method and the random selection method are significant for all three groups of comparisons ($p \leq 0.001$).

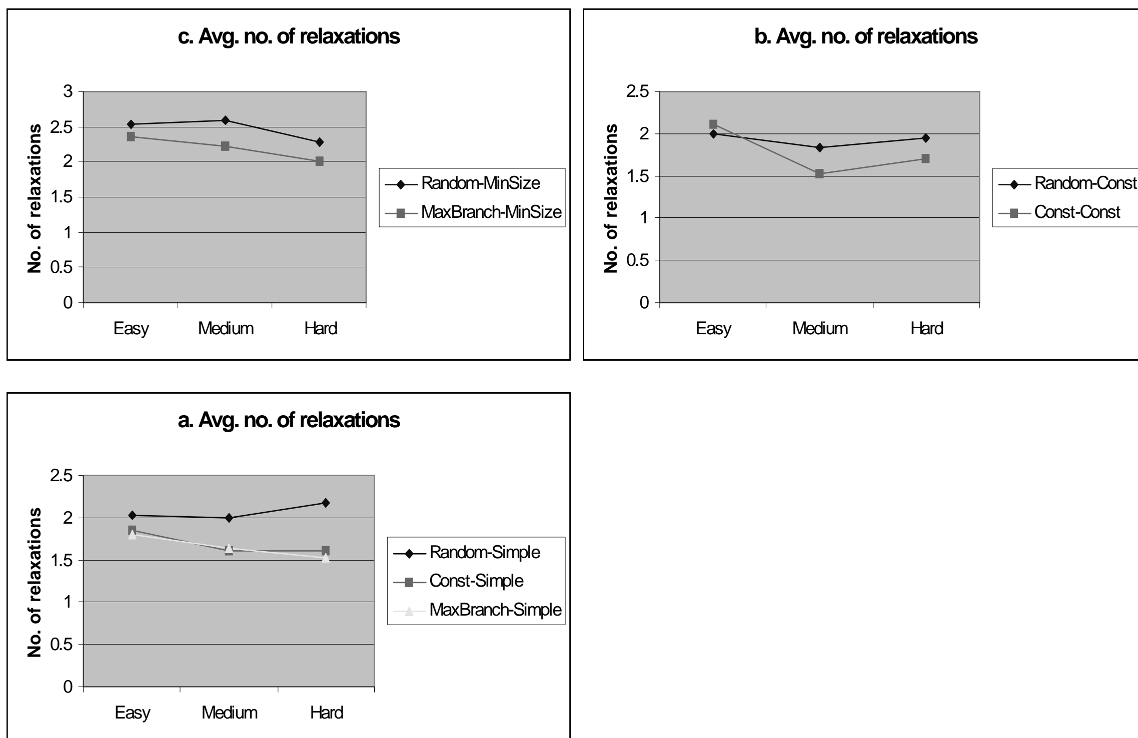


Figure 22: Average numbers of relaxation requests with the relaxation methods fixed as the simple backtracking method in (a), as the constraint hierarchy based backtracking in (b), and the minimum solution size based backtracking in (c).

Second, analysis of variance results also shows significant differences at different task difficulty levels (*p=0.006* for group a, *p=0.05* for group b, and *p=0.005* for group c). While we have observed that a decreasing number of questions is required as the task difficulty level increases for most question selection method (section 7.1.1), such trend does not exist as far as relaxation is concerned. Instead, the numbers of relaxation candidates increase significantly for some methods (e.g., Random-Simple, Random-Const and Const-Const) at the Hard task difficulty levels.

In summary, empirical results have demonstrated that the constraint hierarchy based relaxation method is the most efficient method for identifying relaxation candidates. In addition, the efficiency of relaxation depends heavily upon the question selection method, which determines the structure of the partial solution space in which a relaxation method operates.

7.1.3.2. Task difficulty and relaxation selection time

In the last section, we have observed that the heuristic method that employs constraint strength information generally reduces the number of relaxation candidates required in over-constrained situations. This, in turn, improves the interaction efficiency between the system and the user. In this section, we look at the actual system time for identifying the candidates in the relaxation process.

We compare the system times for methods with the question selection method fixed as the random selection: the Random-Simple method, the Random-Const method and the Random-Size method.

Figure 23 shows that the average system time for selecting relaxation candidates for solving an over-constrained problem ranges between 0.319 to 2.051 seconds. The system times used by the Random-Simple method and the Random-Const method are very similar, while the system time used by the Random-Size method is significantly more than the former two. Analysis of variance shows that the difference is significant ($p \geq 0.001$). The observation here corresponds to that in Figure 20, where the numbers of relaxation candidates are compared.

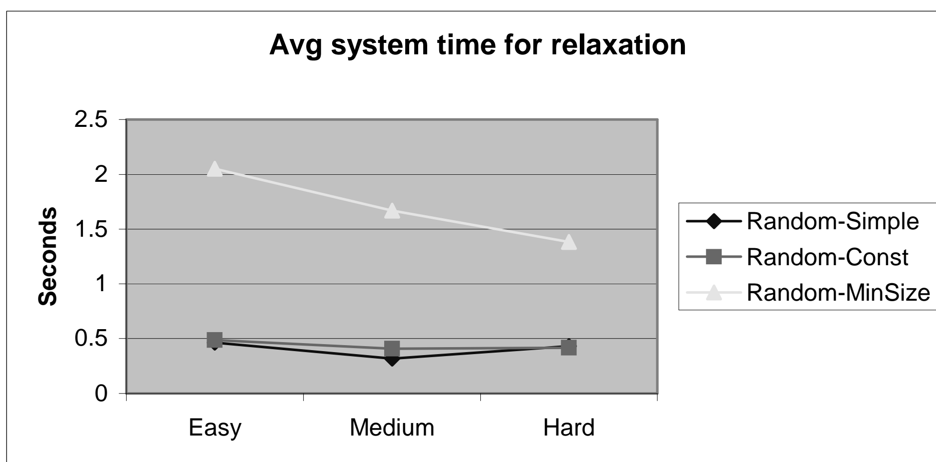


Figure 23: Average system times for selecting relaxation requests with the question selection fixed as the random selection method.

Analysis of variance does not show that task difficulty has significant effect on the average system time for relaxation, and that the interaction between the method factor and the task difficulty factor are not significant.

7.1.4. Task difficulty: summary

To summarize, we have observed that, compared with the baseline random question selection method (Random-Simple), the question selection heuristics based on constraint hierarchy (Const-Simple) and the knowledge of the partial solution size (MaxBranch-Simple) significantly reduce the average number of questions that the system needs to elicit from the user in under-constrained situations across all task difficulty levels. This means in human-computer dialogues, these two heuristics will reduce the number of dialogue turns between the user and the system. In terms of system time required for identifying questions, the Const-Simple heuristic takes less time compared with the Random-Simple baseline across all task difficulty levels, while the MaxBranch-Simple heuristic takes significantly more time compared with the Random-Simple baseline. Therefore, when we take into account both the number of questions required and the time required to obtain them, the Const-Simple heuristic is the most efficient method for question selection.

In terms of task success as measured by the kappa statistic, we observe that the kappa scores decrease as task difficulty increases for all the methods. The Const-Simple heuristic and the FixedOrder-Simple heuristic outperform the Random-Simple method significantly, and such difference results primarily from Hard problems. The difference between the Const-Simple heuristic and the FixedOrder-Simple heuristic is not significant. The Const-Simple heuristic produces higher task success scores, due to the bias we build into the weighted kappa statistic, in which the attributes with higher weight contribute more to the final score. The FixedOrder-Simple heuristic produces comparable kappa scores compared with the Const-Simple heuristic due to the fact that the fixed attribute ordering obtained from the airline database used for the experiments correlates closely to the attribute ordering based on the constraint strength distributions of the airline domain.

In terms of the numbers of relaxation candidate for over-constrained situations, we observe that when the question selection method is fixed, there is no significant difference between the efficiency of the relaxation selection methods across the different task difficulty levels (except between the Random-Simple method and the Random-MinSize method for Easy problems). When the relaxation selection method is fixed, we observe that, in general, the more efficient the question

selection method is, the more efficient the relaxation selection. The constraint hierarchy based method and the maximum branching based method generally outperforms the random attribute selection method. These two aspects of analysis suggest that when heuristics of question selection and relaxation selection are combined together, the question selection heuristics dominate the efficiency performance in resolving over-constrained problems.

7.2. Result Analysis: The Effect of Goal-State Size

The size of the goal states determines how many solutions the system is to present to the user. Since human-computer dialogue is a collaboration process, the size of the goal states affects the computation efforts of both parties. For example, when a larger goal-state size is sufficient, the system generally does not need to request many clarification questions in order to locate the solutions to satisfy the user's need. Rather, the users will be able to work with a larger set of solutions and choose the solution that satisfies their needs better. On the other hand, if the goal-state size is small, then the system needs to ask many clarification questions in order to get to the exact solutions that satisfy the user's information needs. Communication medium is one factor that influences the size of the goal states. For example, in speech based dialogue systems, due to the cognitive and memory limitations of the user (Walker, 1993), the set of solutions that can present to a user is usually a smaller number (e.g., a size of five solutions is used in many systems). In such cases, it is desirable that the smaller number of solutions consists of the solutions that satisfy the user's needs most. With a screen-based interface, however, the user can browse more solutions. For example, the user may be able to browse up to twenty solutions within one computer screen. Again, in such cases, it is desirable that the twenty solutions are the top twenty that best satisfy the user's information needs. Therefore, determination of a reasonable size of the goal state is dependent upon the media mode in information access applications, which in turn influences the efficiency of human-computer dialogues.

7.2.1. Goal-state sizes and question selection efficiency

In this section, we examine the five methods – Random-Simple, FixedOrder-Simple, MaxBranch-Simple, Const-Simple, and MaxInfo-Simple – with respect to different goal-state sizes. In these methods, the question selection methods are varied, but the relaxation candidate selection method is fixed as the chronological backtracking method. We use three goal-state sizes (*G=1, 5*, and *20*). Again, the attribute set consists of attributes for DeptCity, ArriveCity, DeptTime, ArriveTime,

Reward, and DOW; the time interval chosen for DeptTime and ArriveTime is one hour; and the fixed ordering method uses the attribute ordering: $DeptCity > ArriveCity > \mathrm{Re}\,ward > DeptTime > ArriveTime > DOW$. In the following, we look at the effect of goal-state sizes on question selection efficiency, task success, and relaxation efficiency. Since we have observed that task difficulty levels affect the performance statistics of the methods (section 7.1.1), we investigate the effect of goal-state sizes for all task difficulty levels.

To examine the effect of goal-state size on question selection efficiency, we investigate three questions:

- For a particular question selection method, what is the effect of different goal-state size?
- For a particular goal-state size, what is the most efficient method in question selection?
- Is there a method that outperforms other methods in terms of question selection across all goal-state sizes?

7.2.1.1. Goal-state sizes and number of questions for the Easy task difficulty problems

Table 18 is a two-way analysis of the effects of different goal sizes and heuristic methods on the number of questions for Easy problems and the interaction between the two factors. First, note that both the goal size and the methods have significant effects ($p<0.001$). In addition, the two factors interact; the interaction is significant ($p=0.004$).

Source	*F-value*	*P-value*	*F crit*
Method	55.080	<0.001	2.382
Goal-state Size	235.843	<0.001	3.006
Method X Goal-state Size	2.885	0.004	1.949

Table 18: Two-factor ANOVA of the number of questions with different question seleciton methods at different goal-state sizes for the Easy task difficulty problems.

Figure 24 presents the average number of clarification questions requested by the system to solve the problems. We observe that in Figure 24, the lines are decreasing as the goal-state size increases, i.e., the number of candidates required for solving the key problem is in reverse proportion with the size of the pre-determined solution states. That is, the larger the solution size, the fewer the number of candidates required. In particular, when we look at the numbers of candidate required for the question selection, pair-wise comparison of two goal-state sizes shows that the difference is significant for almost all methods (Table 19). For example, for the MaxBranch-Simple method,

fewer numbers of questions are required to solve the *G=20* problems than the *G=5* problems (*t=5.416*, *p<0.001,* one-tail), and fewer numbers of questions are required to solve the *G=5* problems than the *G=1* problems (*t=2.564*, *p<0.001,* one-tail). The only exception is with the FixedOrder-Simple method when comparing goal-state sizes *G=1* vs *G=5*, where the difference is not significant.

Methods	G=1 vs G=5		G=5 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value
Random-Simple	3.925	0.001	5.286	<0.001
FixedOrder-Simple	1.671	0.161	6.080	<0.001
Const-Simple	4.162	<0.001	8.301	<0.001
MaxBranch-Simple	2.564	0.006	5.416	<0.001
MaxInfo-Simple	5.028	<0.001	5.210	<0.001

Table 19: Pairwise comparisons of the numbers of questions at different goal-state sizes for the Easy task difficulty problems.

When comparing the methods across the goal-state sizes, we observe that across all the goal-state sizes (*G=1*, *5*, and *20*), the Random-Simple baseline is significantly more efficient than the FixedOrder-Simple method with respect to the number of candidates selected (Table 20) at across all goal-state sizes (*p<=0.001*). Compared with the Random-Simple method, the MaxInfo-Simple method is slight more efficient at *G=1* and *G=5* goal-state sizes, but such efficiency improvement is not significant. At G=20, the Random-Simple is significantly better than the MaxInfo-Simple method (*p=0.018*). Both the Const-Simple and the MaxBranch-Simple methods are significantly more efficient across all goal-state sizes, with *p* values ranging from 0.015 to 0.001 across different goal-state sizes. When the Const-Simple method is compared with the MaxBranch-Simple method, the MaxBranch-Simple method demonstrates improved efficiency at *G=1* and *G=5* states, while the Const-Simple performs better at *G=20* state. However, these differences are not statistically significant.

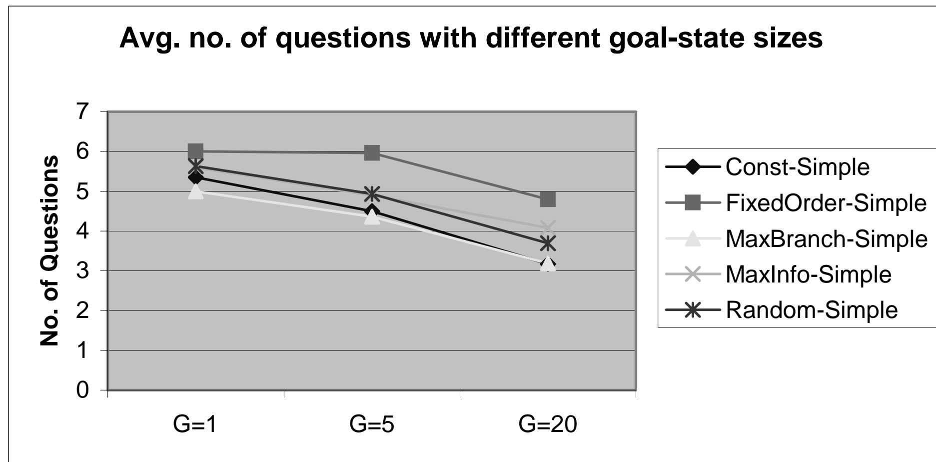


Figure 24: Average numbers of question elicited by the system at different goal-state sizes for the Easy task difficulty problems.

	G=1		G=5		G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple vs FixedOrder-Simple	-3.861	0.001	-6.556	<0.001	-4.807	<0.001
Random-Simple vs MaxInfo-Simple	0.281	0.390	0.189	0.425	-2.134	0.018
Random-Simple vs Const-Simple	2.210	0.014	2.221	0.015	2.795	0.004
Random-Simple vs MaxBranch-Simple	3.332	0.001	3.043	0.002	2.438	0.009
Const-Simple vs MaxBranch-Simple	1.618	0.056	0.841	0.201	0.092	0.463

Table 20: Pairwise comparisons of the numbers of questions between different question selection methods at different goal-state sizes for the Easy task difficulty problems.

7.2.1.2. Goal-state sizes and number of questions for the Medium task difficulty problems

Table 21 is a two-way analysis of the effects of different goal sizes and heuristic methods on the number of questions for Medium problems and the interaction between the two factors. We observe that both the goal-state size and the methods have significant effects ($p<0.001$), and that the two factors interact in a significant way ($p<0.001$).

Figure 25 presents the average number of clarification questions requested by the system to solve the Medium level problems. Similar to what we have observed for Easy problems, we observe that

in this figure that the lines are decreasing as the goal-state size increases, i.e., the number of candidates required for solving the key problem is in reverse proportion with the size of the pre-determined solution states.

In particular, when we look at the numbers of candidate required for the question selection, pair-wise comparison of two goal-state sizes shows that the differences are significant for almost all methods (Table 22). The only exception is with the fixed ordering method when comparing goal-state sizes *G=1* vs *G=5*. While the average number of questions for *G=5* is smaller than that of *G=1*, the difference is not significant.

Source	*F-value*	*P-value*	*F crit*
Method	112.817	<0.001	2.382
Goal-state Size	280.600	<0.001	3.006
Method X Goal-state Size	13.080	<0.001	1.949

Table 21: Two-factor ANOVA of the number of questions with different question selection methods at different goal-state sizes for the Medium task difficulty problems.

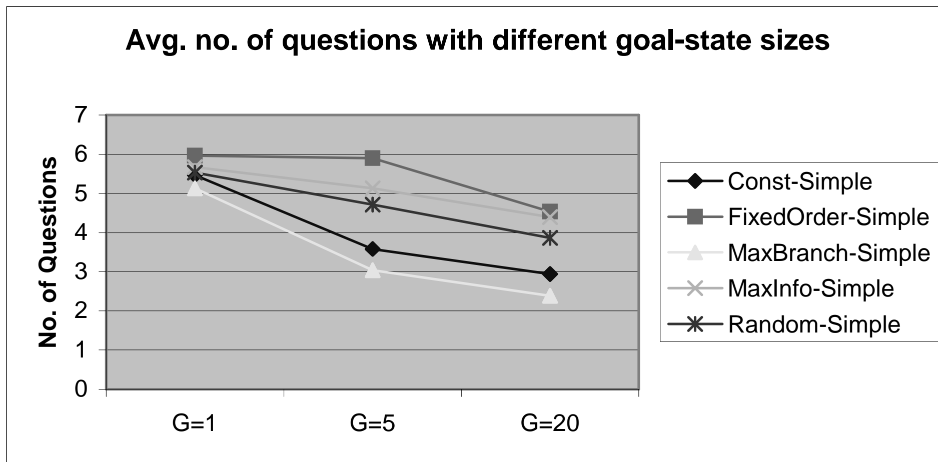


Figure 25: Average numbers of questions elicited by the system at different goal-state sizes for the Medium task difficulty problems.

When comparing the methods across the different goal-state sizes (Table 23), we observe that across all the goal-state sizes (*G=1*, *5*, and *20*), the Random-Simple baseline are more efficient than the FixedOrder-Simple method and the MaxInfo-Simple method with respect to the number of candidates selected. It is significantly more efficient than the FixedOrder-Simple method at all goal-state sizes. The Random-Simple method is significantly more efficient than the MaxInfo-

Simple method at goal-state sizes *G=5* and *G=20*, but the difference is not significant at *G=1*. Compared with the Random-Simple baseline, both the Const-Simple and the MaxBranch-Simple methods are significantly more efficient at all goal-state sizes, with *p* values ranging from 0.005 to less than 0.001 across different goal states. Between the Const-Simple method and the MaxBranch-Simple method, the MaxBranch-Simple method is more efficient at all states, with the differences statistically significant.

Methods	G=1 vs G=5		G=5 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value
Random-Simple	5.039	<0.001	4.375	<0.001
FixedOrder-Simple	1	0.161	5.900	<0.001
Const-Simple	11.546	<0.001	5.603	<0.001
MaxBranch-Simple	10.984	<0.001	5.603	<0.001
MaxInfo-Simple	6.355	<0.001	7.299	<0.001

Table 22: Pairwise comparisons of the numbers of questions at different goal-state sizes for the Medium task difficulty problems.

	G=1		G=5		G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple vs FixedOrder-Simple	-3.550	<0.001	-8.603	<0.001	-2.838	0.003
Random-Simple vs MaxInfo-Simple	-1.158	0.126	-2.833	0.003	-3.223	0.001
Random-Simple vs Const-Simple	0.475	0.318	6.779	<0.001	5.033	<0.001
Random-Simple vs MaxBranch-Simple	2.921	0.002	8.959	<0.001	8.782	<0.001
Const-Simple vs MaxBranch-Simple	2.666	0.005	3.080	0.002	3.782	<0.001

Table 23: Pairwise comparisons of the numbers of questions between different question selection methods at different goal-state sizes for the Medium task difficulty problems.

7.2.1.3. Goal-state sizes and number of questions for the Hard task difficulty problems

Table 24 is a two-way analysis of the effects of different goal sizes and heuristic methods on the number of questions for hard problems and the interaction between the two factors. We observe

that both the goal size and the methods have significant effects (*p<0.001*), and that the two factors interact in a significant way (*p<0.001*).

Figure 26 presents the average number of clarification questions requested by the system to solve the Hard level problems. Again, similar to what we have observed for Easy problems and Medium problems, the lines are decreasing as the goal-state size increases, i.e., the number of candidates required for solving the key problem is in reverse proportion with the size of the pre-determined solution states.

In particular, when we look at the numbers of candidate required for the question selection, pair-wise comparison of two goal-state sizes shows that the differences are significant for almost all methods (Table 25). The only exception is with the fixed ordering method when comparing goal-state sizes *G=5* vs *G=20*. While the average number of questions for *G=20* is smaller than that of *G=5*, the difference is not significant.

Source	*F-value*	*P-value*	*F crit*
Method	88.544	<0.001	2.382
Goal-state Size	234.016	<0.001	3.006
Method X Goal-state Size	9.711	<0.001	1.949

Table 24: Two-factor ANOVA of the number of questions with different question selection methods at different goal-state sizes for the Hard task difficulty problems.

Methods	G=1 vs G=5		G=5 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value
Random-Simple	5.192	<0.001	5.742	<0.001
FixedOrder-Simple	3.266	0.001	1.035	0.152
Const-Simple	9.337	<0.001	7.618	<0.001
MaxBranch-Simple	10.603	<0.001	8.671	<0.001
MaxInfo-Simple	6.404	<0.001	7.400	<0.001

Table 25: Pairwise comparisons of the numbers of questions at different goal-state sizes for the Hard task difficulty problems.

When comparing the methods at different goal-state sizes, we observe that across all the goal-state sizes (*G=1*, *5*, and *20*), the Random-Simple baseline is more efficient than the FixedOrder-Simple method at all states. Compared with the Random-Simple baseline, the MaxInfo-Simple method is more efficient at *G=1*, but are less efficient at *G=5* and *G=20*. The difference is significant at *G=20* but not at *G=1* or *G=5*. Compared with the Random-Simple baseline, the MaxBranch-Simple and the Const-Simple method are more efficient across all goal-state sizes; the improvement is significant except with goal-state size *G=1* for the MaxBranch-Simple method. Between the Const-Simple method and the MaxBranch-Simple method, the MaxBranch-Simple

method demonstrated more efficiency at *G=5* and *G=20*, but less efficient at *G=1*. But differences between these two methods are not statistically significant.

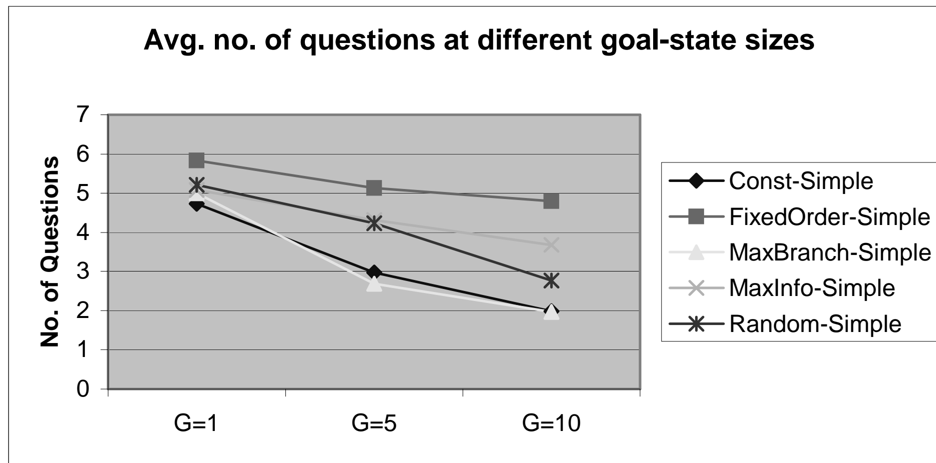


Figure 26: Average numbers of questions elicited by the system at different goal-state sizes for the Hard task difficulty problems.

	G=1		G=5		G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple vs FixedOrder-Simple	-4.729	<0.001	-3.632	<0.001	-7.283	<0.001
Random-Simple vs MaxInfo-Simple	0.823	0.207	-0.382	0.352	-5.008	0.001
Random-Simple vs Const-Simple	3.366	0.001	6.200	<0.001	4.656	<0.001
Random-Simple vs MaxBranch-Simple	0.907	0.184	7.271	<0.001	4.181	<0.001
Const-Simple vs MaxBranch-Simple	-1.117	0.134	1.606	0.057	0.121	0.452

Table 26: Pairwise comparisons of the numbers of questions between different question selection methods at different goal-state sizes for the Hard task difficulty problems.

7.2.1.4. **Goal-state sizes and question selection time for the Easy task difficulty problems**

Figure 27 presents the average system times used for selecting the information request questions at different goal-state sizes for Easy problems. We present the system time in two charts, since the MaxBranch-Simple and the MaxInfo-Simple methods require significantly more time (between 20 and 30 seconds) than the Random-Simple, Const-Simple and the FixedOrder-Simple methods (between 0.010 and 0.028 seconds). Two-factor analysis of variance among the Random-Simple, FixedOrder-Simple and Const-Simple methods shows that both the goal-state factor and the method factor are significant (*p=0.008* for the method factor, and *p=0.01* for the goal-state size factor), while the interaction between these factors are not. Two-factor analysis of variance among the MaxBranch-Simple and the MaxInfo-Simple methods shows that both the goal-state factor and

the method factor are significant (*p<0.001* in both cases); the interaction between these factors is also significant (*p<0.001*).

We observe that, overall, the time required for solving the Easy problems decreases as the goal-state size becomes larger. The exceptions are with the FixedOrder-Simple method, the MaxBranch-Simple method and the MaxInfo-Simple method, where the times for *G=5* are actually higher than those of *G=1*. The differences are not significant for the FixedOrder-Simple method and the MaxInfo-Simple method, but is significant for the MaxBranch-Simple method (Table 27). The general trend of decreasing time across the goal-state sizes is consistent with the observations in Figure 24, as the number of attributes required to be computed also decreases as the goal-state sizes increase. The differences are significant for the FixedOrder-Simple method for *G=5* vs *G=20* (*p=0.025*), the Const-Simple method for *G=5* vs *G=20* and *G=1* vs *G=20*, the MaxBranch-Simple method for *G=1* vs *G=5*, and the MaxInfo-Simple method for *G=5* vs *G=20* and *G=1* vs *G=20* (Table 27).

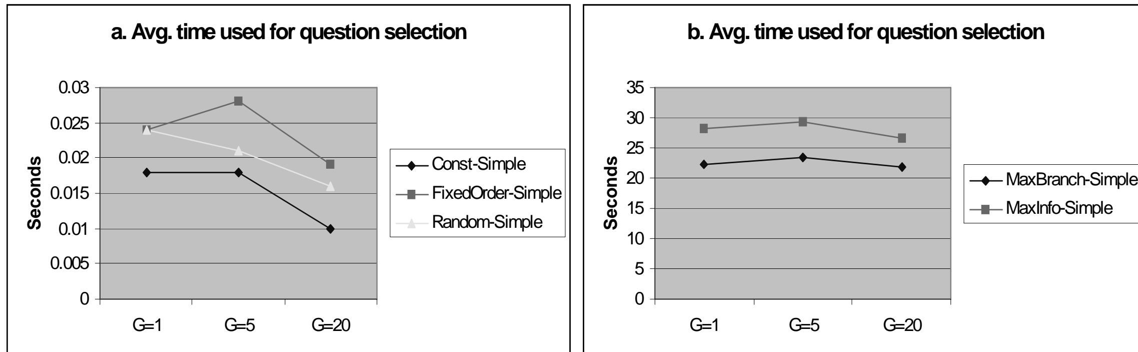


Figure 27: Average system times used for question selection for the Easy task difficulty problems.

Method	G=1 vs G=5		G=5 vs G=20		G=1 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple	1.317	0.096	0.908	0.184	1.465	0.074
FixedOrder-Simple	-0.937	0.176	1.997	0.025	1.524	0.066
Const-Simple	-0.075	0.470	1.826	0.036	3.461	0.001
MaxBranch-Simple	2.82	0.004	1.21	0.115	1.21	0.115
MaxInfo-Simple	1.54	0.065	3.35	0.001	3.35	0.001

Table 27: Pairwise *t*-test results for the system time required for question selection at different goal-state sizes.

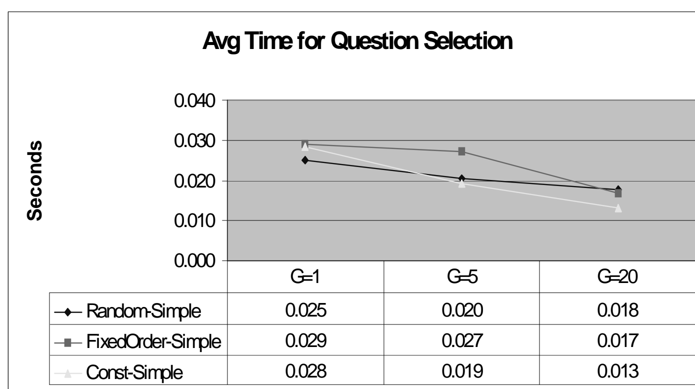
When we conduct pairwise comparisons between the methods, we have found that the significant differences between the methods result mostly from the differences between the FixedOrder-Simple method and the Const-Simple method. Pairwise comparisons of these two methods show

significant differences across all goal-state sizes (*p=0.009* at *G=1, p=0.043* at *G=5*, and *p<0.001* at *G=20*). In addition, the Const-Simple method is also significant better than the Random-Simple method at *G=20* (*p=0.002*). The performance differences between the methods are consistent with what we have observed with the numbers of questions required by the system presented earlier in Figure 24.

When we consider together the number of questions selected by the system and the time required for making these selections from Figure 24 and Figure 27, we can see that Const-Simple method is the most efficient method, overall. Even though the MaxBranch-Simple method requires fewer number of questions, but the time required to obtaining these selections are a multitude more expensive than the Const-Simple method. The MaxInfo-Simple method also consumes much system time, but does not yield must improvement over the Random-Simple method. Similar behaviors are also observed with the Medium and the Hard problems for these two methods. Therefore, in the discussion of the methods in the following sections, we will leave out discussion of the MaxBranch-Simple and the MaxInfo-Simple method.

7.2.1.5. Goal-state sizes and question selection time for the Medium task difficulty problems

Figure 28 presents the average system times used for selecting the information request questions at different goal-state sizes for the Medium task difficulty problems with the Random-Simple, Const-Simple and the FixedOrder-Simple methods. Two-factor analysis of variance among the Random-Simple, FixedOrder-Simple and Const-Simple methods shows that only the goal-state factor is significant (*p=0.008*), while the method factor and the interaction between the method and goal-state size factors are not.



	G=1	G=5	G=20
Random-Simple	0.025	0.020	0.018
FixedOrder-Simple	0.029	0.027	0.017
Const-Simple	0.028	0.019	0.013

Figure 28: Average system times used for question selection for the Medium task difficulty problems.

We observe that for all methods the time required for solving the Medium task difficulty problems decreases as the goal-state size becomes larger. The general trend of decreasing time across the goal-state sizes is consistent with the observations earlier, as the number of attributes required to be computed also decreases as the goal-state sizes increase. The differences, however, are not significant between *G=1* and *G=5* and between *G=5* and *G=20* for all methods. The differences are only significant between *G=1* and *G=20* (Table 28).

Method	G=1 vs G=5		G=5 vs G=20		G=1 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple	0.686	0.247	0.449	0.327	1.738	0.044
FixedOrder-Simple	0.204	0.419	1.306	0.098	2.133	0.019
Const-Simple	1.221	0.114	1.171	0.123	2.969	0.002

Table 28: Pairwise *t*-test results for the system time required for question selection with different goal-state sizes.

7.2.1.6. Goal-state sizes and question selection time for the Hard task difficulty problems

Figure 29 presents the average system times used for selecting the information request questions at different goal-state sizes for the Hard task difficulty problems with the Random-Simple, Const-Simple and the FixedOrder-Simple methods. Two-factor analysis of variance among the Random-Simple, FixedOrder-Simple and Const-Simple methods shows that both the goal-state factor and the method factor are significant (*p=0.04* for the method factor and *p<0.001* for the goal-state size factor), while the interaction between the method and goal-state size factors are not significant.

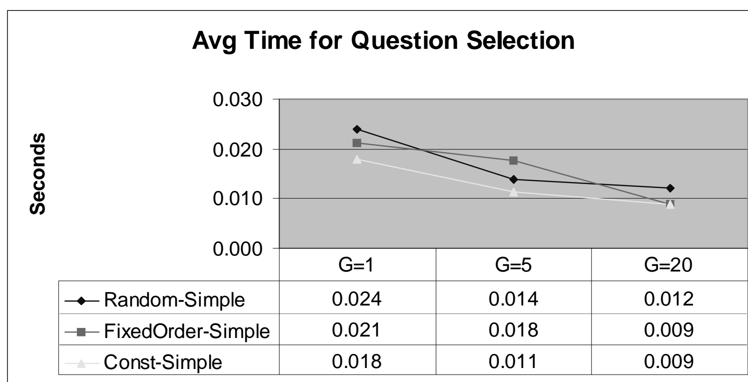


Figure 29: Average system times used for question selection for the Hard task difficulty problems.

In Figure 29, we observe that, similar to what we have observed with Easy and Medium problems, the time required for solving the Hard problems generally decreases as the goal-state size becomes larger. The differences are significant for most of the pairwise comparisons between the different goal-state sizes. The only exceptions are $G=1$ vs $G=5$ for the FixedOrder-Simple method and $G=5$ vs $G=20$ for the Random-Simple method.

When we conduct pairwise comparisons between the methods, we have found the Const-Simple method is the most efficient method, while the FixedOrder-Simple method is more efficient than the Random-Simple method at $G=1$ and $G=20$, but is less efficient at $G=5$. The performance differences are significant between the Const-Simple method and the FixedOrder-Simple method at $G=5$ ($p=0.003$), between the Const-Simple method and the Random-Simple method at $G=20$ ($p=0.023$), and between the FixedOrder-Simple method and the Random-Simple method at $G=20$ ($p=0.028$).

Method	G=1 vs G=5		G=5 vs G=20		G=1 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple	2.366	0.011	0.922	0.180	2.848	0.003
FixedOrder-Simple	1.602	0.057	4.256	<0.001	6.030	<0.001
Const-Simple	2.942	0.002	1.681	0.050	5.361	<0.001

Table 29: Pairwise t-test results for the system time required for question selection with different goal-state sizes.

7.2.2. Goal-state sizes and task success

In this section, we look at how the five methods perform in terms of goal-state sizes. We examine three questions:

For a particular question selection method, what is the effect of different goal-state sizes on task success?

For a particular goal-state size, which method produces the best task success rate?

- Is there a method that outperforms other methods in terms of task success across all goal-state sizes?

In the following sections, we look at the experimental results for the Easy, Medium, and Hard task difficulty problems, respectively.

7.2.2.1. Goal-state size and task success for the Easy task difficulty problems

Figure 30 presents the average kappa scores for Easy problems by different methods at the three goal-state sizes. First of all, we observe that the lines of these methods are relatively flat across the three goal-state sizes, with the exception of the MaxInfo-Simple method. With the MaxInfo-Simple method, the line is decreasing as the size of the goal state increases. Second, we observe that in general, the Const-Simple method and the FixedOrder-Simple method perform better than the Random-Simple method, while the MaxInfo-Simple method performs worse compared with the Random-Simple method. The MaxBranch-Simple method performs worse than the Random-Simple method at $G=1$, but performs better at $G=5$ and $G=20$. Two-factor analysis of variances shows that the goal-state size factor does not exert significant effect on the kappa scores, while the method factor does produce significant difference ($p<0.001$). No significant interaction effects are observed between the two factors.

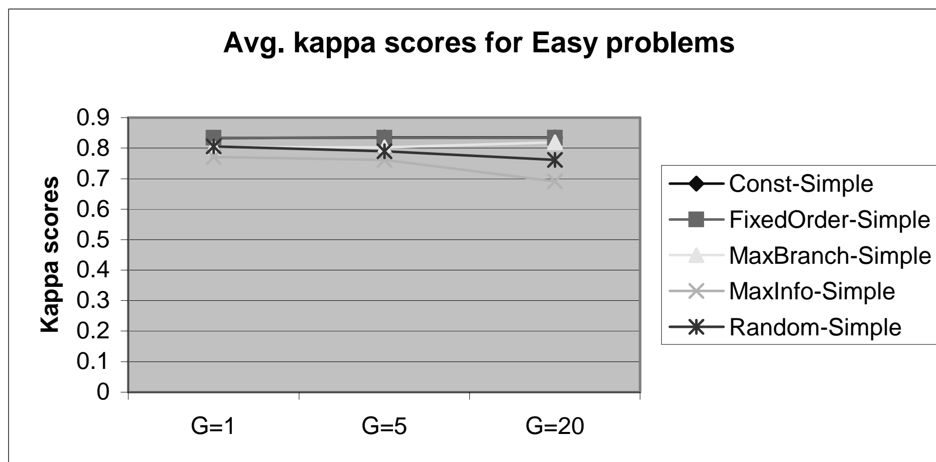


Figure 30: Average kappa scores at different goal-state sizes for the Easy task difficulty problems.

Now, we compare the performances of the five methods. *T*-tests show that the Const-Simple method and the FixedOrder-Simple methods are significantly better than the Random-Simple method, with $p=0.002$ at $G=1$, $p=0.004$ at $G=5$, and $p<0.001$ at $G=20$ between the Const-Simple and Random-Simple methods, and with $p=0.002$ at $G=1$, $p=0.004$ at $G=5$, and $p<0.001$ at $G=20$ between the FixedOrder-Simple and the Random-Simple methods. The FixedOrder-Simple method and the Const-Simple methods perform very similar; *t*-tests show that the differences in performance are not significant. The performance scores from the MaxBranch-Simple method and the MaxInfo-Simple method are very similar to the Random-Simple method. *T*-tests show that the differences are significant only for the hard problems ($p=0.014$ between the MaxInfo-Simple and

the Random-Simple methods, and $p=0.003$ between the MaxBranch-Simple and the Random-Simple methods).

7.2.2.2. Goal-state size and task success for the Medium task difficulty problems

Figure 31 presents the average kappa scores for Medium problems by different methods at the three goal-state sizes. For Medium problems, we observe that the lines for FixedOrder-Simple and Const-Simple methods are relatively flat, but there is more variation for the Random-Simple method, the MaxBranch-Simple method, and the MaxInfo-Simple method. With the MaxInfo-Simple method, the line is decreasing as the size of the goal state increases. Second, we observe that the Const-Simple method and the FixedOrder-Simple method perform better than the Random-Simple method, while the MaxInfo-Simple method performs worse compared with the Random-Simple method. The MaxBranch-Simple method performs better than the Random-Simple method at $G=1$and $G=20$, but performs worse at $G=5$. Two-factor analysis of variances shows that the method factor exerts significant effect on the kappa scores, with $p<0.001$. The goal-state size factor does not have significant influence over the kappa scores. There is no significant interaction effect between the method factor and the goal-state size factor.

Now we compare the performances of the methods in detail. *T*-tests show that the FixedOrder-Simple method is significant more efficient than the Random-Simple method across all goal states, with $p<0.001$ at $G=1$, $p=0.026$ at $G=5$, and $p=0.001$ at $G=20$. Compared with the Random-Simple method, the Const-Simple method is significantly more efficient at G=1 ($p=0.001$) and at G=20 ($p=0.001$). It is also more efficient at G=5, but the difference is not significant. The FixedOrder-Simple is in general more efficient than the Const-Simple method, but their difference is not significantly different. The Random-Simple method is more efficient than the MaxInfo-Simple method across all goal states, but the difference is only significant at $G=20$ ($p=0.018$). Compared with the Random-Simple method, the MaxBranch-Simple method is more efficient at $G=5$, but less efficient at $G=1$ and $G=20$. However, the difference is significant only when G=5.

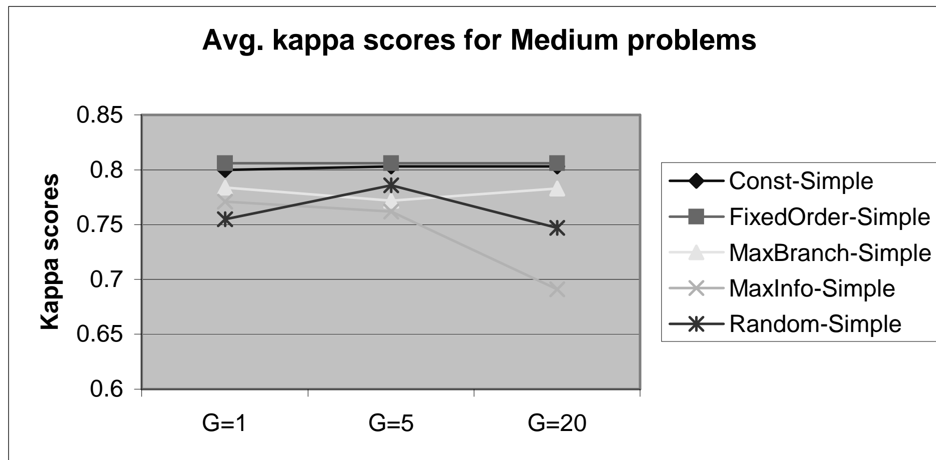


Figure 31: Average kappa scores at different goal-state sizes for the Medium task difficulty problems.

7.2.2.3. Goal-state size and task success for the Hard task difficulty problems

Figure 32 presents the average kappa scores for Hard problems by different methods at the three goal-state sizes. First of all, we observe that the lines of these methods are relatively flat across the three goal-state sizes, with the exception of the MaxInfo-Simple method. With the MaxInfo-Simple method, the line is decreasing as the size of the goal state increases. Second, we observe that in general, the Const-Simple method and the FixedOrder-Simple method perform better than the Random-Simple method, while the MaxBranch-Simple method performs worse compared with the Random-Simple method. The MaxInfo-Simple method performs better than the Random-Simple method at $G=1$, but performs worse at $G=5$ and $G=20$. Two-factor analysis of variances shows that both the method and the goal-state factors exerts significant effect on the kappa scores, with $p<0.001$ for the method and $p=0.01$ for the goal-state factor. The interaction effect is also significant, with $p=0.002$.

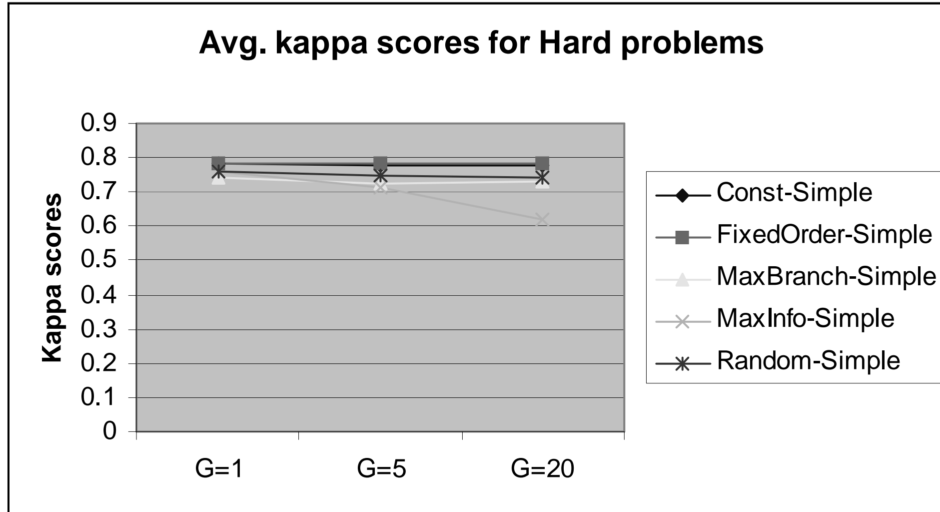


Figure 32: Average kappa scores at different goal-state sizes for the Hard task difficulty problems.

Detailed *t*-tests show that, for the Random-Simple method, the FixedOrder-Simple method and the Const-Simple method, there are no significant differences between the kappa scores at different goal-state sizes. With the MaxBranch-Simple method, the kappa scores at $G=5$ are significantly worse than those at $G=1$ ($p=0.04$). The MaxInfo-Simple method demonstrates significant performance decrease from $G=1$ to $G=20$. The differences are significant, with $p=0.001$ when comparing $G=1$ with $G=5$, $p<0.001$ when comparing $G=5$ with $G=20$, and $p<0.001$ when comparing $G=1$ with $G=20$.

Next, we compare the performances of the methods. *T*-tests show that the Const-Simple method and the FixedOrder-Simple methods are significantly better than the Random-Simple method, with $p=0.012$ at $G=1$, $p=0.027$ at $G=5$, and $p=0.008$ at $G=20$ between the Const-Simple and Random-Simple methods, and with $p=0.008$ at $G=1$, $p=0.003$ at $G=5$, and $p=0.002$ at $G=20$ between the FixedOrder-Simple and the Random-Simple methods. Even though the FixedOrder-Simple method performs slight better than the Const-Simple method, the differences in kappa scores between these two methods are not significant. While the MaxBranch-Simple method is in general worse compared with the Random-Simple method, the differences are not significant. Compared with the Random-Simple method, the MaxInfo-Simple method performs a little better at $G=1$, but the difference is not significant. The MaxInfo-Simple method performs worse than the Random-Simple method at $G=5$ and $G=20$; these differences are significant ($p=0.03$ and $p<0.001$, respectively).

7.2.3. Goal-state sizes and relaxation efficiency

To examine the effect of goal-state sizes on relaxation selection efficiency, we investigate three questions:

For a particular relaxation selection method, what is the effect of different goal-state sizes?

To solve problems at a particular goal-state size, what is the most efficient method in relaxation candidate selection?

Is there a method that outperforms other methods in terms of relaxation candidate selection across all goal-state sizes?

7.2.3.1. Goal-state sizes and relaxation efficiency for the Easy task difficulty problems

Two-way analysis of the effects of different goal sizes and heuristic methods on the number of relaxation candidates for Easy problems shows that both the goal-state size factor and the method factor have significant effects ($p=0.006$ for the method factor and $p<0.001$ for the goal-state factor). But the two factors do not interact in a significant manner.

Figure 33 presents the average number of relaxation candidate selections by the system to solve the problems. We observe that in the figure, the lines are decreasing as the goal-state size increases, i.e., the number of relaxation candidates required for solving the problems is in reverse proportion with the size of the pre-determined solution states. That is, the larger the solution size, the fewer the number of relaxation candidates required. In particular, when we look at the numbers of candidate required for the question selection, pair-wise comparison of two goal-state sizes shows that the difference is significant for almost all methods (Table 30). For example, for the Random-Simple method, fewer numbers of questions are required to solve the $G=20$ problems than the $G=5$ problems ($p<0.007$, one-tail), and fewer numbers of questions are required to solve the $G=5$ problems than the $G=1$ problems ($p<0.007$, one-tail). The only exception is with the Random-MinSize method when comparing goal-state sizes $G=1$ vs $G=5$, where the two samples from the two states are almost the same.

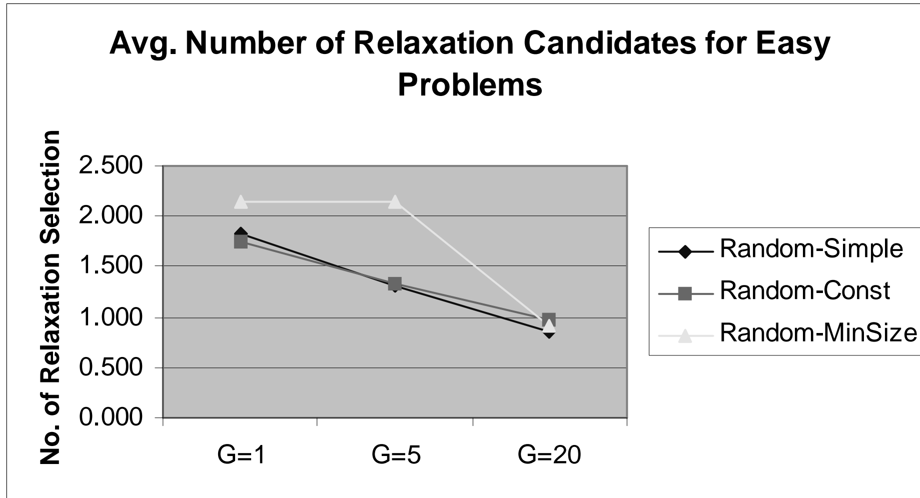


Figure 33: Average numbers of relaxation candidates for the Easy task difficulty problems.

Methods	G=1 vs G=5		G=5 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value
Random-Simple	2.523	0.007	2.525	0.007
Random-Const	2.239	0.014	1.920	0.029
Random-MinSize	N/A	N/A	5.582	<0.001

Table 30: Pairwise comparisons of the number of relaxation candiates at different goal-state sizes for the Easy task difficulty problems.

	G=1		G=5		G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple vs Random-Const	0.333	0.370	-0.096	0.462	-0.608	0.273
Random-Simple vs Random-MinSize	-1.742	0.043	-4.008	<0.001	-0.614	0.271
Random-Const vs Random-MinSize	-2.014	0.024	-3.792	<0.001	0.271	0.394

Table 31: Pairwise comparisons of the number of relaxation candidates between methods at different goal-state sizes for the Easy task difficulty problems.

When comparing the methods across the goal-state sizes, we observe that across all the goal-state sizes (*G=1*, *5*, and *20*), the Random-Simple baseline is significantly more efficient than the Random-MinSize method with respect to the number of candidates selected (Table 31) at goal-state sizes *G=1* and *G=5* ($p=0.043$ at *G=1* and $p<=0.001$ at *G=5,* respectively). The Random-Const method is also significantly more efficient than the Random-Simple method with respect to the number of relaxation candidates selected at goal-state sizes *G=1* and *G=5* ($p=0.024$ at *G=1* and $p<=0.001$ at *G=5,* respectively). Compared with the Random-Simple method, the Random-Const method is slight more efficient at *G=1*, but less efficient at *G=5* and *G=20*. The differences at all states, however, are not significant.

7.2.3.2. Goal-state sizes and relaxation efficiency for the Medium task difficulty problems

Figure 34 presents the average number of relaxation candidate selections by the system to solve the Medium problems. We observe that in the figure, the lines are decreasing as the goal-state size increases, i.e., the number of relaxation candidates required for solving the problems is in reverse proportion with the size of the pre-determined solution states. That is, the larger the solution size, the fewer the number of relaxation candidates required. When we compare the methods, we observe that the Random-Simple method is more efficient than the Random-Const method, which is in turn more efficient than the Random-MinSize method across all goal-state sizes. Two-way analysis of the effects of different goal sizes and heuristic methods on the number of relaxation candidates for the problems shows that only the goal-state size factor has significant effect on the performance ($p<0.001$). There is no significant difference between the methods and the interaction between the goal-state size factor and the method factor is not significant.

Methods	G=1 vs G=5		G=5 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value
Random-Simple	3.506	<0.001	1.829	0.036
Random-Const	5.411	<0.001	1.742	0.043
Random-MinSize	3.936	<0.001	1.608	0.057

Table 32: Pairwise comparisons of the number of relaxation candidates at different goal-state sizes for the Medium task difficulty problems.

When we look at in detail the numbers of relaxation candidates required for solving over-constrained problems, pair-wise comparison of two goal-state sizes shows that the difference is significant for almost all methods (Table 32). For example, for the Random-Simple method, fewer numbers of relaxation selections are required for solving the $G=20$ problems than the $G=5$ problems ($p<0.036$, one-tail), and fewer numbers of relaxation selections are required for solving the $G=5$ problems than the $G=1$ problems ($p<0.001$, one-tail). The only exception is with the Random-MinSize method when comparing goal-state sizes $G=5$ vs $G=20$, where the two samples the $G=5$ states requires fewer number of relaxation selections than the $G=20$ does, but the difference is not significant.

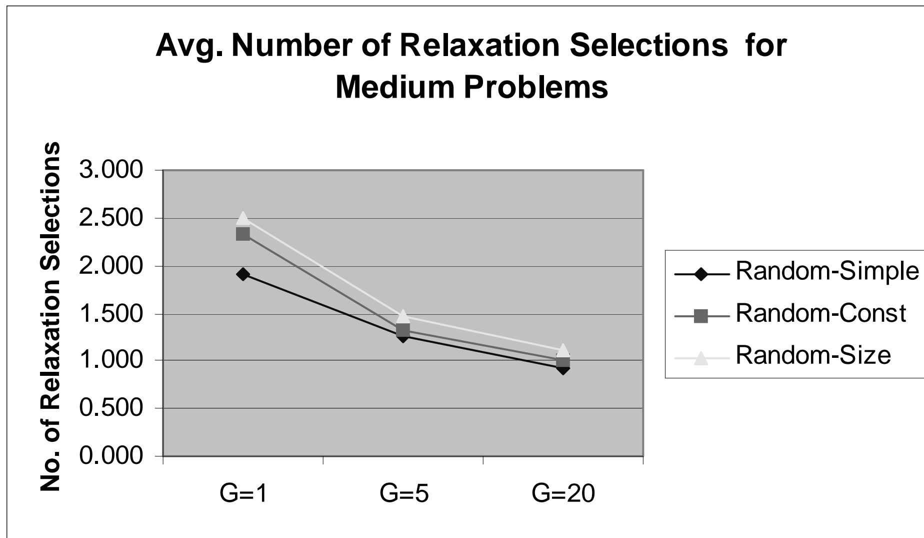


Figure 34: Average numbers of relaxation candidates for the Medium task difficulty problems.

7.2.3.3. Goal-state sizes and relaxation efficiency for the Hard task difficulty problems

Figure 35 presents the average number of relaxation candidate selections by the system to solve the Hard problems. Similar to what we have observed for Easy and Medium problems, we observe that for Hard problems, the lines are decreasing as the goal-state size increases, i.e., the number of relaxation candidates required for solving the problems is in reverse proportion with the size of the pre-determined solution states. When we compare the methods, we observe that at G=1, the Random-Simple method is more efficient than the Random-Const method, which is in turn more efficient than the Random-MinSize method. At *G=5*, the Random-Const method is more efficient than the Random-Simple method, which in turn is more efficient than the Random-MinSize method. At *G=20*, the Random-Const is more efficient than the Random-Size method, which is in turn more efficient than the Random-MinSize method. Two-way analysis of the effects of different goal sizes and heuristic methods on the number of relaxation candidates for the problems shows that only the goal-state size factor has significant effect on the performance ($p<0.001$). There is no significant difference between the methods and the interaction between the goal-state size factor and the method factor is not significant.

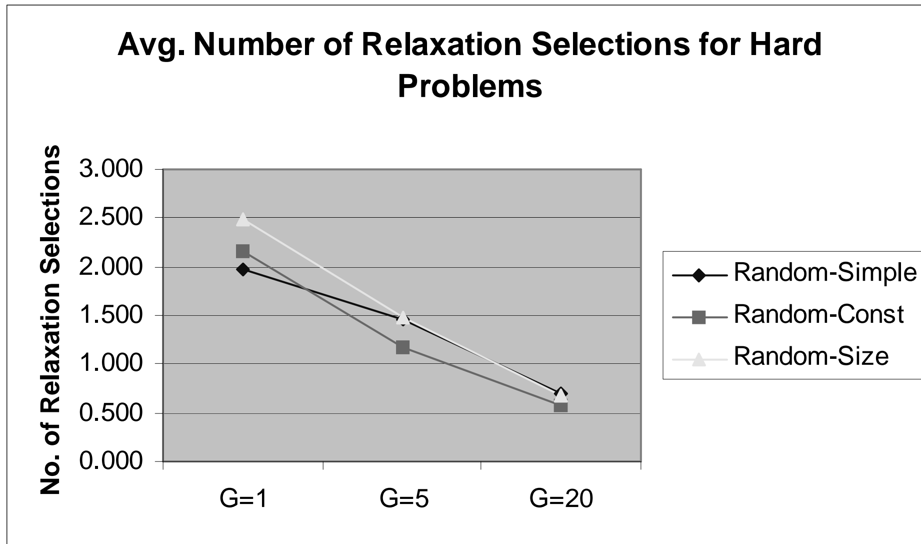


Figure 35: Average numbers of relaxation candidates for the Hard task difficulty problems.

When we look at in detail the numbers of relaxation candidates required for solving over-constrained problems, pair-wise comparison of two goal-state sizes shows that the difference is significant for all methods (Table 33). For example, for the Random-Simple method, fewer numbers of relaxation selections are required for solving the *G=20* problems than the *G=5* problems (*p<0.007,* one-tail), and fewer numbers of relaxation selections are required for solving the *G=5* problems than the *G=1* problems (*p<0.001,* one-tail).

Methods	G=1 vs G=5		G=5 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value
Random-Simple	2.553	0.007	3.653	<0.001
Random-Const	5.140	<0.001	3.730	<0.001
Random-MinSize	4.011	<0.001	4.181	<0.001

Table 33: Pairwise comparisons of the number of relaxation candidates at different goal-state sizes for the Hard task difficulty problems.

7.2.3.4. Goal-state sizes and relaxation time for the Easy task difficulty problems

Figure 36 presents the average system times used for selecting the relaxation candidates at different goal-state sizes for Easy problems. From the figure, we observe that the time used for the Random-Simple method and the Random-Const methods are very similar, which the Random-MinSize method requires significantly more time. Two-factor analysis of variance between the method factor and the goal-state size factor shows that both the goal-state factor and the method

factor are significant (*p<0.001* for the method factor, and *p<0.001* for the goal-state size factor). The interaction between the two factors is also significant.

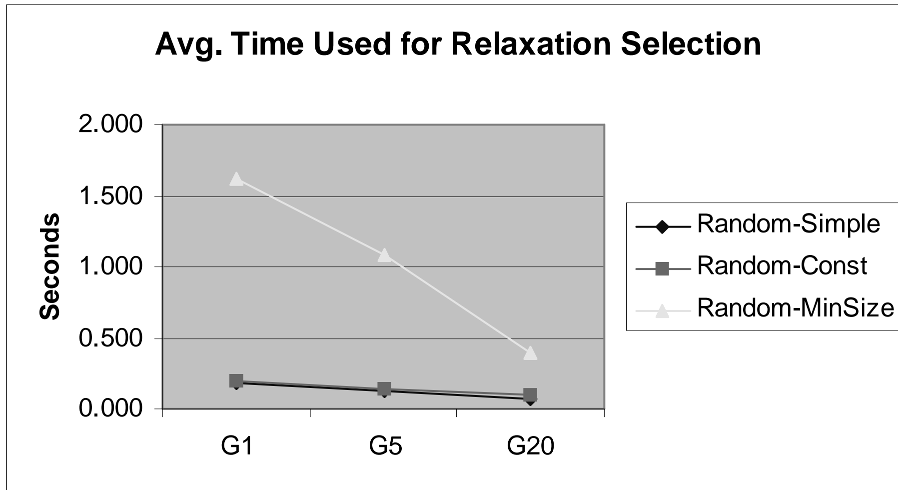


Figure 36: Average system time used for relaxation candidate selection for the Easy task difficulty problems.

We observe that, overall, the time required for solving the Easy problems decreases as the goal-state size becomes larger. The general trend of decreasing time across the goal-state sizes is consistent with the observations in Figure 33, as the number of relaxation candidates required to be computed also decreases as the goal-state sizes increase. The differences are significant for all the methods when pairwise states are compared (Table 34).

Method	G=1 vs G=5		G=5 vs G=20		G=1 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple	2.077	0.021	2.901	0.003	4.854	<0.001
Random-Const	1.960	0.027	2.158	0.018	5.303	<0.001
Random-MinSize	2.211	0.015	3.372	<0.001	5.934	<0.001

Table 34: Pairwise *t*-test results for the system time used for relaxation candidate selection for the Easy task difficulty problems at different goal-state sizes.

When we conduct pairwise comparisons between the methods, we have found that no significant differences between the Random-Simple method and the Random-Const method, even though the Random-Simple method is somewhat more efficient across all state sizes. When we compare the differences between the Random-Simple method and the Random-MinSize method and between theRandom-Const method and the Random-MinSize method, the differences are significant for all the pairwise comparisons (*p<0.001* for all). As is illustrated in Figure 36, the Random-MinSize

method requires significantly more time than either the Random-Simple method or the Random-Const method does.

7.2.3.5. Goal-state sizes and relaxation time for the Medium task difficulty problems

Figure 37 presents the average system times used for selecting the relaxation candidates at different goal-state sizes for Medium problems. From the figure, again, we observe that the time used for the Random-Simple method and the Random-Const methods are very similar, which the Random-MinSize method requires significantly more time. Two-factor analysis of variance between the method factor and the goal-state size factor shows that both the goal-state factor and the method factor are significant ($p<0.001$ for the method factor, and $p<0.001$ for the goal-state size factor). The interaction between the two factors is also significant ($p<0.001$).

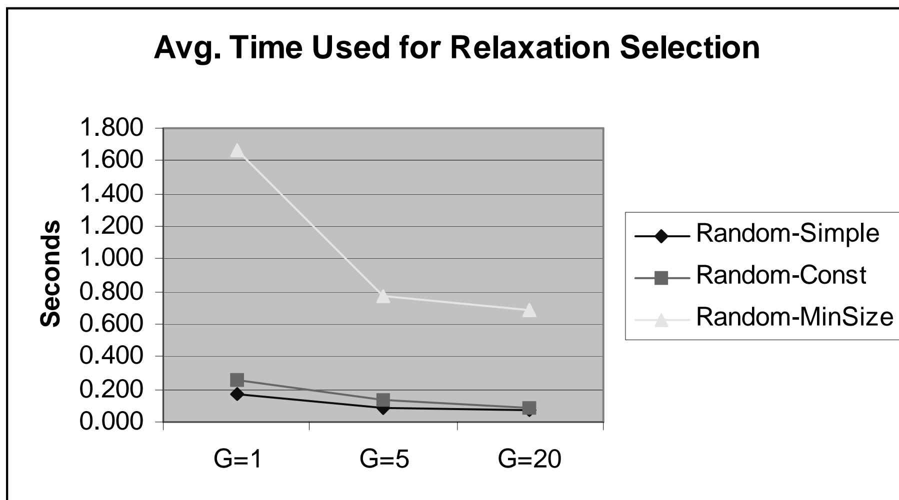


Figure 37: Average system time used for relaxation candidate selection for the Medium task difficulty problems.

Similar to what we have observed with Easy problems, the time required for solving the Medium problems also decreases as the goal-state size becomes larger. The differences are significant for almost all the methods except the Random-Simple method between $G=5$ and $G=20$ and the Random-MinSize method between $G=1$ and $G=5$. (Table 35).

Method	G=1 vs G=5		G=5 vs G=20		G=1 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple	3.338	0.001	0.384	0.351	3.371	0.001
Random-Const	4.128	<0.001	1.824	0.037	5.678	<0.001
Random-MinSize	3.668	<0.001	0.515	0.304	4.273	<0.001

Table 35: Pairwise t-test results for the system time used for relaxation candidate selection for the Medium task difficulty problems at different goal-state sizes.

When we conduct pairwise comparisons between the methods, we have found that the Random-Simple method is more efficient than the Random-Const method. The difference is significant at both the G=1 and G=5 goal-state sizes (*p=0.001* and *p=0.022*, respectively), but not significant at G=20 goal-state size. Both the Random-Simple method and the Random-Const method are significant more efficient than the Random-MinSize method across all goal-state sizes.

	G=1		G=5		G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple vs Random-Const	-3.109	0.001	-2.055	0.022	-0.873	0.193
Random-Simple vs Random-MinSize	-8.167	<0.001	-4.894	<0.001	-4.205	<0.001
Random-Const vs Random-MinSize	-7.714	<0.001	-4.313	<0.001	-4.227	<0.001

Table 36: Pairwise comparisons between the relaxation methods at different goal-state sizes for the Medium task difficulty problems.

7.2.3.6. Goal-state sizes and relaxation time for the Hard task difficulty problems

Figure 38 presents the average system times used for selecting the relaxation candidates at different goal-state sizes for Hard problems. From the figure, again, we observe that the time used for the Random-Simple method and the Random-Const methods are very similar, which the Random-MinSize method requires significantly more time. Two-factor analysis of variance between the method factor and the goal-state size factor shows that both the goal-state factor and the method factor are significant ($p<0.001$ for the method factor, and $p<0.001$ for the goal-state size factor). The interaction between the two factors is also significant ($p<0.001$).

Similar to what we have observed with Easy and Medium problems, the time required for solving the Hard problems also decreases as the goal-state size becomes larger. The differences are significant for almost all the methods except the Random-Simple method between *G=1* and *G=5.* (Table 37).

When we conduct pairwise comparisons between the methods, we have found that the Random-Simple method is more efficient than the Random-Const method at G=1, but is less efficient than the Random-Const method at G=5 and G=20. The differences, however, are not significant. Both the Random-Simple method and the Random-Const method are significantly more efficient than the Random-MinSize method across all goal-state sizes.

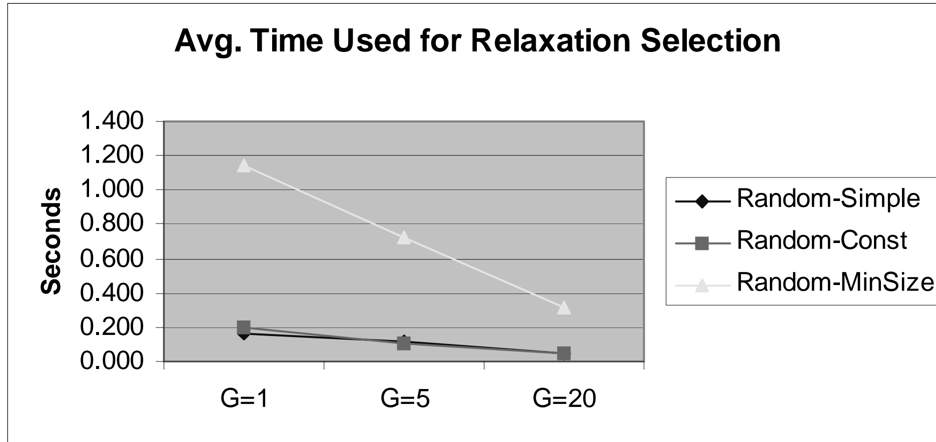


Figure 38: Average system time used for relaxation candidate selection for the Hard task difficulty problems.

Method	G=1 vs G=5		G=5 vs G=20		G=1 vs G=20	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
Random-Simple	1.593	0.058	4.763	<0.001	3.407	0.001
Random-Const	3.979	<0.001	7.216	<0.001	3.899	<0.001
Random-MinSize	2.203	0.016	5.935	<0.001	3.108	0.001

Table 37: Pairwise *t*-test results for the system time used for relaxation candidate selection for the Hard task difficulty problems at different goal-state sizes.

7.2.4. Goal-state sizes: summary

To summarize, we observe that as the size of goal states increases, the number of questions required for solving a problem generally decreases for all methods. This observation holds true for all kinds of problems, regardless of the task difficulty levels.

Among the methods, the FixedOrder-Simple method is the least efficient one. It is significantly less efficient compared with the Random-Simple method for all types of problems. Compared with the Random-Simple method, the MaxInfo-Simple method is significantly more efficient with Easy and Medium task difficulty problems at all goal-state sizes, but is significantly less efficient with Hard problems at *G=20*. The Const-Simple and the MaxBranch-Simple methods are significantly more efficient than the Random-Simple method at all goal-state sizes for all types of problems. Between the Const-Simple method and the MaxBranch-Simple method, the MaxBranch-Simple method is generally more efficient; the differences, however, are significant only for Medium problems at all goal-state sizes.

In terms of the average system time, the goal-state sizes affect the time used for question selection for most of the methods. The general trend is that with increased goal-state size, the time used for question selection decreases. Overall, the Const-Simple method is the most efficient method among all methods, while the FixedOrder-Simple method is generally less efficient than the Random-Simple method. The MaxBranch-Simple and MaxInfo-Simple method all require significant amount of time for question selection, thus are not as desirable as the Const-Simple method.

In terms of task success, the size of the goal states generally does not demonstrate significant influence over the kappa scores for all types of problems. Should the system perform better at a certain state size? On the one hand, since smaller goal-state size requires more clarification questions from the user, thus yielding more specified information needs, we would expect that the success score for the smaller goal-state size to be higher. On the other hand, since each clarification question is chosen among competing candidates, selecting a particular candidate might lead to solutions that end in less optimal goal state, resulting in lower kappa scores. Since the variations in kappa scores across different goal-state sizes are not significant, we can only account for the differences as random noise.

Some exceptions to this generalization include the MaxInfo-Simple method for Hard problems for pair-wise comparisons between the states $G=1$ and $G=5$, and the MaxBranch-Simple method for Hard problems between any pairs of the goal states. Such observations suggest that the MaxBranch-Simple method and the MaxInfo-Simple method are sensitive to the size of the goal states to a certain degree, while the other methods are not sensitive.

The FixedOrder-Simple method and the Const-Simple method demonstrate the best of kappa scores among all the methods for problems of different task difficulty levels, with the FixedOrder-Simple method generally performs better than the Const-Simple method, even though the difference is not significant. The MaxInfo-Simple method generally performs worse compared with the baseline Random-Simple method for all types of problems. Compared with the Random-Simple method, the MaxBranch-Simple method sometimes produce better kappa scores, sometimes worse. Such an observation is consistent with our earlier observations on kappa scores in section 7.1.2.

In terms of the selection of relaxation candidates, the empirical results show that, in general, the number of relaxation candidates decreases as the size of the goal-state size increases. This is reasonable because when the size of the goal state increases, the number of questions requested by the system decreases. Therefore, when over-constrained situations occur, the possible number of

relaxation candidates is in general smaller. When we look at the relaxation methods across all states, we observe that for almost all states, the differences between the methods are not significant. This suggests that, like what we have observed in section 7.1.3, that effectiveness of the relaxation methods is strongly dominated by the question selection methods.

In terms of the system time required for selection of relaxation candidates, the empirical results show that, in general, the time required for selecting relaxation candidates decreases as the size of the goal-state size increases. This is reasonable because when the size of the goal-state size increases, the number of questions requested by the system decreases, consequently the system time required for identifying the relaxation candidates. When we look at the relaxation methods across all states, we observe that for all states, the Random-Simple method and the Const-Simple Method are significantly more efficient than the Random-MinSize method. The differences between the Random-Simple method and the Random-Const method is not significant except at *G=1* and *G=5* for Medium problems. This suggests again that in general when the question selection method is fixed, knowledge sources such as constraint hierarchy or solution status do not contribute more to the efficiency of relaxation candidate selection.

7.3. Result Analysis: The Effect of Interval Selection

Three interval attributes exist in our example attribute vector: DeptTime, ArriveTime, and Reward. Different interval groupings of the values for an attribute influence the branching factor of that attribute, which in turn could influence the effectiveness of the heuristic methods. In this section, we examine to what extent different interval groupings affect the effectiveness of the heuristic methods for question selection and relaxation candidate selection.

In this work, we only look at the effect of time intervals on the time factors. To simplify the discussion, we treat departure time and arrival time identically. We compare two interval grouping methods for the time factors.

Hour based: group the departure times and arrive times into 24 segments, with each segment starting from *h:00:00* and ending at *h:59:59*, where $h \in [0,24)$. The maximal branching factor is 24.

4-hour segments: group the time values into 6 segments, roughly corresponding to the times of day. They are early morning $h \in [3:00:00-7:00:00)$, morning $h \in [7:00:00-11:00:00)$,

afternoon $h \in [11:00:00-15:00:00)$, late afternoon $h \in [15:00:00-19:00:00)$, evening $h \in [19:00:00-23:00:00)$, and night $h \in [23:00:00-3:00:00)$. The maximal branching factor is 6. Note this segmentation also roughly corresponds to the segments of peaks and bottoms of flights spanning over the 24-hour period.

In real world situations, users often refer to time in terms of time of day rather than specific hour intervals[8]. Therefore, the 4-hour-based value grouping method is more realistic compared with the hour-based grouping.

We have observed from the previous two sections that regardless the task difficulty level and the goal-state size, among the fixed ordering methods, the Const-Simple method generally outperforms the FixedOrder-Simple method, and among the methods employing solution size information, the MaxBranch-Simple method outperforms the MaxInfo-Simple method. Thus, in this section, we only examine the Const-Simple method and the MaxBraching-Simple method with respect to the Random-Simple method.

We conducted the following experiments to investigate the following issues:

- Is there any interaction between goal-state size and interval selection (I=1hour or I=4hour)?
- Does different interval selection significantly change the behaviors of the heuristic methods?

Here, we do not distinguish the problems by their degree of task difficulty, as we have done in previous sections, the task difficulty factor does not change the performance comparisons of the methods. The goal-state size is set to *G=5*. In the following reported experiments, we report on a total of 180 problems, which consist of 60 problems for the Easy, Medium, and Hard difficulty levels, respectively.

7.3.1. Interval selection and question selection efficiency

Table 38 presents the summary statistics for different question selection methods with I=1hr or I=4hr. Two-factor analysis of variance of the number of questions using the method factor and the interval factor shows that both the method factor and the interval size factor are significant

---

[8] Often, an information seeker in the flight information domain specifies a constraint using inequalities, e.g., departure time after 7pm. For attributes with finite domain, inequalities can be represented as intervals of certain size. Therefore, constraints with inequalities are covered when the interval size is sufficiently large.

($p<0.001$ for the method factor and $p=0.017$ for the interval factor). The interaction between these two factors is also significant ($p=0.002$).

For both of the Random-Simple method and the MaxBranch-Simple method, the average number of questions required by the system decreases as the interval size increases, while the average number of questions actually increases for the Const-Simple method. Pairwise *t*-tests show that the performance difference resulting from different interval sizes is only significant for the MaxBranch-Simple method ($p<0.001$). When we compare the average number of questions required between the methods, paired sample *t*-tests show that the MaxBranch-Simple method is significantly more efficient than the Const-Simple method ($p<0.001$), which is in turn significantly more efficient than the Random-Simple method ($p<0.001$) with either I=1hr or I=4hr.

	Avg. No. of Questions		Avg. System Time	
	I=1hr	I=4hr	I=1hr	I=4hr
Random-Simple	4.567	4.378	0.016	0.014
Const-Simple	3.683	3.828	0.015	0.013
MaxBranch-Simple	3.361	2.872	21.836	35.597

Table 38: Statistics of question selection for different question selection methods with I=1hr and I=4hr.

In terms of question selection time, two-factor analysis of variance of the number of questions using the method factor and the interval factor shows that both the method factor and the interval size factor are significant ($p<0.001$ for the method factor and $p<0.001$ for the interval factor). The interaction between these two factors is also significant ($p<0.001$). Both the Random-Simple method and the Const-Simple method has decreasing system time as the interval size increases. The differences, however, is not significant. When the interval size increases, the MaxBranch-Simple method actually uses increased system time. The increase is statistically significant ($p<0.001$). Such observations suggest that for solution-size insensitive methods such as the Random-Simple method and the Const-Simple method, an attribute with a coarser classification (e.g., DeptTime with 4-hour intervals) does not significantly affect efficiency. However, for solution-size sensitive method such as the MaxBranch-Simple method, the change in interval sizes does affect performance significantly. When we compare the performance between methods, paired sample *t*-tests show that the Const-Simple method is significantly more efficient than the Random-Simple method ($p<0.001$) with either I=1hr or I=4hr, both of which are significantly more efficient than the MaxBranch-Simple method ($p<0.001$) with either I=1hr or I=4hr. Such observations are consistent with what we have observed earlier.

7.3.2. Interval selection and task success

When we compare the task success scores for the three methods (Table 39), we observe that with increased interval size, the kappa scores for the Const-Simple method and the MaxBranch-Simple method decrease, while the kappa scores for the Random-Simple method increases. Two-factor analysis of variance with the method factor and the interval factor shows that neither factor has a significant effect on the kappa scores. The interaction between the factors is not significant, either.

	I=1hr	I=4hr
Random-Simple	0.784	0.799
Const-Simple	0.835	0.812
MaxBranch-Simple	0.802	0.774

Table 39: Kappa scores wtih I=1hr or I=4hr for different question selection methods.

7.3.3. Interval selection and relaxation efficiency

Table 40 presents the statistics for relaxation candidate selection for different relaxation candidate selection methods with I=1hr or I=4hr. Two-factor analysis of variance of the number of relaxation candidates using the method factor and the interval factor shows that both the method factor and the interval size factor are significant ($p<0.001$ for both the method factor and the interval factor). The interaction between these two factors is also significant ($p=0.006$).

	Avg. No. of Relaxation Candidates		Avg. System Time	
	I=1hr	I=4hr	I=1hr	I=4hr
Random-Simple	1.818	1.908	0.183	0.166
Const-Simple	1.774	1.731	0.111	0.109
MaxBranch-Simple	1.778	1.7	0.214	0.207

Table 40: Statistics for relaxation candidate selection for different question selection methods with I=1hr and I=4hr.

For all the methods, the average number of relaxation candidate selected by the system decreases as the interval size increases. Pairwise *t*-tests show that the performance differences resulting from different interval sizes are significant for all the methods ($p<0.001$ for the Random-Simple method and the MaxBranch-Simple method and $p=0.009$ for the Const-Simple method). When we compare the number of relaxation candidates between the methods, pairwise *t*-tests show that both the Const-Simple method and the MaxBranch-Simple method are significantly more efficient than the Random-Simple method with either I=1hr or I=4hr ($p<0.001$ for all pairs). The MaxBranch-Simple

method is more efficient than the Const-Simple method with either I=1hr or I=4hr; the difference, however, is only significant with I=4hr ($p<0.001$).

In terms of the system time required for selecting the relaxation candidates, for all the methods, the average system time used for relaxation candidate selection decreases as the interval size increases. Pairwise *t*-tests show that the differences resulting from different interval sizes are significant for all the methods ($p<0.001$ for the Random-Simple method and the MaxBranch-Simple method and $p=0.024$ for the Const-Simple method). When we compare the system times between the methods, pairwise *t*-tests show that both the Const-Simple method and the Random-Simple method are significantly more efficient than the MaxBranch-Simple method with either I=1hr or I=4hr ($p<0.001$ for all pairs). The Const-Simple method is also significantly more efficient than the Random-Simple method with either I=1hr or I=4hr ($p<0.001$).

7.3.4. Interval selection: summary

To summarize, with increased interval size, the numbers of questions elicited by the system and the relaxation candidates selected by the system generally decrease. While such decreases are not significant in terms of question selection (except with the MaxBranch-Simple method), the decrease are significant for relaxation candidate selection. With both interval sizes, the MaxBranch-Simple method is more efficient than the Const-Simple method, which is more efficient than the Random-Simple method in terms of the number of questions. The MaxBranch-Simple method, however, consumes significantly more system time for question selection. In terms of selecting relaxation candidates, the Const-Simple method and the MaxBranch-Simple method are significantly more efficient than the Random-Simple method in terms of the number of relaxation candidates, with the Const-Simple method also outperforms both the Random-Simple method and the MaxBranch-Simple method in terms of system time.

7.4. Result Analysis: The Effect of Task Complexity

In this section, we report our experiments on problems involving multi-leg flight planning. The purpose of these experiments is to evaluate the scalability of the heuristic methods and to validate whether the heuristics effective for single-leg flight planning continue to be effective for more complex problems.

These experiments are based on test collection C_2, where three-leg flights involving four cities are requested (section 6.1.4). Twenty problems are three-destination flights where the final destination is different from the initial departure city. Another twenty problems are two-destination flights with the final destination city being the departure city. For each leg of the flights, the attributes of that leg are assigned a preference strength value based on the preference strength distributions (section 6.1.3). Since the multiple legs of a trip can have task difficulty levels, we do not calculate the total task difficulty level for the entire trip. Time intervals are set at 1 hour, and the goal-state size is set to 5. Next, we report the results of the two sets of problems with respect to question selection efficiency, task success, and relaxation candidate selection efficiency.

7.4.1. Task complexity and question selection efficiency

To compare the effectiveness of the question selection methods, we evaluate the Random-Simple method, the Const-Simple method, and the MaxBranch-Simple method, with the Random-Simple method as the baseline and the relaxation candidate selection method fixed as the simple chronological backtracking method.

Table 41 presents the performance statistics for the 2-destination problems. We observe that the MaxBranch-Simple method is more efficient than the Const-Simple method, which is in turn more efficient than the Random-Simple method in terms of the average number of questions requested by the system for solving the problems. One-factor analysis of variance shows that the differences are significant ($p=0.001$). Specifically, even though the Const-Simple method is in general more efficient than the Random-Simple method, pairwise t-tests show that the difference is, however, not significant. The MaxBranch-Simple method, however, is significantly more efficient than both the Random-Simple method ($p=0.001$) and the Const-Simple method ($p=0.002$).

In terms of the average question selection time, we notice significant differences between the methods. In particular, the MaxBranch-Simple method is much more expensive than either the Random-Simple method or the Const-Simple method. One-factor analysis of variance shows that the differences are significant ($p<0.001$). Pairwise t-tests also show significant differences, with the Const-Simple method as the most efficient method and the MaxBranch-Simple method as the most expensive method ($p<0.001$).

In terms of the average number of relaxation candidate selections, one-factor ANOVA shows that the differences are significant ($p=0.001$). Even though the Const-Simple method is more efficient

than the Random-Simple method, *t*-test does not yield significant differences. However, the MaxBranch-Simple method is significantly more efficient than either the Const-Simple method (*p=0.001*) or the Random-Simple method (*p<0.001*).

	Avg. No. of Questions	Avg. Question Time	Avg. No. of Relaxation	Avg. Relaxation Time
Random-Simple	20.176	0.661	5.235	1.265
Const-Simple	17.474	0.376	4.059	0.809
MaxBranch-Simple	15.417	136.999	3.222	0.441

Table 41: Statistics for question selection for the 2-destination problems.

In terms of average relaxation candidate selection time, one-factor ANOVA shows that the differences are significant (*p=0.001*). Even though the Const-Simple method is more efficient than the Random-Simple method, *t*-test does not yield significant differences. However, the MaxBranch-Simple method is significantly more efficient than either the Const-Simple method (*p=0.003*) or the Random-Simple method (*p<0.001*). The observed relations between the methods correspond consistently with the observations with the average relaxation candidates.

Table 42 presents the performance statistics for the 3-destination problems. We observe that the MaxBranch-Simple method is more efficient than the Const-Simple method, which is in turn more efficient than the Random-Simple method in terms of the average number of questions requested by the system to solve the problems. One-factor analysis of variance, however, does not show that the differences are significant.

	Avg. No. of Questions	Avg. Question Time	Avg. No. of Relaxation	Avg. Relaxation Time
Random-Simple	20.200	0.732	5.733	1.187
Const-Simple	16.938	0.392	3.231	0.585
MaxBranch-Simple	15.000	136.623	2.938	0.434

Table 42: Statistics for question selection for the 3-destination problems**.**

In terms of average question selection time, we notice significant differences between the methods. In particular, the MaxBranch-Simple method is much more expensive than either the Random-Simple method or the Const-Simple method. One-factor analysis of variance shows that the differences are significant (*p<0.001*). Pairwise *t*-tests show that the Const-Simple method is significantly more efficient than the Random-Simple method (*p=0.004*), and that both the Random-

Simple method and the Const-Simple method are significantly more efficient than the MaxBranch-Simple method ($p<0.001$).

In terms of average number of relaxation candidate selections, one-factor ANOVA shows that the differences are significant ($p=0.026$). Even though the MaxBranch-Simple method is more efficient than the Const-Simple method, *t*-test does not yield significant differences. However, both the MaxBranch-Simple method and the Const-Simple method are significantly more efficient than the Random-Simple method ($p=0.008$ and $p=0.016$, respectively).

In terms of average relaxation candidate selection time, one-factor ANOVA shows that the differences are significant ($p=0.007$). Even though the MaxBranch-Simple method is more efficient than the Const-Simple method, *t*-test does not yield significant differences. However, both the MaxBranch-Simple method and the Const-Simple method are significantly more efficient than the Random-Simple method ($p=0.002$ and $p=0.013$, respectively). The observed relations between the methods correspond consistently with the observations with the average relaxation candidates.

7.4.2. Task complexity and task success

Table 43 presents the kappa statistics for each of the methods for solving both the 2-destination and the 3-destination problems. For the 2-destination problems, the MaxBranch-Simple method demonstrates the highest success rate, while the Random-Simple method the lowest success rate. For the 3-destination problems, the highest success rate is achieved through the Const-Simple method, while the lowest success rate is achieved though the MaxBranch-Simple method. One-factor analysis of variance shows that the differences between the methods are not significant regardless of the destination types.

	2-destinations	3-destinations
Random-Simple	0.746	0.831
Const-Simple	0.784	0.850
MaxBranch-Simple	0.812	0.829

Table 43: Kappa statistics for the 2-destination and 3-destination problems.

7.4.3. Task complexity and relaxation efficiency

To compare the effectiveness of the relaxation selection methods, we first evaluate the Random-Simple method, the Random-Const method, and the Random-MinSize method, with the Random-

Simple method as the baseline and the question selection method fixed as the random question selection method. In addition, we compare the Const-Simple method and the Const-Const method with the question selection fixed as the constraint hierarchy based approach, and we compare the MaxBranch-Simple method and the MaxBranch-MinSize method with the maximum branching heuristic fixed as the question selection method.

Table 44 presents the statistics of relaxation selection for the 2-destination problems. We observe that the Random-Simple method is more efficient than the Random-Const method, which is in turn more efficient than the Random-MinSize method in terms of average number of relaxation candidates selected by the system for solving the over-constrained problems. One-factor analysis of variance shows that the differences are significant ($p=0.002$). Specifically, even though the Random-Const method is in general more efficient than the Random-MinSize method, *t*-test shows that the difference between them is, however, not significant. The Random-Simple method, however, is significantly more efficient than both the Random-Const method ($p=0.008$) and the Random-MinSize method ($p<0.001$).

	Avg. Relaxation	Avg. Relaxation Time
Random-Simple	5.059	0.568
Random-Const	6.550	1.084
Random-MinSize	7.600	7.366
Const-Simple	4.059	0.809
Const-Const	4.111	0.771
MaxBranch-Simple	3.222	0.441
MaxBranch-MinSize	5.294	3.801

Table 44: Statistics of relaxation candidate selection for the 2-destination problems

In terms of average relaxation selection time, we notice significant differences between the methods. In particular, the Random-MinSize method is much more expensive than either the Random-Simple method or the Random-Const method. One-factor analysis of variance shows that the differences are significant ($p<0.001$). Pairwise *t*-tests also show that the Random-Simple method is significantly more efficient than the Random-Const method ($p=0.003$), and the Random-Const method is significantly more efficient than the Random-MinSize method ($p<0.001$).

When we compare the Const-Simple method and the Const-Const method in terms of average number of relaxation selections, we observe that the Const-Simple method is somewhat more efficient than the Const-Const method. The difference, however, is not statistically significant. In

terms of average time used for relaxation candidate selection, the Const-Const method is somewhat more efficient. Again, the difference is not statistically significant.

When we compare the MaxBranch-Simple method and the MaxBranch-MinSize method in terms of average number of relaxation selections, we observe that the MaxBranch-Simple method is more efficient than the MaxBranch-MinSize method. The difference is statistically significant ($p=0.001$). In terms of average time used for relaxation candidate selection, the MaxBranch-Simple method is again significantly more efficient than the MaxBranch-MinSize method.

When we take the best method from each group (i.e., the Random-Simple method, the Const-Simple method, and the MaxBranch-Simple method), we observe that the MaxBranch-Simple method is more efficient than the Const-Simple method, which is in turn more efficient than the Random-Simple method in terms of the average number of relaxation candidates. Pairwise *t*-tests show that the MaxBranch-Simple method significantly outperforms either the Const-Simple method ($p=0.001$) or the Random-Simple method ($p<0.001$), while the difference between the Const-Simple method and the Random-Simple method is not significant. When we compare the three methods in terms of the relaxation time, we observe that the MaxBranch-Simple method is significantly more efficient than both the Const-Simple method and the Random-Simple method ($p=0.003$ and $p=0.003$, respectively). The difference between the Const-Simple method and the MaxBranch-Simple method is not significant.

	Avg. No. of Relaxation	Avg. Relaxation Time
Random-Simple	5.733	1.187
Random-Const	5.100	1.288
Random-MinSize	5.211	4.045
Const-Simple	3.231	0.585
Const-Const	3.438	0.693
MaxBranch-Simple	2.938	0.434
MaxBranch-MinSize	3.750	1.928

Table 45: Statistics of relaxation candidate selection for the 3-destination problems.

Table 45 presents the statistics of relaxation candidate selection for the 3-destination problems. We observe that for the random question selection method, the Random-Const method is the most efficient, while the Random-Simple method is the less efficient, with the Random-MinSize method in the middle. The differences between these three methods, however, are not significant. In terms of system time, the Random-Simple method is significant more efficient than the Random-Const

method (*p=0.017*) and the Random-Const method is significant more efficient than the Random-MinSize method (*p<0.001*).

When we compare the Const-Simple method and the Const-Const method in terms of the average number of relaxation selections, we observe that the Const-Simple method is somewhat efficient than the Const-Const method. The difference, however, is not statistically significant. In terms of the average time used for relaxation candidate selection, the Const-Simple method is again somewhat more efficient. Again, the difference is not statistically significant.

When we compare the MaxBranch-Simple method and the MaxBranch-MinSize method in terms of the average number of relaxation selections, we observe that the MaxBranch-Simple method is somewhat efficient than the MaxBranch-MinSize method. The difference, however, is not statistically significant. In terms of average time used for relaxation candidate selection, the MaxBranch-Simple method is significantly more efficient than the MaxBranch-MinSize method (*p=0.001*).

When we compare between the Random-Simple method, the Const-Simple method and the MaxBranch-Simple method in terms of the average number of relaxation candidates, we observe that the MaxBranch-Simple method is more efficient than the Const-Simple method, which is in turn more efficient than the Random-Simple method. Pairwise *t*-tests show that both the Const-Simple method and the MaxBranch-Simple method significantly outperforms the Random-Simple method (*p=0.016* and *p=0.008*, respectively). The difference between the Const-Simple method and the MaxBranch-Simple method is, however, not significant. When we compare the three methods in terms of the relaxation time, we observe that both the Const-Simple method and the MaxBranch-Simple method are significantly more efficient than the Random-Simple method (*p=0.013* and *p=0.002*, respectively). The difference between the Const-Simple method and the MaxBranch-Simple method is not significant.

7.4.4. Task complexity: summary

To summarize, when different question selection strategies are compared, the MaxBranching-Simple method is in general more efficient than the Const-Simple method, which is in turn more efficient than the baseline Random-Simple method in terms of the average number of questions requested by the system. In terms of system time, however, the Const-Simple method is the most efficient method and the MaxBranching-Simple method is the most expensive method. Minimizing

the average number of questions reduces the turn-taking efforts and communication costs during human-computer interaction. Minimizing the average system time for question selection improves the system response time. Deciding on which aspect of interaction that the system needs to optimize involves balancing the trade-off.

In terms of task success, we do not observe any significant advantage from any of the methods. This suggests that the effectiveness of these heuristic methods contribute more to the efficiency aspect of human-computer interaction, rather than task success.

In terms of relaxation efficiency, with the question selection methods fixed as the random selection method, the constraint-hierarchy based method, or the maximum branching based method, the baseline simple chronological backtracking method consistently requires fewer number of relaxation candidates than either the constraint-hierarchy based relaxation selection method or the minimum solution size based method. The baseline backtracking method is also more efficient in terms of system time for obtaining relaxation candidates. When we look at the interaction between the question selection methods and the relaxation selection methods, we observe that the question selection methods dominate the relaxation efficiency factor: i.e., efficient question selection methods tend to give rise to efficient relaxation behaviors. For example, the constraint-hierarchy based question selection method and the maximum branching based method are both more efficient than the random question selection method; subsequently, these two methods also give rise to more efficient relaxation behavior in terms of the number of relaxation candidates required. This suggests that when the question selection method is fixed, additional knowledge in deciding relaxation candidates does not make much contribution compared to the baseline. It could be that all the necessary knowledge has already been built into the network structure; therefore, more sophisticated approaches will not give rise to any additional benefit, rather than incurring more cost on the part of the system.

7.5. Summary and Implications for Usability Testing

In this chapter, we have evaluated different heuristics for question selection and relaxation candidate selection at a variety of problem settings specified by task difficulty, goal-state size, interval size, and task complexity. Now we summarize our findings with an emphasis on identifying the most efficient and effective heuristics that generalize over various problem settings.

Table 46 through Table 50 summarize the experimental results from earlier sections of this chapter. In each table, the first column specified the main parameter setting of the experiments, the second column specifies the values of the parameter settings or sub-parameter settings. The third column lists the methods compared, and the fourth column lists method pairs that have demonstrated to be significantly different based on paired sample *t*-tests.

As we can see form these summaries, across different problem settings, the Const-Simple method and the FixedOrder-Simple methods are the best two heuristics for achieving higher task success scores (Table 46).

In terms of reducing the number of questions requested by the system for under-constrained situations, the Const-Simple method and the MaxBranch-Simple method are generally more efficient than the other methods, with the MaxBranch-Simple more efficient than the Const-Simple method (Table 47). In terms of the system time required for selecting the questions, however, the MaxBranch-Simple method takes significantly more time than the Random-Simple method and the Const-Simple method. The Const-Simple method is significantly more efficient than the Random-Simple method for certain one-leg problems and for the more complex 2-destination and 3-destination problems (Table 48).

In terms of reducing the number of relaxation candidates suggested by the system for over-constrained situations (Table 49), when the question selection method is fixed, knowledge sources such as constraint hierarchy and solution status do not improve efficiency. When the relaxation candidate selection method is fixed, the more efficient the question selection method is, the more efficient relaxation is. For example, the MaxBranch-Simple method and the Const-Simple method are generally more efficient than the Random-Simple method. This suggests that relaxation efficiency is dominated primarily by the question selection method. Similar trends are observed for the system time required for selecting the relaxation candidates (Table 50).

The above observations suggest that when designing a cooperative information-seeking dialogue system, what heuristic methods are appropriate for generating initiative-taking actions depends on which aspect of performance (e.g., task success, question selection efficiency, relaxation candidate selection efficiency) that the system designer aims to optimize. When all aspects are considered, we have observed that the heuristic that based on user's preference strength (i.e., the Const-Simple method) produces the best overall result: better task success scores and improved dialogue efficiency. Therefore, we select this heuristic in the usability study to be presented in Chapter 9.

Parameters	Parameters/Sub-parameters values	Methods compared	Significant differences observed (paired sample *t*-tests)
Task difficulty(T)	T={Easy,Medium,Hard}	Random-Simple FixedOrder-Simple Const-Simple MaxBranch-Simple MaxInfo-Simple	Const-Simple>Random-Simple FixedOrder-Simple>Random-Simple
	T={Easy}	Same as above	
	T={Medium}	Same as above	
	T={Hard}	Same as above	Const-Simple>Random-Simple FixedOrder-Simple>Random-Simple
Goal-state (G)	T={Easy} G={1,5,20} unless specified otherwise	Random-Simple FixedOrder-Simple Const-Simple MaxBranch-Simple MaxInfo-Simple	Const-Simple>Random-Simple FixedOrder-Simple>Random-Simple
	T={Medium} G={1,5,20} unless specified otherwise	Same as above	Const-Simple>Random-Simple (G={1,20}) FixedOrder-Simple>Random-Simple Random-Simple>MaxInfo-Simple (G={20}) MaxBranch-Simple>Random-Simple (G={5})
	T={Hard} G={1,5,20} unless specified otherwise	Same as above	Const-Simple>Random-Simple FixedOrder-Simple>Random-Simple Random-Simple>MaxInfo-Simple (G={5,20})
Interval size (I)	I=1hr	Random-Simple Const-Simple MaxBranch-Simple	
	I=4hr	Same as above	
Complexity (C)	C=3-destinations	Random-Simple Const-Simple MaxBranch-Simple	
	C=2-destinations	Same as above	

Table 46: Performance of different methods in terms of task success scores at different parameter settings. “>” means significantly higher.

Parameters	Parameters/Sub-parameters values	Methods compared	Significant differences observed (paired sample *t*-tests)
Task difficulty(T)	T={Easy,Medium,Hard} unless specified otherwise	Random-Simple FixedOrder-Simple Const-Simple MaxBranch-Simple MaxInfo-Simple	MaxBranch-Simple<Random-Simple Const-Simple<Random-Simple Random-Simple<FixedOrder-Simple Random-Simple<MaxInfo-Simple (T={Medium})
Goal-state (G)	T={Easy} G={1,5,20} unless specified otherwise	Random-Simple FixedOrder-Simple Const-Simple MaxBranch-Simple MaxInfo-Simple	MaxBranch-Simple<Random-Simple Const-Simple<Random-Simple Random-Simple<FixedOrder-Simple Random-Simple<MaxInfo-Simple (G={20})
	T={Medium} G={1,5,20} unless specified otherwise	Same as above	MaxBranch-Simple<Random-Simple Const-Simple<Random-Simple MaxBranch-Simple<Const-Simple Random-Simple<FixedOrder-Simple Random-Simple<MaxInfo-Simple (G={5,20})
	T={Hard} G={1,5,20} unless specified otherwise	Same as above	MaxBranch-Simple<Random-Simple Const-Simple<Random-Simple Random-Simple<FixedOrder-Simple Random-Simple<MaxInfo-Simple (G={20})
Interval size (I)	I=1hr	Random-Simple Const-Simple MaxBranch-Simple	Const-Simple<Random-Simple MaxBranch-Simple<Random-Simple MaxBranch-Simple<Const-Simple
	I=4hr	Same as above	Same as above
Complexity (C)	C=2-destinations	Random-Simple Const-Simple MaxBranch-Simple	MaxBranch-Simple<Random-Simple MaxBranch-Simple<Const-Simple
	C=3-destinations	Same as above	

Table 47: Performance of different methods in terms of the average number of questions in question selection at different problem settings. “<” means significantly more efficient.

Parameters	Parameters/Sub-parameters values	Methods compared	Significant differences observed (paired *t*-tests)
Task difficulty(T)	T={Easy,Medium,Hard}	Random-Simple FixedOrder-Simple Const-Simple	
		Random-Simple MaxBranch-Simple MaxInfo-Simple	Random-Simple<MaxBranch-Simple Random-Simple<MaxInfo-Simple MaxBranch-Simple<MaxInfo-Simple
Goal-state (G)	T={Easy} G={1,5,20} unless specified otherwise	Random-Simple FixedOrder-Simple Const-Simple	Const-Simple<FixedOrder-Simple Const-Simple<Random-Simple (G={20})
		Random-Simple MaxBranch-Simple MaxInfo-Simple	Random-Simple<MaxBranch-Simple Random-Simple<MaxInfo-Simple MaxBranch-Simple<MaxInfo-Simple
	T={Medium} G={1,5,20} unless specified otherwise	Random-Simple FixedOrder-Simple Const-Simple	
	T={Hard} G={1,5,20} unless specified otherwise	Same as above	Const-Simple<FixedOrder-Simple (G={5}) Const-Simple<Random-Simple (G={20}) FixedOrder-Simple<Random-Simple (G={20})
Interval size (I)	I=1hr	Random-Simple Const-Simple MaxBranch-Simple	Const-Simple<Random-Simple Random-Simple<MaxBranch-Simple Const-Simple<MaxBranch-Simple
	I=4hr	Same as above	Same as above
Complexity (C)	C=2-destinations	Random-Simple Const-Simple MaxBranch-Simple	Const-Simple<Random-Simple Random-Simple<MaxBranch-Simple Const-Simple<MaxBranch-Simple
	C=3-destinations	Same as above	Same as above

Table 48: Performance of different methods in terms question selection time in question selection at different problem settings. "<" means significantly more efficient.

Parameters	Parameters/Sub-parameters values	Methods compared	Significant differences observed (paired sample *t*-tests)
Task difficulty(T)	T={Easy,Medium,Hard} unless specified otherwise	Random-Simple Random-Const Random-MinSize	Random-Simple<Random-MinSize
		Const-Simple Const-Const	
		MaxBranch-Simple MaxBranch-MinSize	
		Random-Simple Const-Simple MaxBranch-Simple	Const-Simple<Random-Simple MaxBranch-Simple<Random-Simple
		Random-Const Const-Const	Const-Const<Random-Const
		Random-MinSize MaxBranch-MinSize	MaxBranch-MinSize<Random-MinSize
Goal-state (G)	T={Easy} G={1,5,20} unless specified otherwise	Random-Simple Random-Const Random-MinSize	Random-Simple<Random-MinSize Random-Const<Random-MinSize (G={1,5})
	T={Medium} G={1,5,20} unless specified otherwise	Same as above	
	T={Hard} G={1,520} unless specified otherwise	Same as above	
Interval size (I)	I=1hr	Random-Simple Const-Simple MaxBranch-Simple	Const-Simple<Random-Simple MaxBranch-Simple<Random-Simple
	I=4hr	Same as above	Const-Simple<Random-Simple MaxBranch-Simple<Random-Simple MaxBranch-Simple<Const-Simple
Complexity (C)	C=2-destinations	Random-Simple Random-Const Random-MinSize	Random-Simple<Random-Const Random-Simple<Random-MinSize
		Const-Simple Const-Const	
		MaxBranch-Simple MaxBranch-MinSize	MaxBranch-Simple<MaxBranch-MinSize
		Random-Simple Const-Simple MaxBranch-Simple	MaxBranch-Simple<Const-Simple MaxBranch-Simple<Random-Simple
	C=3-destinations	Random-Simple Random-Const Random-MinSize	
		Const-Simple Const-Const	
		MaxBranch-Simple MaxBranch-MinSize	
		Random-Simple Const-Simple MaxBranch-Simple	Const-Simple<Random-Simple MaxBranch-Simple<Random-Simple

Table 49: Performance of different methods in terms of the average number of relaxation candidates at different problem settings. "<" means significantly more efficient.

Parameters	Parameters/Sub-parameters values	Methods compared	Significant differences observed (paired sample *t*-tests)
Task difficulty(T)	T={Easy,Medium,Hard}	Random-Simple Random-Const Random-MinSize	Random-Simple<Random-Const Random-Const<Random-MinSize
Goal-state (G)	T={Easy} G={1,5,20} unless specified otherwise	Random-Simple Random-Const Random-MinSize	Random-Simple<Random-MinSize Random-Const<Random-MinSize
	T={Medium} G={1,5,20} unless specified otherwise	Same as above	Random-Simple<Random-Const (G={1,5}) Random-Simple<Random-MinSize Random-Const<Random-MinSize
	T={Hard} G={1,5,20} unless specified otherwise	Same as above	Random-Simple<Random-MinSize Random-Const<Random-MinSize
Interval size (I)	I=1hr	Random-Simple Const-Simple MaxBranch-Simple	Random-Simple<MaxBranch-Simple Const-Simple<MaxBranch-Simple Const-Simple<Random-Simple
	I=4hr	Same as above	Random-Simple<MaxBranch-Simple Const-Simple<MaxBranch-Simple Const-Simple<Random-Simple
Complexity (C)	C=2-destinations	Random-Simple Random-Const Random-MinSize	Random-Simple<Random-Const Random-Const<Random-MinSize Random-Simple<Random-MinSize
		Const-Simple Const-Const	
		MaxBranch-Simple MaxBranch-MinSize	MaxBranch-Simple<MaxBranch-MinSize
		Random-Simple Const-Simple MaxBranch-Simple	MaxBranch-Simple<Random-Simple MaxBranch-Simple<Random-Simple
	C=3-destinations	Random-Simple Const-Simple MaxBranch-Simple	Random-Simple<Random-Const Random-Const<Random-MinSize
		Const-Simple Const-Const	
		MaxBranch-Simple MaxBranch-MinSize	MaxBranch-Simple<MaxBranch-MinSize
		Random-Simple Const-Simple MaxBranch-Simple	Const-Simple<Random-Simple MaxBranch-Simple<Random-Simple

Table 50: Performance of different methods in terms of the average system time for relaxation candidate selection at different problem settings. "<" means significantly more efficient.

Chapter 8 Architecture of a Cooperative Mixed-Initiative Information Dialogue System

In this chapter, we describe a prototype form-based cooperative mixed-initiative (COMIX) information system. In the following sections, we first present the task and the application domain. Then we present the architecture of the mixed-initiative information system, specifying the interactions between the components. We present in detail the components of the information system, emphasizing especially the dialogue manager and the initiative-taking dialogue actions that are supported by the system. Lastly, we trace through an example form-based dialogue between the system and the user.

8.1. Task Description

The architecture of the cooperative mixed-initiative information dialogue system is illustrated by examples taken from the airline flight access domain. The user has an information need to find flights between cities subject to certain travel preferences and restrictions. The system takes in the user's information need and provides answers to satisfy the user's requests. Several features of this goal-oriented task make it a good domain at the right level of task complexity:

- In order to find the flights that meet the user's requirements and preferences, the system may need to engage in extended interactions with the user over multiple turns.
- The user's information requests may be over-constrained (i.e., the user's constraints cannot be satisfied by the real world availability of the flights), which offers a cooperative dialogue system the opportunity for constraint conflict resolution.
- The user's information request may involve multi-leg flight access, which increases the number of constraints and subsequently the complexity of the task.

8.2. The Airline Domain Model

The domain model refers to a declarative description of the objects and relations in a particular domain. We use the standard Entity-Relation (ER) approach (Chen, 1976) for specifying the conceptual model for a domain. The ER approach diagrammatically captures a high-level understanding of the data independent of its storage in the database. I adopt this model because a

conceptual model of the data is easy to understand and ER diagrams are widely used in the information systems community.

We have constructed a partial simplified airline domain model based on flight schedule information downloaded from the web site of Northwest Airlines (www.nwa.com). The downloaded data, which consists of detailed information for flights (such as departure city, arrival city, airplane type, etc.), is stored in a relational database. In order to increase the variability of the data, we introduce two other airlines – United Airlines and Delta Airlines – into the data, by reassigning one-third of the original flights to United Airlines, reassigning one-third of the original flights to Delta Airlines, and leaving the rest to Northwest Airlines.

8.2.1. ER diagram

The basic concepts in ER diagrams include entities, attributes, and relationships. Entities are the basic objects of an ER model representing objects in the world and abstract objects. Attributes are properties that describe entities. Entities are related with each other through relationships. Relationships may vary in the number of participating entities. The graphical representation follows the notation as specified in (Chen, 1976). In general, rectangles represent entities, and links between entities represent relationship between entities, with the numbers at the ends of the links specifying the relationship types. Relationship between entities can be 1-to-1, 1-to-many, or many-to-many.

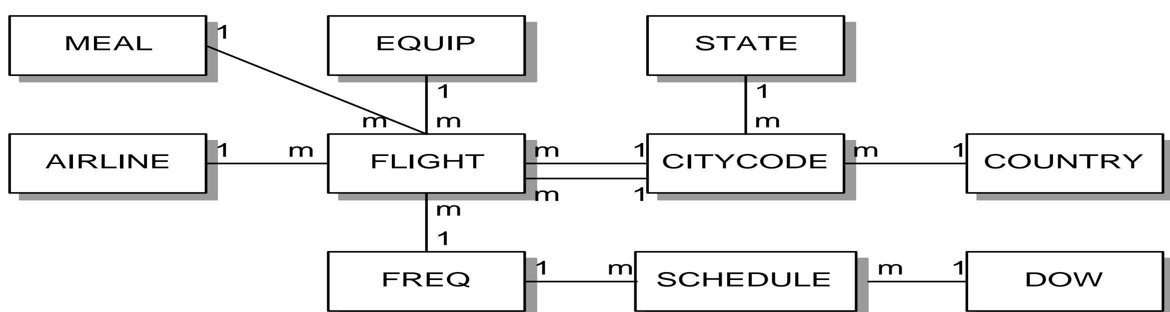


Figure 39: ER diagram for the airline domain. Each rectangle represents an entity and each link represents a relation between entities. The numbers on the links specify the cardinality of the relations.

The ER diagram in Figure 39 represents the model of the flight information domain used in the prototype system. In this model, we make several assumptions about the domain. Specifically, we assume the following business rules (that are all expressed in the ER diagram):

- An airline has many flights; a flight belongs to an airline.
- Each flight has only one departure city and one arrival city; a city can be a departure city or an arrival city for many flights.
- Each flight has one equipment type; an equipment type can be used by many flights.
- Each flight serves at most one meal; meals can be served on many flights.
- A city has one state code and one country code; a state or a country can have many cities.
- Frequency of flights is daily unless noted by the specified frequency symbols. The frequency symbols include:

X – Except	1 – Monday	2 – Tuesday	3 - Wednesday
4 – Thursday	5 – Friday	6 – Saturday	7 - Sunday

For example, the frequency code “246” means flights are available on Tuesday, Thursday, and Saturday, while the frequency code “x67” means flights are available any day of the week except Saturday and Sunday.

- Each flight has either no frequency code (meaning daily flight) or exactly one frequency code. Each frequency code can be used by many flights.
- Each frequency code can refer to many days of week. A day of week can be found in many frequency codes. The many-to-many relationship between the frequency code and day-of-week is represented by the entity SCHEDULE.

8.2.2. Relational model

The ER diagram in Figure 39 can be represented using a relational model. In the following, we present the entities and the relationships together with their respective attributes, with the primary keys[9] underlined and the foreign keys[10] marked with @.

FLIGHT (AirlineCode@, FlightNumber, DepartCityCode@, ArriveCityCode@, DepartTime, ArriveTime, Stops, Eqp@, FreqCode@, MealCode@, Mileage)

AirlineCode: the airline, foreign key to AIRLINE

[9] A primary key is a unique identifier for a relation. A primary key can be defined using one attribute or combinations of attributes.
[10] A foreign key of a relation *R2* is a subset of attributes of *R2*, such that there exists a relation *R1* with a primary key and each value of the foreign key of *R2* is identical to the value of the primary key of *R1*.

FlightNumber: flight number

DepartCityCode: the city code of the departure city, foreign key to CITYCODE
ArriveCityCode: the city code of the arrival city, foreign key to CITYCODE
DepartTime: departure time
ArriveTime: arrival time
Stops: the number of stops, the default value is 0, can be 1 or more
FreqCode: frequency of flight, foreign key to FREQ
MealCode: the type of meals served, foreign key to MEAL
Eqp: the type of aircraft, foreign key to EQUIP

Mileage: a numerical value specifying the approximate mileage between the departure city and the arrival city

AIRLINE (AirlineCode, AirlineName)

AirlineCode: the code name of the airline
AirlineName: the name of the airline

FREQ (FreqCode)

This is a code table. The attribute specifies a list of frequency codes.

DOW (day_of_week)

This is a code table. The attribute specifies the seven days in a week, i.e., Monday through Sunday.

SCHEDULE (FreqCode@, DayOfWeek@)

This table specifies what days of week are included in each frequency code.
FreqCode: the frequency code, primary key, foreign key to FREQ.
DayOfWeek: the day of week, primary key, foreign key to DOW.

CITY_CODE (CityCode, CityName, State@, ZipCode, Country@)

CityCode: the city code, primary key
CityName: the name of the city
State: the state or province where the city is located, foreign key to STATE

ZipCode: the zip code of the city
Country: the name of the country, foreign key to COUNTRY
An alternative candidate key is the composite key: (CityName, State, Country).

STATE (StateName)

This is a code table, which specifies a list of state or province names.

COUNTRY (CountryName)

This is a code table, which specifies a list of country names.

MEAL (MealCode)

This is a code table, which includes four values:

B – Breakfast	L – Lunch	D – Dinner	S – Snack

EQUIP (Equipment)

This is a code table, which a list of possible aircraft types.

8.3. The Architecture of an Initiative-Taking Information Dialogue System

The goal of the implemented prototype initiative-taking information system is to engage in a conversation with a user in a cooperative way to satisfy the user's requests. In information access systems, a user request is often a query for a piece of information from a database. The system needs to understand the user's requests, know when to take the initiative, particularly for under-specified queries or for over-constrained queries that no database records can satisfy, and provide appropriate information to the user.

Figure 40 illustrates the flow of data and control through our model of a mixed-initiative information system and the constraint-based problem solving components (within the dotted box) in particular. A user's input is first parsed into a semantic representation form, which is passed to the dialogue manager. The dialogue manager processes the information request, and decides what to do by testing two cases: (1) whether to respond to the utterance immediately, or (2) whether to

send constraints to the constraint-based problem solver for processing. The process goes to case (2) only if there is no need for case (1).

Case (1) is concerned with checking missing values (e.g., no input value for the departure city), determining value ambiguity (e.g., the city name "*Rochester*" representing either "*Rochester, MN*" or "*Rochester, NY*"), and checking errors (e.g., an input city is not found in the database). With case (1), the system may sometimes need to consult the database to detect ambiguity or invalid values (e.g., detecting the ambiguity with the city name "*Rochester*" or checking whether a city name is valid in the database). This interaction with the database is represented as the links between the dialogue manager and the DBMS outside the dotted rectangle.

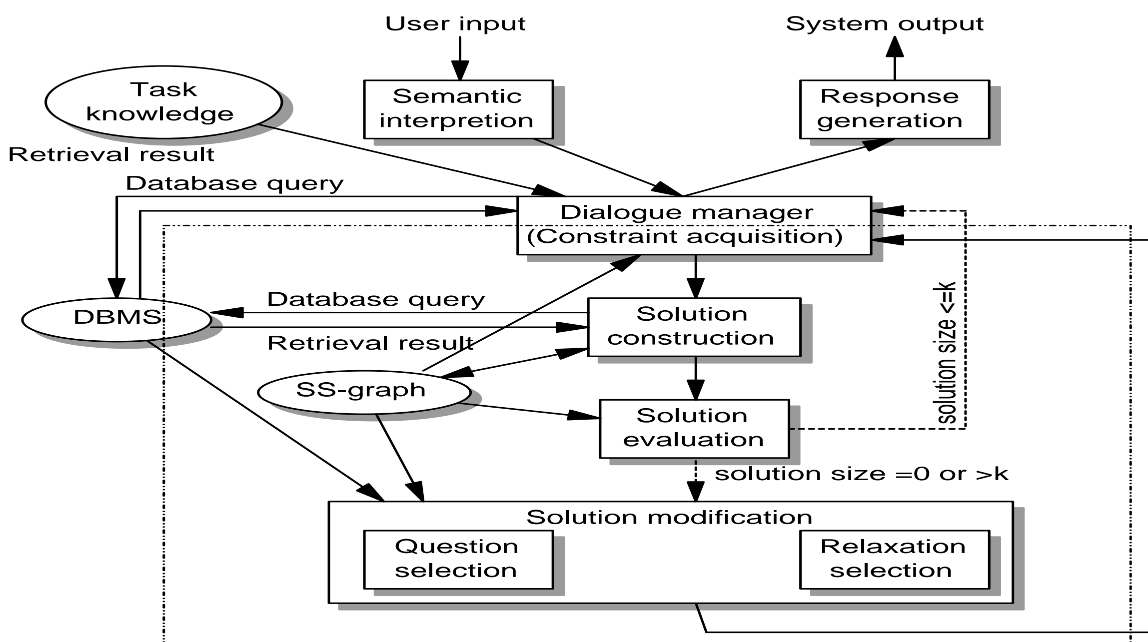


Figure 40: Architecture of the COMIX information system. The rectangles represent modules and the ovals represent stored data. The components within the dashed rectangle are concerned with constraint-based problem solving. The solid lines with arrows represent data flow and the dashed lines with arrows represent control flow.

With case (2), the constraints to the problem solver will be incorporated into the dynamically adapted solution graph (i.e., SS-graph) to yield a new or updated solution set, which is in turn evaluated. Depending on the solution evaluation status, the problem solver decides whether to exploit knowledge sources in the solution space along with the domain constraints in the DBMS to identify relaxation or restriction candidates for over-constrained or under-constrained situations. If the solution size is equal to or smaller than a pre-determined number *k*, then the solution set is passed to the dialogue manager. If the solution size is greater than *k*, then an under-constrained

situation is detected and the question selection module is invoked for identifying attributes to further constrain the problem definition. If the solution size is empty, then an over-constrained situation is detected and the relaxation selection module is invoked. The solution set, together with possible relaxation or restriction candidates, is passed to the dialogue manager, which selects appropriate dialogue actions. The selected dialogue action is then realized through natural language output or through form-based objects to the user. In our prototype information system, user input and system output are communicated through form-based objects. It is worth mentioning, however, that the architecture in Figure 40 is a general architecture that does not restrict the type of input and output. For instance, the input and output could as well be natural language utterances in natural language dialogue systems. In the following subsections, we present the details of the major components in the architecture.

8.3.1. Task knowledge representation

In our system, the domain-dependent knowledge about a task is organized as a hierarchy of objects. The hierarchy defines the relationships between the objects in the task domain. Each object has a header (i.e., the name of the object), a body (i.e., a list of variables and their associated values specific to the object), and some methods associated with it (mostly methods calling the dialogue motivators). A simplified task knowledge hierarchy used in the airline domain for retrieving one-way flights is shown in Figure 41 (the associated methods for these objects are left out of the figure). For instance, in this hierarchy, the AirlineAccessOneWay object is comprised of objects that are required for this application, such as the DeptCity object for the departure city information, the DeptCityStrength object for the preference strength of the attribute DeptCity having certain value, and the CarrierOption object for the carrier choices. Retrieving round-trip flights and multi-leg flights employs independent but similar object hierarchies.

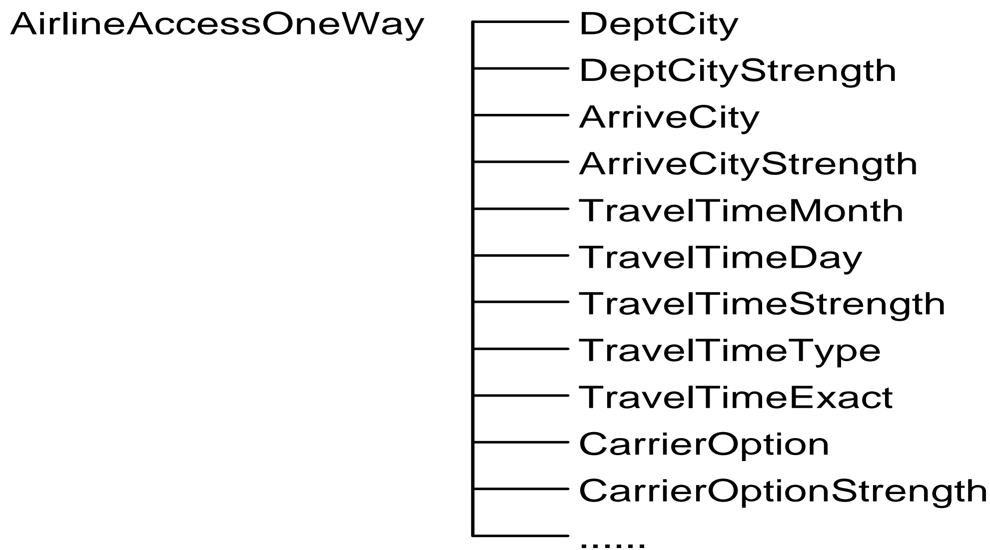


Figure 41: The task knowledge hierarchy for retrieving one-way flights. Each node represents an object in the knowledge hierarchy.

In COMIX, the task knowledge objects are implemented as form-based objects such as text boxes, combo boxes, etc[11]. (The reader is referred to any book on programming for Windows applications for the definitions and usage of these form-based objects, e.g., *Microsoft Visual Basic 5.0 Programmer's Guide*, 1997). Each form-based object also has a header, a body, and some methods associated with it. For example, in Figure 42, the task knowledge object AirlineAccessOneWay is represented as a form object frmAirlineAccessOneWay, which contains other objects encoding task knowledge such as the combo object cboDeptCity as DeptCity, the combo object cboDeptCityStrength as the preference strength for the DeptCity attribute having certain value, and the option group object optAirlineOption as CarrierOption.

8.3.2. Semantic interpretation

In the prototype system, the semantic interpretation component receives user input through form objects (e.g., text boxes, options) and generates a set of labeled constraints (introduced earlier in section 3.1). For example, the instantiated form in Figure 43 represents a user's request for a one-way flight from Beijing to Pittsburgh preferably on June 20th with Northwest Airlines. The user requests that the constraints on the departure city, the arrival city be required, the travel date and

[11] Generally speaking, *command buttons* are used to perform actions, *labels* to display text, *text boxes* for users to enter or edit text, *check boxes* of a small set of choices for users to choose one or more options, *option buttons* of a small set of options for users to choose just one option, *list boxes* of a scrollable list of choices for users to choose from the list, and *combo boxes* of a scrollable list of choices for uses to choose from the list or type a new value.

travel time be strongly preferred, and the airline carrier be medium preferred. The semantic interpretation component generates a conjunction of labeled constraints for this form as follows:

DeptCity = “PEK”, required
ArriveCity = “PIT”, required
TravelDate = #June-20-2001#, strong
TravelTimeType = “arrival-time”
TravelTime < #8:00pm#, strong
CarrierOption = “Northwest”, medium

The labeled constraints are used by the constraint-based problem-solving model for constructing solutions or for constructing queries in the structured query language (SQL) for information access from the domain database.

8.3.3. Problem-solving components

Our constraint-based model of information systems supports the functionality 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 supported by the constraint-based problem solver. Through solution evaluation, under-constrained or over-constrained situations can be identified immediately during the problem formulation and problem solving process. Through solution modification, the information system can use its knowledge about the solution states to identify restriction or relaxation candidates, which in turn support the adoption of certain cooperative dialogue actions (e.g., RequestVal and ProposeNewVal to be introduced in section 8.3.4.3) for resolving under-constrained or over-constrained situations. Repeating the cycles allows the information system to help the users with their problem formulation until a satisfying solution is found.

8.3.3.1. Constraint acquisition

In the constraint acquisition phase, the system acquires constraints to update the problem definition that is modeled as a CSP. The system gathers constraints via the dialogue manager through (1) accepting constraints from user input, (2) requesting constraints from the user, or (3) proposing constraints for the 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 parameter instantiations from the solution modification module; the module uses knowledge of the solution status and the domain constraints in the DBMS. The user's response to system-initiated requests may result in new constraints being added. System-initiated proposals need to be negotiated with the user before they are finalized in the problem definition. Acquired constraints are used to update the existing CSP for constructing refined solutions.

Figure 42: Screen shot of the top-level form for retrieving one-way flights, with some of the form-based objects annotated with their names following the arrows.

Figure 43: Screen shot of an instantiated one-way flight request.

In NL-based dialogue systems, the preference strengths of the acquired constraints can be deduced based on the linguistic cues in a user's utterances and the conversational circumstances in which the utterances occur (section 6.4.3). In form-based dialogue systems, the system could provide interfaces for the user to explicitly specify their preference strengths for certain constraints, thus acquiring constraint strengths directly from user input.

8.3.3.2. Solution construction

We have proposed using solution synthesis techniques to generate all solutions to a CSP by iteratively combining partial answers to arrive at a complete list of all correct answers. We have extended the traditional solution synthesis to dynamic CSPs and with constraint hierarchy (section 4.1).

The solution planner determines what variables to include in the solution space and in what order and instantiates the variables with appropriate value bindings through solution synthesis. In the

prototype system, solution construction can be achieved in two ways described in detail below: (1) computing the actual partial solutions at each step or (2) constructing view representations of the partial solutions. Both methods have been implemented, but in the form-based prototype system, we choose the view representation method to save storage space.

The first way is to process the constraints one at a time and then to compute partial solutions for the constraints through SQL queries. During solution synthesis, these partial solutions are combined with the existing partial solutions at the top-level node in the solution synthesis graph to form a set of higher-level partial solutions. If the attributes in the new constraints hold certain dependent relationship with the existing attributes in the solution synthesis graph, then new SQL queries need to be formed to represent the dependent relationship and the new queries need to be executed against the database for filtering out illegal combinations of values. Otherwise, the higher-level partial solutions are simply obtained through solution synthesis without the need to query the database.

For example, suppose the system has collected the following user constraints in the airline domain described in section 8.2: (1) the departure city as "Detroit" ('DTW'), (2) the arrival city as "Pittsburgh" ('PIT'), (3) the departure time is after 5PM, and (4) the airline is Northwest ('NW'). These constraints are represented in the system as:

```
C1: FromCityCode = 'DTW'
C2: ArriveCity = 'PIT'
C3: DepartTime >= #05:00:00 PM#
C4: AirlineCode in ('NW')
```

Figure 44 enumerates the SQLs required for finding solutions that satisfy the above four constraints. Note these SQLs are simplified to remove the reader from system implementation details by eliminating certain attributes that are irrelevant with the example constraints and by excluding the computation of flights with connections. For constraint C1, node 1 is created in the solution synthesis graph (Figure 45). For this node, we introduce SQL1. SQL1 is sent to the database to get all the solutions that satisfy this SQL. For constraint 2, node 2 is created and its corresponding query SQL2 is created. SQL2 is sent to the database to get all the solutions that satisfy this SQL. Then the solutions obtained at both node 1 and node 2 are combined together through solution synthesis to form node 3. Since there is no constraining relationship between FromCityCode and ToCityCode in the domain (refer to the ER diagram and the relational model in section 8.2), and both of them are dependent on the FlightNumber attribute which has been

represented in the partial solutions, we can simply combine the two sets of partial solutions at node 1 and node2 as the solutions at node 3, without the need to access the database again. Similarly, we introduce SQL3 and SQL4 for C3 and C4 at node 3 and node 4, respectively, and combine each set of the partial solutions with the partial solutions at the top node of the solution synthesis graph. Using this approach needs a total of four accesses to the backend database for computing the solutions that satisfy the four constraints.

```
SQL1:  Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
       From tblFlights
       Where FromCityCode = 'DTW'
SQL2:  Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
       From tblFlights
       Where ToCityCode = 'PIT'
SQL3:  Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
       From tblFlights
       Where DepartTime >=#05:00:00 PM#
SQL4:  Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
       From tblFlights
       Where AirlineCode in ('NW')
```

Figure 44: SQLs required for computing solutions for the example airline retrieval problem.

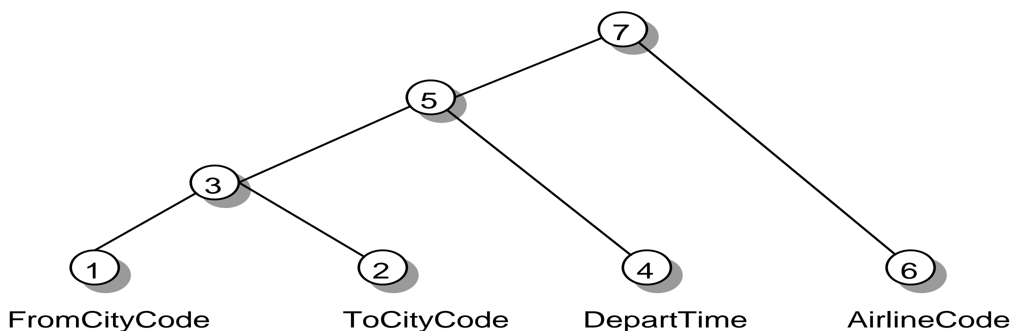


Figure 45: An example solution synthesis graph in the airline domain.

The second way is to make use of *views* to specify how to compute the solutions rather than the solutions themselves. Each constraint is represented as a view, specifying how to compute the partial solutions if needed. The view representation saves storage requirement for the solution synthesis graph by the system and could potentially save database accesses, but will delay detection of over-constrained situations. Computation starts with the final view of the complete problem. The

final solutions are immediately available if the computation of the final view yields solutions. If the problem is over-constrained, then early nodes of views in the solution synthesis graph need to be computed for identifying relaxation candidates. In such cases, additional SQL queries need to be issued to the database for computing the solutions of the selected views of the solution synthesis graph nodes.

Figure 46 enumerates the SQLs required for finding solutions that satisfy the above four constraints using the view representations. For the four user constraints in the earlier example, again we construct a solution synthesis graph (Figure 45). Again, we introduce SQL1 for C1 at node 1 and SQL2 for C2 at node 2. In contrast to the first approach of computing the partial solutions by sending these SQLs to the backend database, we simply record the SQLs as the views of how to compute partial solutions at node 1 and node 2. During solution synthesis at node 3, we combine the two SQLs into a new SQL, i.e., SQL3, by merging the constraint conditions. Similarly, we introduce SQL4 and SQL6 for the base level nodes 4 and 6 for constraints C3 and C4, respectively, and construct the combo node 5 (SQL5) by merging SQL3 and SQL4 and the combo node 7 (SQL7) by merging SQL5 and SQL6. While constructing this solution synthesis graph of structured views, we do not send any SQLs to the backend database. When providing the user with solutions at the top-level node, the view at the top level node (SQL7) is sent to the backend database. If solutions exist, then the solutions are returned, and computation of solutions requires a single access to the backend database. In over-constrained situations in which no solution is found, the system needs to traverse the graph to locate one or more constraints to relax using methods proposed in section 5.2. During the traversal back to the parent nodes (i.e., the lower level nodes), the system computes the partial solutions for these nodes to determine (1) whether the over-constrained situation at the higher-level node results from an over-constrained situation from the parent nodes, and (2) whether relaxation of existing constraints will result in solutions. For (1), the system needs to send the existing SQL at the node to the backend database, and for (2), the system needs to send the updated SQL at the node to the backend database.

```
SQL1:   Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
        From tblFlights
        Where FromCityCode = 'DTW'
SQL2:   Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
        From tblFlights
        Where ToCityCode = 'PIT'
SQL3:   Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
        From tblFlights
```

```
          Where (FromCityCode = 'DTW') AND (ToCityCode = 'PIT')
SQL4:     Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
          From tblFlights
          Where DepartTime >= #05:00:00 PM#
SQL5:     Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
          From tblFlights
          Where (FromCityCode = 'DTW') AND (ToCityCode = 'PIT') AND (DepartTime >= #05:00:00 PM#)
SQL6:     Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
          From tblFlights
          Where AirlineCode in ('NW')
SQL7:     Select AirlineCode, FlightNumber, FromCityCode, ToCityCode, DepartTime, ArriveTime
          From tblFlights
          Where (FromCityCode = 'DTW') AND (ToCityCode = 'PIT') AND (DepartTime >= #05:00:00 PM#)
          AND (AirlineCode in ('NW')
```

Figure 46: A complete set of SQLs recorded as views in the example solution synthesis graph.

During solution synthesis, nodes in the solution synthesis graph are annotated with appropriate strengths as specified in the constraint hierarchy. The constraint hierarchy is dynamic in that users can modify their information requests by adding or dropping certain preferences or restrictions.

8.3.3.3. Solution evaluation

A desirable feature of our framework is that at each stage of incremental problem definition, the system is able to evaluate the solution space and adjust its behaviors accordingly. Solution evaluation characterizes the solution space into different solution situations. Specifically, we use an evaluation function to characterize the solution space. 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 there is no solution to satisfy the user's request. In other words, an over-constrained situation has been detected, which suggests that some constraints need to be relaxed for a solution to be obtained. If the top node evaluates to a set whose number of solutions exceeds a pre-defined threshold *k* (e.g., *k=5* is a commonly used heuristic number in NL dialogue systems and *k=20* in form-based systems), then the problem is under-constrained, which suggests that additional constraints may be required to further refine the existing CSP. If the top node evaluates to a set whose number of solutions is within the pre-defined threshold *k*, then the solution set is small enough, and the system can choose to present the solutions to the user at this point. If the goal of the dialogue is to seek optimal solutions, the system can rank the partial solutions at each state of the problem solving process using a separate optimization evaluation

function. The solution synthesis network provides a unified model for detection of these solution situations.

8.3.3.4. Solution modification

The solution modification module is invoked when under-constrained or over-constrained situations are detected. Its task is to support the generation of initiative-taking dialogue actions for relaxation in over-constrained situations or for restriction in under-constrained situations, which will guide the user with problem formulation. These initiative-taking dialogue actions, some of which we describe later in section 8.3.4.3, are motivated by a corpus analysis of naturally occurring information dialogues. Efficient instantiation of the parameters of these initiative-taking dialogue actions is supported by exploiting knowledge sources represented in the SS-graph. The knowledge sources we exploit in the SS-graph include the constraint hierarchy and the solution network structure. The details on how to exploit the individual knowledge sources or the combination of various knowledge sources to support efficient (linear) parameter instantiation for these initiative-taking dialogue actions have been discussed in Chapter 5.

The system will be considered uncooperative, however, if it modifies the user's information request without the user's consent. Thus cooperative principles require that the proposed modification 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.

8.3.4. Dialogue management

Our approach to dialogue management adopts the concept of dialogue motivators, proposed by Abella et al. (Abella et al., 1996; Abella and Gorin, 1999). Dialogue motivators are general principles that motivate a dialogue. Unlike a procedure-driven finite-state approach, this approach to dialogue management is object-oriented and information-driven. This model of object-oriented dialogue management consists of two major components: *the task knowledge representation* and *a collection of dialogue motivators*. First, the task knowledge specific to an application is encoded in task objects, which can be organized in a hierarchy. Second, the dialogue manager consists of a collection of special objects, called dialogue motivators. These dialogue motivators operate on the task objects and decide what actions to take. Using this model, creating a new application requires defining the hierarchy for encoding task knowledge and designing a set of dialogue motivators.

This object-oriented dialogue management approach applies naturally to my prototype application, as the task objects that define the task knowledge are already defined in the form-based application (section 8.3.1). The dialogue manager only requires additional definitions of dialogue motivators to operate on these form-based objects.

In Abella et al. (1999), a dialogue motivator determines what action the dialogue manager needs to take in conducting its dialogue with a user. In our system, however, a dialogue motivator can be realized through more than one dialogue action depending on the initiative distribution. For example, when there exists ambiguity, the system asks a clarification question when it has dialogue initiative, but takes no action when it does not have dialogue initiative. For initiative tracking, we adopt a simple fixed policy approach; e.g., in the user study to be described in Chapter 9, we compare user-initiative setting with mixed-initiative setting. In the user-initiative setting, we allow the system dialogue initiative at each turn, but no task initiative. In the mixed-initiative setting, we allow the system both dialogue initiative and task initiative at each turn. In our form-based dialogue system, once a dialogue action is selected based on the dialogue motivator and initiative distribution, it is then realized through form-based objects.

8.3.4.1. Dialogue motivators

Currently, the dialogue manager in COMIX uses six dialogue motivators. They are: Restriction, Relaxation, Clarification, ProvideAnswer, NotifyFailure, and ErrorCorrection. These dialogue motivators are identified from 41 naturally occurring information-seeking dialogues (SRI transcripts, 1992) which consists of 3086 dialogue turns. Here, we illustrate these motivators with examples from the SRI transcripts, with IP as information provider (the travel agent) and IS as information seeker (the client). The utterances that support a specific dialogue motivator are highlighted in bold type. The applicable conditions for these dialogue motivators are provided later in Table 51 in the dialogue manager algorithm section.

Restriction

Restriction is to seek pieces of information that the information provider must know in order to process the information seeker's request. When the problem definition is under-specified, restriction generally applies.

Dialogue excerpt 1

(1) IS: oh hi D I need an airfare for a proposal

(2) IP: ok from where to where
(3) IS: SFO to Kwalalampour

Dialogue excerpt 2

(1) IS: now let's see if we get a quote here.
(2) IP: ah full coach
(3) IS: yeah
(4) IP: all right round trip it's twenty three thirty eight or twenty excuse me twenty two thirty eight

In Dialogue excerpt 1, the departure city and the arrival city are required for retrieving the airline information, while in Dialogue excerpt 2, the type of seating is required. These pieces of information must be asked first before relevant flights can be retrieved.

Relaxation

Relaxation refers to the ability of the information provider to drop certain constraints that result in information access failure in order to process the information seeker's request. Relaxation occurs in over-constrained situations.

Dialogue excerpt 3

(1) IS: San Jose to LA
(2) IP: and it was into LA on Continental
(3) IS: yeah
(4) IP: I don't show that Continental flies there from San Jose
(5) IP: um maybe she was looking out of San Francisco
(6) IP: yes, there's flight out of San Francisco at nine o'clock

Dialogue excerpt 4

(1) IS: He wants to take a Piedmont flight that leaves Phoenix aimed for Baltimore, at ten thirty in the morning.
(2) **IP: Okay I show a - a Piedmont flight f- at nine twenty-five a.m**.
(3) IP: I don't show a ten thirty one.

In both dialogue excerpts, the requests are over-constrained in that no flights are available to satisfy the requests. The requests must be relaxed for finding a flight. In Dialogue excerpt 3, the information provider chooses to offer another value for the departure city, which yields a flight, while in Dialogue excerpt 4, the travel time is relaxed for available flights.

Clarification

Clarification occurs when the information provider cannot uniquely identify the values intended for an attribute to execute the task. For example,

Dialogue excerpt 5

(1) IS: let's see what would get you there leaving the seventh.
(2) IP: from San Jose or San Francisco?
(3) IS: San Francisco. Actually Oakland would be good too on that.

In this excerpt, the information provider has two eligible departure cities for planning the flights. A clarification question is thus asked in utterance (2).

<u>ErrorCorrection</u>

ErrorCorrection occurs when the information seeker gives an invalid value to an attribute because of invalid beliefs. For example,

Dialogue excerpt 6

(1) IS: I am flying two different airlines, is that going to foul up my ticketing?
(2) **IP: no**.

In this case, the information seeker has the invalid belief that flying two different airlines may affect ticketing. The information provider corrects this invalid belief in utterance (2).

<u>ProvideAnswer</u>

ProvideAnswer occurs when the information provider has all the necessary information to execute the requests of the information seeker. For example,

Dialogue excerpt 7

(1) IP: and you want to travel again on what day now?
(2) IS: it depends on when the flights are. I would leave late Friday after work on the eighth of September and if that doesn't work out then I could go the morning of the ninth of September
(3) IP: ok were going San Francisco to Syracuse it's just going to be a simple round trip?
(4) IS: right.
(5) IP: ok and do you have a preference as to airlines?
(6) IS: um I don't.
(7) IP: ok the last flight out is going to be at one forty seven p.m. on TWA in terms of something that will get you there the same day that would get in at just about midnight

In this dialogue excerpt, the information provider is able to find available flights to satisfy the request of the information seeker. The flight is communicated to the information seeker in utterance (7).

<u>NotifyFailure</u>

When the requests of the information seeker cannot be satisfied, it is customary for the information provider to inform the information seeker of the over-constrained situations. For example:

Dialogue excerpt 8

(1) IS: San Jose to LA
(2) IP: and it was into LA on Continental
(3) IS: yeah
(4) IP: I don't show that Continental flies there from San Jose

Here, the information provider cannot retrieval a Continental flight from San Jose to LA, and informs the information seeker of the no-solution status in utterance (4).

In summary, the Restriction motivator determines whether a request is under-specified and determines the additional constraints to request. The Relaxation motivator determines whether a request is over-constrained and determines potential relaxation candidates. The Clarification motivator seeks to clarify between multiple possible values. The ErrorCorrection motivator determines the invalidity of proposed values. The ProvideAnswer motivator determines whether the system has acquired enough information to query a database for the requested information. The NotifyFailure motivator determines the no-solution status of certain information requests.

8.3.4.2. The dialogue manager

The dialogue manager works as follows: for each object c in the task knowledge representation, the dialogue manager performs all applicable dialogue motivators to the object. The dialogue motivators are attempted in the sequence of ErrorCorrection, Clarification, ProvideAnswer, NotifyFailure, Restriction, and Relaxation. This order ensures, for instance, that ambiguity (Clarification) and invalid beliefs (ErrorCorrection) are resolved first before the system attempts to obtain answers (ProvideAnswer or NotifyFailure). If a dialogue motivator DM_i applies, then an appropriate dialogue action will be initiated taking into account initiative distribution (e.g., initiate a clarification sub-dialogue when the system has dialogue initiative) and the object c gets updated with the answer c_a from the interaction. This process repeats until no dialogue motivator applies, and the dialogue manager provides the final answer of the object. The algorithm is summarized in Figure 47, which is adapted from (Abella and Gorin, 1999: p197):

Repeat
For all dialogue motivators
    If DM_i applies to c, Then
        Select an appropriate dialogue action based on initiative distribution
        Perform the selected dialogue action
        Obtain answer c_a through interaction

Update c with answer c_a

End If

Next DM_i

Until no dialogue motivator applies

Return c

Figure 47: Dialogue manager algorithm over an object *c*.

While formal definitions of the dialogue motivators are beyond the scope of this thesis, an information summary of the applicable conditions of these motivators will shed light on how the dialogue manager selects them. Table 51 provides such a summary.

Dialogue motivators	Applicable conditions
ErrorCorrection	The value of an attribute in an object is invalid (e.g., the value does not exist in the database).
Clarification	An attribute in an object (other than the solution object) has more than one value.
ProvideAnswer	The value of the solution object is set to a non-empty set after execution. The size of the non-empty set is equal to or smaller than a pre-determined size *k*.
NotifyFailure	The value of the solution object is set to empty after execution.
Restriction	The value of the solution object is set to a non-empty set after execution. The size of the non-empty set is greater than a pre-determined size *k*. There exist attributes in the object whose instantiation could result in a reduced set of solutions.
Relaxation	The value of the solution object is set to empty after execution. There exist attributes in the object whose relaxation could result in a reduced set of solutions.

Table 51: Summary of the applicable conditions for the dialogue motivators.

For instance, suppose that the user provides "Rochester" as the departure city when searching for a flight. The dialogue manager applies all the dialogue motivators to the departure city object DeptCity in the order of ErrorCorrection, Clarification, ProvideAnswer, NotifyFailure, Restriction and Relaxation. When the ErrorCorrection motivator is attempted, the DeptCity object sends the value "Rochester" to the airline database, checking for its validity. The city "Rochester" exists in the database, so this motivator does not apply. When the Clarification motivator is attempted, the DeptCity object sends the value "Rochester" again to the airline database for ambiguity checking. The search returns two possible values for the string "Rochester" – "Rochester, NY" or "Rochester, MN". This invokes the Clarification motivator and the dialogue manager adopts appropriate dialogue actions to initiate clarification sub-dialogues (refer to section 8.4 for an example clarification sub-dialogue in a form-based dialogue system). Suppose that through the sub-

dialogue, the user selects "Rochester, MN" as the right value. The dialogue manager then instantiates the DeptCity object with this unambiguous value. The remaining four dialogue motivators (ProvideAnswer, NotifyFailure, Restriction and Relaxation) continue to be tried on the now instantiated DeptCity object. Since the DeptCity object is not a solution object, none of the four motivators applies.

8.3.4.3. Initiative-taking dialogue actions

Depending on the initiative distribution, the system can adopt different dialogue actions to satisfy the applicable dialogue motivators. In our work, these dialogue actions are motivated by cooperative dialogue actions employed by human information seekers and information providers in information access dialogues.

Corpus-based initiative-taking dialogue acts

We have analyzed naturally occurring information-seeking dialogues (SRI Transcript, 1992) to identify initiative-taking dialogue actions. We identify both dialogue initiative-taking actions and task initiative-taking actions. We identify two dialogue initiative-taking actions:

Clarify: to initiate a sub-dialogue to ask the information seeker to choose among possible values, e.g., *from San Jose or San Francisco?*

InformError: to initiate a sub-dialogue to notify the information seeker the invalidity of value assignments, e.g., *No* as in utterance (2) dialogue except 6.

Our focus, however, is on task initiative-taking actions for resolving under-constrained and over-constrained situations and their potential effects on problem formulation and solution refinement. Here we present five such task initiative-taking dialogue actions employed by the information provider (IP), though some of the same actions can be employed by the information seeker (IS). We now illustrate three task initiative-taking dialogue actions (DA1-3), presented in an increasing degree of agent initiative, for resolving under-constrained situations:

DA1. Request values for attributes (RequestVal)

Dialogue excerpt 9

(1) IS: oh hi D I need an airfare for a proposal
(2) IP: ok from where to where

(3) IS: SFO to Kwalalampour

The IP suggests to the IS that the attributes departure city and arrival city are important to the problem definition, and requests their values to be added to the problem definition in utterance (2).

DA2. Propose values for attributes (ProposeVal)

Dialogue excerpt 10

(1) IS: now let's see if we get a quote here.
(2) IP: ah full coach
(3) IS: yeah
(4) IP: all right round trip it's twenty three thirty eight or twenty excuse me twenty two thirty eight

The IP explicitly proposes to the IS that the new attribute "class" is important to the problem definition and offers a value for this attribute in utterance (2).

DA3. Propose values for attributes and inform solutions (ProposeVal&Solutions)

Utterance (4) in Dialogue excerpt 10 is an example of such a dialogue act. The IP further refines the problem definition by adding constraint "travel type" being round trip while presenting a solution.

Next we illustrate two task initiative-taking dialogue actions (DA4-5) for resolving over-constrained situations:

DA4: Propose changed values for attributes (ProposeNewVal)

Dialogue excerpt 11

(1). IS: San Jose to LA
(2). IP: and it was into LA on Continental
(3). IS: yeah
(4). IP: I don't show that Continental flies there from San Jose
(5). IP: um maybe she was looking out of San Francisco
(6). IP: yes, there's flight out of San Francisco at nine o'clock

Since no flight departing from San Jose is found to satisfy the problem definition (utterance (4)), the IP takes the task initiative to change the value of the departure city to San Francisco (utterance (5)), thus changing the original problem definition. A solution is successfully found with the new problem definition at the next dialogue turn (utterance (6)).

DA5: Propose changed values for attributes and inform solutions (ProposeNewVal&InformAnswer)

Dialogue excerpt 12

(1). IS: He wants to take a Piedmont flight that leaves Phoenix aimed for Baltimore, at ten thirty in the morning.
(2). IP: Okay I show a - a Piedmont flight f- at nine twenty-five a.m.
(3). I don't show a ten thirty one.

Since no flight is found for the Piedmont flight that leaves Phoenix for Baltimore at ten thirty in the morning, the IP relaxes the travel time constraint. A solution at another time is successfully found (utterance (2)).

Constraint-based support of task initiative-taking dialogue actions

Now, we describe how constraint-based knowledge sources support the five task initiative-taking dialogue actions illustrated above. Again, we analyze the actions from the perspective of the system.

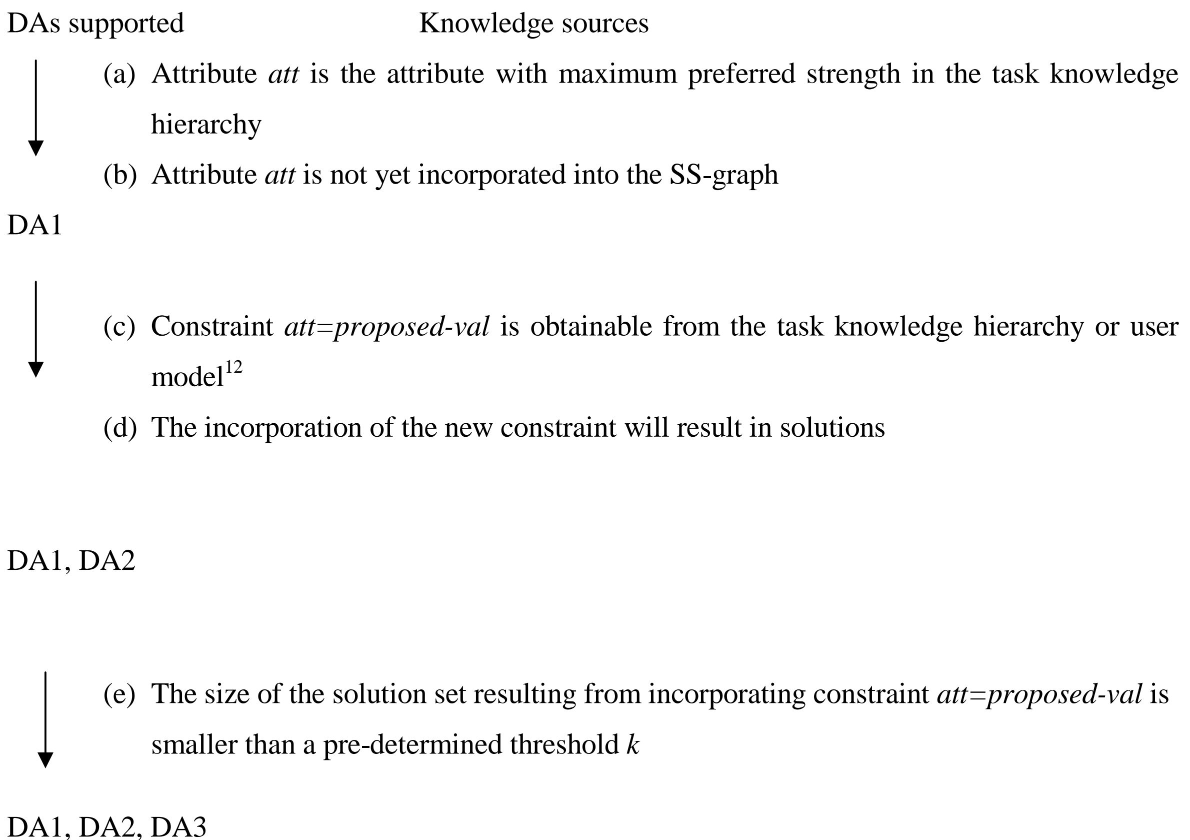


Figure 48: How knowledge sources support generation of DA1-3 in under-constrained situations.

[12] User model is the specification of stereotypical user preferences including a set of attributes, possible values for these attributes (constraints), and the importance (i.e., constraint strength) of the attributes and constraints for solving the task at hand.

Dialogue actions DA1-3 specify the dialogue actions that an agent can adopt in under-constrained situations. Figure 48 illustrates how exploiting additional knowledge sources in the solution space can help increase the choices in selecting dialogue actions for resolving such situations. In under-constrained situations, new constraints need to be introduced to refine the problem definition. In information dialogues, new constraints are new attribute value pairs that are yet to be introduced to the problem definition. When an attribute *att* is not yet incorporated into the SS-graph (condition (b) in Figure 48), but is identified as the most important attribute for solving the task according to task-specific knowledge sources (condition (a)), initiative-taking action DA1 is supported and can be selected for generation. If, in addition, stereotypical constraining values for the attribute *att* are defined in the task-specific knowledge sources or the user model (condition (c)), the system can then propose a default value along with the identified attribute. However, a cooperative system needs to make sure that the proposed attribute-value pair constraint will result in solutions (condition (d)). When conditions (c) and (d) are satisfied besides conditions (a) and (b), initiative-taking action DA2 is supported and can be selected for response generation. If, in addition, the size of the solution set resulting from incorporating the constraint *att=proposed-val* is smaller than a pre-determined threshold *k* (condition (e)), then initiative-taking action DA3 is supported and can be selected for generation. By DA3, solutions are presented together with the proposed constraint before the proposed constraint is confirmed by the user. The goals of DA1-3 are to update the CSP by adding the variable *att* and its constraints (possibly the constraint *att=proposed-val* as in DA1 or DA3 if the user accepts the proposed constraint), and to update the SS-graph to generate new solutions by incorporating the new constraints.

Dialogue actions DA4-5 specify the dialogue actions that an agent can adopt in over-constrained situations. Figure 49 illustrates how exploiting additional knowledge sources in the solution space can help increase the choices in selecting dialogue actions for resolving such situations. To resolve an over-constrained situation, constraints need to be relaxed or modified. When an attribute *att* has already been incorporated into the SS-graph (condition (a) in Figure 49), and the constraint *att=old-val* is identified by the solution modification module as the relaxation candidate (condition (b)), and a new constraint *att=new-val* is identified by the solution modification module as a proposal which will result in solutions (condition (c)), then initiative-taking action DA4 is supported and can be selected for generation. If, in addition, the size of the solution set resulting from incorporating *att=new-val* is smaller than a pre-determined threshold *k* (condition (d)), then initiative-taking action DA5 is supported and can be selected for generation. By taking DA5, solutions are presented together with the proposed constraint modification before the proposed

modification is confirmed by the user. The goal of DA4-5 is to update the problem definition by removing the old constraint *att=old-val* and by adding in a new constraint *att=new-val* in order to obtain a non-empty set of solutions. By incorporating the modified constraint into the SS-graph, new solutions can be computed.

DAs Supported | Knowledge sources

↓
(a) Attribute *att* is already incorporated into the SS-graph
(b) Constraint *att=old-val* is identified by the solution modification module as the relaxation candidate
(c) Constraint *att=new-val* is identified by the solution modification module as a potential proposal which will result in solutions

DA4

↓
(d) The size of the solution set resulting from incorporating constraint *att=new-val* is smaller than a pre-determined threshold *k*

DA5, DA4

Figure 49: How knowledge sources support generation of DA4-5 in over-constrained situations.

A possible future extension is to generate reasons to support the above dialogue acts, especially those associated with relaxation, 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.

Adoption of different dialogue actions can affect the dialogue efficiency and effectiveness of human-computer dialogue. For example, we hypothesize that RequestVal may be less efficient than ProposeVal or ProposeVal&Answer even when all these dialogue actions are supported by the knowledge sources as possible strategies for dealing with under-constrained problems, because new user constraints obtained from a user's response to RequestVal may lead to an over-constrained situation, while a user's acceptance of ProposeVal or ProposeVal&Answer will guarantee valid solutions. Empirical evaluation of the efficiency and effectiveness of adopting different dialogue actions is required but we leave this as our future work.

Effect of task initiative-taking dialogue actions on CSP

Initiative actions	Effects on CSP
DA1	The attribute is added to problem definition; The value from the information seeker will be added to the problem definition; Partial solutions using this attribute value pair are constructed.
DA2	The attribute is added to the problem definition; The confirmed value or the value from the information seeker will be added to the problem definition; Partial solutions using this attribute value pair are constructed.
DA3	The attribute is added to the problem definition; The confirmed value or the value from the information seeker will be added to the problem definition; Partial solutions using this attribute value pair are constructed; Solutions are communicated to the information seeker
DA4	The confirmed value or the value from the information seeker will be added to the problem definition; Partial solutions using this attribute value pair are constructed.
DA5	The confirmed value or the value from the information seeker will be added to the problem definition; Partial solutions using this attribute value pair are constructed; Solutions are communicated to the information seeker.

Table 52 : Task initiative-taking dialogue actions and their effects on CSP.

Table 52 summarizes the effect of task initiative-taking dialogue actions on the constraint-based problem solver. Note that in Table 52, regardless of whether the proposed value is accepted or not by the information seeker, the attribute itself should be added to the problem definition, as the information provider should not have asked this attribute if it is not of certain importance. The final value for this attribute in the problem definition can be the proposed value when the information seeker accepts, or some alternative values offered by the information seeker.

Selection of dialogue actions based on dialogue motivator and initiative distribution

Table 53 presents the alternative dialogue actions applicable to the dialogue motivators based on initiative distribution. Once a dialogue motivator is applied, the distribution of initiative affects the selection of initiative-taking actions. For instance, the dialogue motivator Relaxation is invoked when the problem is over-constrained. When the system finds a relaxation candidate that can maximally support the information request, the system can adopt the ProposeNewVal dialogue action, suggesting a new alternative to the user's original constraints. When the resulting solution set is small enough, the system can adopt the ProposeNewVal&Answer, providing the possible solutions when the new alternative is selected. In these cases, the system takes over both the task and dialogue initiative. If, in the next turn, the system has only dialogue initiative, but not the task

initiative, or the system does not have either initiative, then neither ProposeNewVal nor ProposeNewVal&Answer will be considered as candidate dialogue actions for response generation. In this case, no action will be taken by the system. As we have mentioned earlier, we adopt a simple fixed policy assignment of initiative distribution. More sophisticated initiative tracking can be achieved using the approach proposed by Chu-Carroll and Brown (1997b).

Motivators	Task + Dialogue Initiative	Dialogue Initiative	No Initiative
Restriction	RequestVal ProposeVal ProposeVal&Answer	No action[13]	No action
Relaxation	ProposeNewVal ProposeNewVal&Answer	No action	No action
Clarification	Clarify	Clarify	No action
ErrorCorrection	InformError	InformError	No action
ProvideAnswer	Answer[14]	Answer	Answer
NotifyFailure	InformFailure	InformFailure	InformFailure

Table 53: Selection of dialogue actions based on dialogue motivator and initiative distribution, with the different distributions of initiative listed in the first row and the different dialogue motivators listed in the first column.

8.3.5. Form-based response generation

The response generation component takes as input dialogue acts selected by the dialogue manager and the parameters needed to instantiate the dialogue acts and generates appropriate form-based objects to realize these dialogue acts in a form-based interaction. The form-based objects used for realizing the dialogue acts are shown in Table 54. There is generally more than one way to implement a dialogue action. For example, the ProposeValue action can be presented by text boxes, combo boxes, or lists filled with the suggested values.

The control of the main task dialogue and the clarification or error correction sub-dialogues is managed by main forms and pop-up sub-forms. In the system, we maintain two main forms – a query form and a solution form – for the main dialogue. Dialogue actions that affect task initiatives

[13] In human dialogues, when the agents are not taking over either the dialogue initiative or the task initiative, the agents often use acknowledgements or filler utterances to notify the listener of the engagement in the dialogue.
[14] The Answer action is to deliver the answer (e.g., utterance (7) in dialogue excerpt 7) and the InformFailure action is to notify the no-solution status (e.g., utterance (4) in dialogue excerpt 8). Both actions are independent of initiative distribution.

are implemented as form-based objects in these two main forms. Clarification sub-dialogues and error correction sub-dialogues, which affect the dialogue initiative but not task initiative, are implemented using pop-up forms. These pop-up forms have two characteristics:

First, once a pop-up form is activated, the focus will be set to the pop-up form and the main form becomes inaccessible. The user is required to respond to the pop-up form first before getting back to the main form. This is based on the assumption that the dialogue participants need to square away ambiguity or invalid belief first because proceeding with the execution of the task, corresponding to the stack-based dialogue management model by Grosz and Sidner (1986).

Dialogue Actions	Form-Based Objects
RequestVal	• Text boxes • Combo boxes • Option groups • Lists
ProposeVal	• Text boxes with suggested values • Combo boxes with suggested values • Option groups with suggested values • Lists with suggested values
ProposeNewVal	• Text boxes with suggested values • Combo boxes with suggested values • Option groups with suggested values • Lists with suggested values
Answer	• Text labels
ProposeVal&Answer	• Text boxes with suggested values plus text labels for answer • Combo boxes with suggested values plus text labels for answer • Option groups with suggested values plus text labels for answer • Lists with suggested values plus text labels for answer
ProposeNewVal&Answer	• Text boxes with suggested values plus text labels for answer • Combo boxes with suggested values plus text labels for answer • Option groups with suggested values plus text labels for answer • Lists with suggested values plus text labels for answer
Clarify	• Text boxes • Combo boxes • Option groups • Lists
InformError	• A message box with error messages
InformFailure	• Text labels informing no solutions

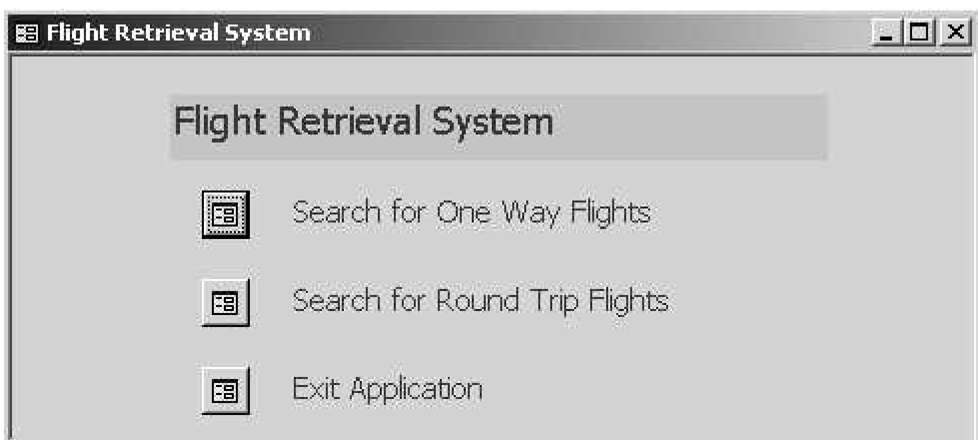
Table 54: Correspondence between dialogue actions and form-based objects

Second, after the user performs appropriate actions on the pop-up forms, the pop-up forms will disappear and the focus will be returned to the main form, where the pop-up form is originated. This corresponds to situations where the ambiguity or invalid belief is resolved through sub-dialogues and the dialogue participants return to the task-executing main dialogue.

8.4. An Illustrative Example

In this section, we step through a dialogue between the user and the system to illustrate several aspects of our dialogue management strategies. System turns are annotated with the relevant dialogue actions.

(1) **User**: the user clicks on the "Search for One Way Flights" button to start the dialogue.



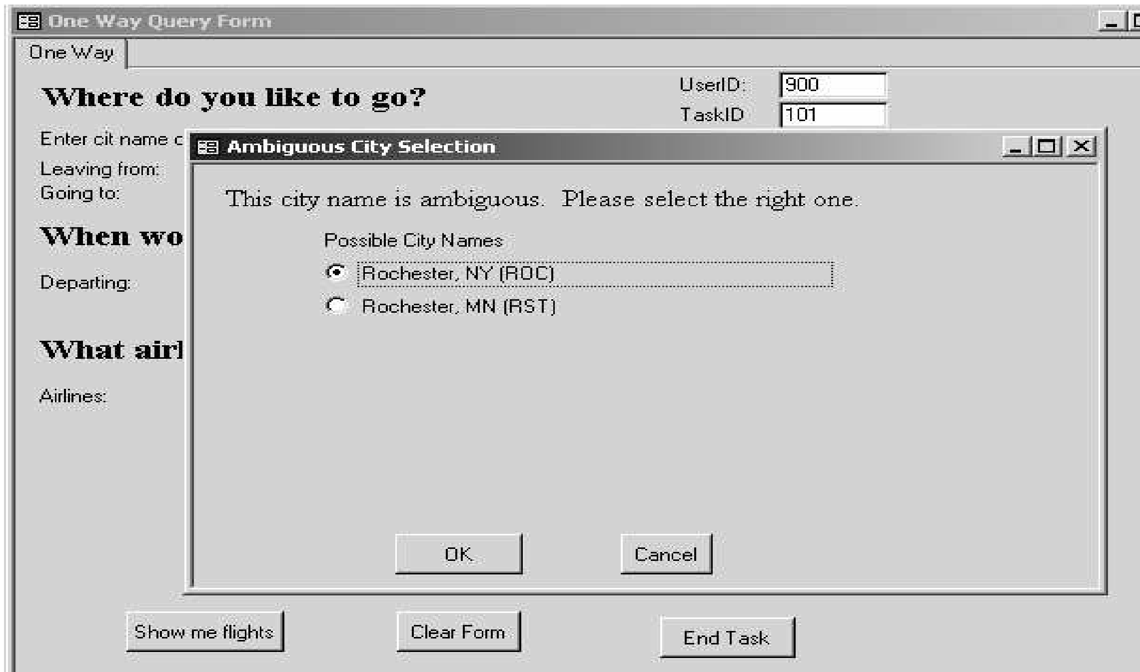
(2) **System**: the system displays the main query form (shown below) to solicit necessary information for solving the task. The system also supplies some default values (e.g., default strengths for departure city, default departure date as today) that the user can update. [RequestVal, ProposeVal]

One Way Query Form
One Way
Where do you like to go?
UserID:
TaskID
Enter cit name or airport code (e.g., "Los Angeles" or "LAX")
Leaving from: Required Airport Code Search
Going to: Required Airport Code Search
When would you like to travel?
Departing: September 8 Calendar Strong Prefer
Departure Time After 5:00:00 AM Strong Prefer
What airlines do you prefer? (OPTIONAL) Weak Prefer
Airlines: Search all airlines
Search these airlines
1st choice:
2nd choice:
3rd choice:
Show me flights Clear Form End Task

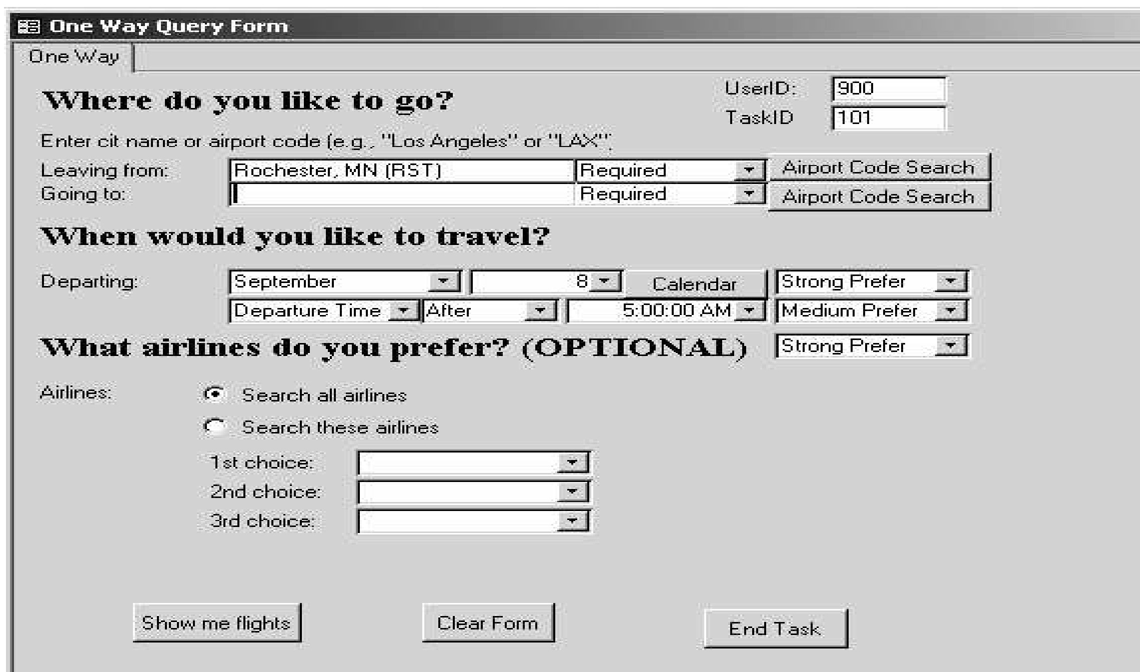
(3) **User**: the user fills in the departure city as “Rochester”.

One Way Query Form
One Way
Where do you like to go?
UserID: 900
TaskID 101
Enter cit name or airport code (e.g., "Los Angeles" or "LAX")
Leaving from: rochester Required Airport Code Search
Going to: Required Airport Code Search
When would you like to travel?
Departing: September 8 Calendar Strong Prefer
Departure Time After 5:00:00 AM Medium Prefer
What airlines do you prefer? (OPTIONAL) Strong Prefer
Airlines: Search all airlines
Search these airlines
1st choice:
2nd choice:
3rd choice:
Show me flights Clear Form End Task

(4) **System**: the system initiates a clarification sub-dialogue for the user to choose the correct value for the departure city. [Clarify]



(5) **User**: the user selects "Rochester, MN (RST)" and clicks the "OK" command. The sub-dialogue form disappears and the user is back to the main query form. The unambiguous value for "Rochester" appears in the departure city text box as "Rochester, MN (RST)".



(6) **User**: the user fills in more constraints, with the arrival city as San Diego, CA, the departure date as September 15, 2001, the departure time as after 5pm, and the airline as Delta. The

preference strengths for the departure city and the arrival city are required, the strength for the departure date strong prefer, the departure time medium prefer, and the airline strong prefer. After filling in the constraints, the user clicks on the "Show me flights" button to request the system to find the flights that satisfy the constraints.

One Way Query Form
One Way
Where do you like to go?
UserID: 900
TaskID 101
Enter cit name or airport code (e.g., "Los Angeles" or "LAX")
Leaving from: Rochester, MN (RST) Required Airport Code Search
Going to: San Diego, CA (SAN) Required Airport Code Search
When would you like to travel?
Departing: September 15 Calendar Strong Prefer
Departure Time After 5:00:00 PM Medium Prefer
What airlines do you prefer? (OPTIONAL) Strong Prefer
Airlines: Search all airlines
Search these airlines
1st choice: Delta
2nd choice:
3rd choice:
Show me flights Clear Form End Task

(7) **System**: the system cannot find any flights that satisfy the user's constraints. The system starts relaxation procedure and identifies the relaxation candidates as the departure time constraint and proposes relaxation values as time between 5:50am and 11:35am when flights are available. [InformFailure, ProposeNewVal]

Flight Result Form
Flight Retrieval Result
No flights are found. The system suggests adjusting the following item(s) simultaneously for relaxation.
Suggest selecting departure time between 5:50:00 AM and 11:35:00 AM
Departure Time Before 11:35:00 AM
Search Flights Go Back

(8) **User**: the user accepts the proposed departure time and clicks on the button "Search Flights" to request the system to find flights again.

(9) **System**: the system is able to find flights this time and return the flights to the user. [Answer]

Flight Result Form

Select a Flight

All times shown are local to each city. A total of 2 flight(s) are found.

Select	Delta (DL) 2898 on a SF3 From: Rochester, MN (RST) 15-September-2001 11:35:00 AM To: Minneapolis/St. Paul, MN (MSP) 12:18:00 PM	Price: $804	Detail
	Delta (DL) 195 on a 320 From: Minneapolis/St. Paul, MN (MSP) 15-September-2001 5:05:00 PM To: San Diego, CA (SAN) 6:57:00 PM		Detail
Select	Delta (DL) 3500 on a ARJ From: Rochester, MN (RST) 15-September-2001 5:50:00 AM To: Minneapolis/St. Paul, MN (MSP) 6:31:00 AM	Price: $804	Detail
	Delta (DL) 195 on a 320 From: Minneapolis/St. Paul, MN (MSP) 15-September-2001 5:05:00 PM To: San Diego, CA (SAN) 6:57:00 PM		Detail

(10) **User**: the user clicks on the first "Select" button and selects the first flight.

(11) **System**: the system displays the details of the itinerary. [Answer]

Confirm Your Flight

This is the flight you selected. Please write it down and confirm your selection.

Total Price: $804 Confirm your selection Cancel

Total Number of Stops: 1

Delta (DL) 2898	SF3		
Departing from Rochester, MN (RST)		15-September-2001	11:35:00 AM
Arriving at Minneapolis/St. Paul, MN (MSP)		15-September-2001	12:18:00 PM
Number of stops: None	Meals served: None		
Delta (DL) 195	320		
Departing from Minneapolis/St. Paul, MN (MS		15-September-2001	5:05:00 PM
Arriving at San Diego, CA (SAN)		15-September-2001	6:57:00 PM
Number of stops: None	Meals served: Dinner		

(12) **User**: the user confirms the flight by clicking the "Confirm your selection" button and exits the dialogue.

At the beginning of the dialogue, the user's information need is yet to be specified. The system adopts the dialogue actions RequestVal and ProposeVal for resolving the under-specification of the information need (turn (2)). After the user fills in "Rochester" as the departure city, the system detects ambiguity through checking with the domain database and initiates a clarification sub-dialogue by adopting the Clarify action in turn (4) (see section 8.3.4on how the dialogue manager applies the Clarification motivator to detect ambiguity). After the ambiguity is resolved, the user inputs more constraints and requests the system for solutions. Since the information need is over-constrained, the system adopts task initiative in identifying relaxation candidates for the user. The departure time is identified as the constraint to relax and its valid values are computed. The system informs the user of the over-constrained status (action InformFailure) and provides relaxation suggestions at the same time (action ProposeNewVal) in turn (7). When the user accepts the suggested relaxation, the system searches the database again, and finds flights that satisfy the modified information need. Answers are then returned to the user in turn (10).

Chapter 9 Usability Study: Experimental Design and Result Analysis

In the previous chapter, we have described the COMIX collaborative information access system with the capability of task initiative-taking actions. To support the generation of these task initiative-taking actions, the system relies on the constraint-based problem solver for selecting appropriate parameter instantiations. In Chapter 7, we have evaluated the effectiveness and efficiency of applying different question selection and relaxation selection methods by the constraint-based problem solver with simulated user information needs. We have observed that, in general, the use of knowledge, in particular a user's preference constraint hierarchy, demonstrates better dialogue efficiency and higher task success rate compared with those of other methods. Since these experiments are based on simulation, we have validated the algorithms but have not yet addressed the issue of usability. In this chapter, we evaluate whether task initiative-taking actions supported by the constraint hierarchy also directly improve usability. The goals of usability testing are focused on various factors including the usefulness, the effectiveness, and the learnability of a particular system. Usefulness of a system is concerned with the degree to which users are able to achieve their goals. For example, are users able to locate the information they are looking for from the system, such as the flight information of a trip? The effectiveness factor or ease of use, is usually measured in terms of speed, performance, or error rates in relation to completing tasks. Can users find the flight information quickly and accurately from the system? Learnability factor focuses on the ability of users to operate the system after a training period. In the following, we describe the controlled experiments we have conducted for exploring the various factors of usability. We describe the experimental design and data collection in section 9.1, and the analysis of the experimental results in section 9.2.

9.1. Experimental Design and Data Collection

9.1.1. Hypothesis

The evaluation design question is whether the mixed-initiative system is preferable to the user-initiative system on the grounds of usability. Specifically, we evaluate the validity of the following hypotheses:

- The mixed-initiative system produces more efficient dialogues compared with the user-initiative system;

- The mixed-initiative system produces higher task success rates compared with the user-initiative system;
- The mixed-initiative system demonstrates better usability than the user-initiative system.

The null hypothesis is that there is no statistically significant difference in terms of usability between the mixed-initiative system and the user-initiative system.

9.1.2. Task scenarios

The users were given flight reservation tasks that required them to access an airline schedule database through COMIX. The airline schedule database has been described earlier in section 8.2. For each task, COMIX and the user exchanged information about the user's information need and the available solutions. Appendix 3 provides examples of typical one-way trip tasks and round-trip tasks.

9.1.3. System settings

We compare dialogues collected via two different task initiative settings: one with system task initiative (COMIX-MI) as described in the previous chapter, the other one without task initiative (COMIX-UI), which takes away the system's ability for generating task initiative. COMIX-UI resembles most of the commercial sites for travel reservations, such as Travelocity or Yahoo!Travel. The system asks the user the necessary travel information, such as departure city, arrival city, departure date, departure time, and carrier (Figure 50). Then the system looks up the airline database and retrieves flights to satisfy the user's request. When over-constrained situations occur, the system generally suggests to the user to relax the original information need (Figure 51). It is up to the user to try different relaxation options to find out what is to relax for flights to be available.

Figure 50: Flight query form from COMIX-UI.

Figure 51: System response form for over-constrained queries by COMIX-UI.

The user interface of COMIX-MI is similar to that of COMIX-UI, but requests the user to make explicit the preference strengths for the attributes through the interface (Figure 52). When over-constrained situations occur, the system takes initiative to identify the cause of the over-constrained situations, and returns that information to the user (Figure 53). The information the system uses for

such identification is the preference hierarchy and system-internal information of partial solutions. The user then is able to update that specific information rather than guessing which attribute needs to be relaxed.

Figure 52: Flight query form from COMIX-MI.

Figure 53: System response form for over-constrained queries by COMIX-MI.

When returning results for under-constrained situations, the two systems do not have much difference. Both system settings simply return the flights to the user, regardless of the initiative mode (Figure 54). The current design of both system settings requires the users to specify their travel constraints through one single form (corresponding roughly to one dialogue turn), instead of

incrementally over several turns as is common in speech-based interfaces or similar to our simulated experiments in Chapter 7. One could design the interface to simulate the incremental process of information acquisition. As a result, solutions could be computed incrementally, resulting in different behaviors in presenting solutions. We leave this exercise as our future work.

Flight Result Form

Select a Flight

All times shown are local to each city. A total of 167 flight(s) are found.

Select | USAir (US) 3621 on a ARJ | Price: $100.5 | Detail
From: Pittsburgh, PA (PIT) 25-July-2001 1:00:00 PM
To: Detroit, MI (DTW) 2:05:00 PM

Select | Northwest (NW) 1481 on a D9S | Price: $100.5 | Detail
From: Pittsburgh, PA (PIT) 25-July-2001 7:00:00 AM
To: Detroit, MI (DTW) 8:05:00 AM

Select | Delta (DL) 3549 on a ARJ | Price: $100.5 | Detail
From: Pittsburgh, PA (PIT) 25-July-2001 11:30:00 AM
To: Detroit, MI (DTW) 12:39:00 PM

Select | Northwest (NW) 1483 on a DC9 | Price: $100.5 | Detail
From: Pittsburgh, PA (PIT) 25-July-2001 3:05:00 PM
To: Detroit, MI (DTW) 4:14:00 PM

Select | Delta (DL) 495 on a D9S | Price: $100.5 | Detail
From: Pittsburgh, PA (PIT) 25-July-2001 10:00:00 AM
To: Detroit, MI (DTW) 11:10:00 AM

Select | USAir (US) 3612 on a ARJ | Price: $100.5 | Detail
From: Pittsburgh, PA (PIT) 25-July-2001 4:45:00 PM
To: Detroit, MI (DTW) 5:56:00 PM

Select | USAir (US) 3591 on a ARJ | Price: $100.5 | Detail

Figure 54: Result form when flights are found for both COMIX-UI and COMIX-MI.

9.1.4. Users

We recruited a total of 16 users for the experiments and obtained an estimate of each user's experience with computers and with online airline reservation systems through a pre-experiment questionnaire (Appendix 1). All users had more than 3 years of experience using computers, with computer usage time ranging from 20 hours per week to more than 40 hours per week. Except for two users who had never made airline reservations, all other users had made airline reservations either with an agent, with the airline, or through the Internet.

9.1.5. Pilot runs

Before bringing in users for the evaluation experiments, we had a rehearsal of our test questions with two pilot users. Both of them are experienced computer users.

Many of our questions proved to be quite easy for the users to solve with both system settings, so we introduced some tasks of increased task difficulty. To keep the experimental running time between 1 hour and 2 hours to avoid user fatigue, we reduced the training questions to 2 tasks (one one-way trip task and one round-trip task) per user per system setting and reduced the evaluation tasks to 4 tasks per user per system setting.

9.1.6. Performance measures

Data were collected in two ways: dialogue logs and user survey data. The dialogue logs, which provide objective measures over the performance of the system, were logged automatically by the system while the user was working with a task. The user surveys, which measure subjective evaluation of the system by the user, were collected through a post-task questionnaire.

For the dialogue logs, the system logs the total completion time of a dialogue (**Total Completion Time**), the user actions, the system actions, and the effect of these actions. In addition, turns and task initiative-taking actions are recorded. From these logs, we can compute the number of dialogue turns that the user takes (**User Turns**) and the number of dialogue turns that the system takes (**System Turns**). Based on the user-recorded solution for each task, we calculated Kappa score to measure the degree of task success (**Kappa**).

The post-task questionnaire asks the user's judgment of task success and user satisfaction. Task success is a yes-no question (mapped to numerical value 1 or 0) to obtain user's perception of task completion (**Task Success**). User satisfaction was calculated from the user's response to survey questions for each task. The survey, adapted from the questionnaire used by Walker et al. (1997), includes multiple-choice questions asking about **Task Ease**, **Solution Confidence, User Expertise, Expected Behavior**, **Behavior Completeness**, **System Speed**, and **Future Use**. The possible responses to these multiple-choice questions ranged over almost always, often, sometimes, rarely, almost never, or an equivalent range. These responses are mapped into integers between 1 and 5. User satisfaction score of a particular dialogue is the cumulative satisfaction score, which is the sum of the scores of the multiple-choice questions in the survey.

The values range between 7 and 35. The user satisfaction measure, thus, gives us numerical data on the subjective experience the user had working with a particular system setting. The user survey used in the usability study can be found in Appendix 4.

The measures collected in the COMIX usability experiments are summarized in Table 55 and are grouped into dimensions of dialogue efficiency, task success, and user satisfaction.

Dialogue efficiency	Total Completion time, System Turns, User Turns
Task success	Kappa, Task Success
User satisfaction	Task Ease, Solution Confidence, User Expertise, Expected Behavior, Behavior Completeness, System Speed, Future Use

Table 55: Measures collected in the usability study.

9.1.7. Data collection procedure

The user experiments were carried out in three sessions, a training session and two evaluation sessions of the two system settings, one evaluation session with COMIX-MI, and another evaluation session with COMIX-UI. The order of working with COMIX-MI or COMIX-UI was varied for users to neutralize the sequencing/learning effect. We made it clear that the answers were always available through constraint relaxation.

The first step was a training session before we gave the users evaluation tasks to solve. During this session, the user read an instruction sheet about the system features and task requirements. They were asked to solve the two training questions and got familiar with the post-task questionnaire. The user profile was collected at this session. Appendices 2 and 3 provide the instruction sheet and the training tasks. Users could ask questions about the system and the questionnaire during the training session. This session lasted about 15 minutes to 30 minutes for a particular system setting.

During the evaluation sessions, the users were asked to solve the problems independently. After a particular task was solved, the user was asked to write down the solution to the task and fill in the post-task questionnaire for that task.

We recruited two groups of users. The first 8 users were asked to solve 4 one-way trip tasks (two of which were under-constrained and two were over-constrained) and to solve 4 round-trip tasks (again two of which were under-constrained and two were over-constrained). The eight tasks were evenly distributed for each system setting. Based on the pilot runs and the runs of the 8 users, we observed that the two system settings demonstrated differences more visibly with increased task

complexity. Therefore, we recruited an additional 8 users to solve the 4 round-trip tasks, again with the tasks evenly distributed for each system setting. This resulted in a total of 32 dialogues for one-way trip tasks and 64 dialogues for round-trip tasks.

9.2. Result Analysis

Based on the features we collected, we compare COMIX-MI and COMIX-UI along two dimensions: the task complexity dimension, i.e., whether the task is one-way or round-trip, and the task constraining dimension, i.e., whether the task is under-constrained or over-constrained.

9.2.1. The effect of task complexity

Performance measures	COMIX-UI	COMIX-MI	% change	*p*
System Turns	7.25	6.188	-14.7	0.291
User Turns	11.313	7.913	-30.9	0.178
Total Comp. Time (s)	235.5	209.563	-11.0	0.232
Task Success	0.938	1	+6.6	0.167
Kappa	0.842	0.852	+1.11	0.356
User Satisfaction	34.375	33.313	-3.1	0.107

Table 56: Comparison of the performance statistics between COMIX-UI and COMIX-MI for one-way trip tasks.

Table 56 summarizes the performance measures collected from the two system settings for the one-way trip tasks. Sixteen dialogues were recorded for each setting. We observe that the COMIX-MI improves the efficiency measures by reducing the average total completion time by 11.0%, the number of system turns by 14.7%, and the number of user turns by 30.9%. Detailed analysis shows that these decreases are primarily due to the over-constrained problems where the COMIX-MI uses less time. However, overall such decreases are not statistically significant, because we observe a significant overlap in the distributions of the sample data.

The users were able to finish all but one task, so task success scores are very similar. The increase in this measure from 0.936 by the COMIX-UI setting to 1 by the COMIX-MI setting is not statistically significant. The average kappa scores increase from 0.842 by the COMIX-UI setting to 0.852 by the COMIX-MI setting. The improvement, again, is not statistically significant.

User satisfaction decreases from 34.375 to 33.313, primarily due to one user who was confused about how to use the system relaxation suggestions for over-constrained tasks provided by the COMIX-MI setting. The same user also believed that the system response time was slower in the

COMIX-MI setting than the COMIX-UI setting. The decrease in user satisfaction, however, is not statistically significant.

Performance measures	COMIX-UI	COMIX-MI	% change	*p*
System turns	11.375	7.375	-35.2	*0.032*
User Turns	17.375	8.75	-49.6	*0.020*
Total Comp. Time (s)	381.313	328.094	-14.0	*0.015*
Task Success	0.969	1	+3.2	0.163
Kappa	0.843	0.889	+5.5	*0.042*
User Satisfaction	33.469	33.719	+0.8	0.351

Table 57: Comparison of the performance statistics between COMIX-UI and COMIX-MI for the round-trip tasks, with significant *p* values in italics.

Table 57 summarizes the performance measures collected from the two system settings for the round-trip tasks. Thirty-two dialogues were recorded for each setting. We observed that COMIX-MI improves the dialogue efficiency measures by reducing the average total completion time by about 14%, the number of system turns by 35.2%, and the number of user turns by 49.6%. The improvements with these efficiency measures are statistically significant (*p=0.032* for **System Turns**, *p=0.020* for **User Turns**, and *p=0.015* for **Total Completion Time**).

Only one task was not completed by one user in the COMIX-UI setting, so the task success scores are again very similar. The increase from 0.969 by the COMIX-UI setting to 1 by the COMIX-MI setting is not statistically significant. Kappa scores increase from 0.843 by the COMIX-UI setting to 0.889 by the COMIX-MI setting. This increase is statistically significant (*p=0.042*).

User satisfaction registers a small increase from 33.469 to 33.719. This increase, however, is not statistically significant.

To sum up, our experiments show that the system's task initiative-taking actions result in better performance in terms of dialogue efficiency and the Kappa scores for the harder round-trip tasks. For the easier one-way trip tasks, no significant differences are observed across the performance measures. User's perception of task success (i.e., **Task Success**) and usability (i.e., **User Satisfaction**) is comparable for both system settings regardless of the system's task initiative-taking capability.

9.2.2. Under-constrained tasks vs. over-constrained tasks

With round-trip tasks, we have observed significant improvement in dialogue efficiency and in the Kappa scores with the COMIX-MI setting compared with the COMIX-UI setting. We further analyze the round-trip tasks in terms of under-constrained and over-constrained situations to gain insight over where the improvements occur.

With under-constrained tasks (sixteen dialogues for each setting), we have observed that the differences between the two system settings are not significant for any of the performance measures. This is not surprising, because in under-constrained situations, when flights that satisfy the user's information needs are available, the COMIX-MI setting does not have opportunities to offer much help.

Performance measures	**COMIX-UI**	**COMIX-MI**	**% change**	**p**
System turns	16.5	8.5	-48.5	*0.031*
User Turns	27.625	10.063	-63.6	*0.016*
Total Comp. Time (s)	445.688	362.875	-18.6	*0.017*
Task Success	0.938	1	+6.7	0.167
Kappa	0.693	0.787	+13.5	*0.035*
User Satisfaction	32.813	33.188	+1.2	0.382

Table 58: Comparison of the performance statistics between COMIX-UI and COMIX-MI for the over-constrained round-trip results, with significant *p* values in italics.

Table 58 summarizes the performance measures collected from the two system settings for the over-constrained round-trip tasks. Sixteen dialogues were recorded for each setting. We observed that the COMIX-MI setting improves the dialogue efficiency measures by reducing the average total completion time by 18.6%, the number of system turns by 48.5%, and the number of user turns by 63.6%. The improvements with these efficiency measures are statistically significant (*p=0.031* for **System Turns**, *p=0.016* for **User Turns**, and *p=0.017* for **Total Completion Time**).

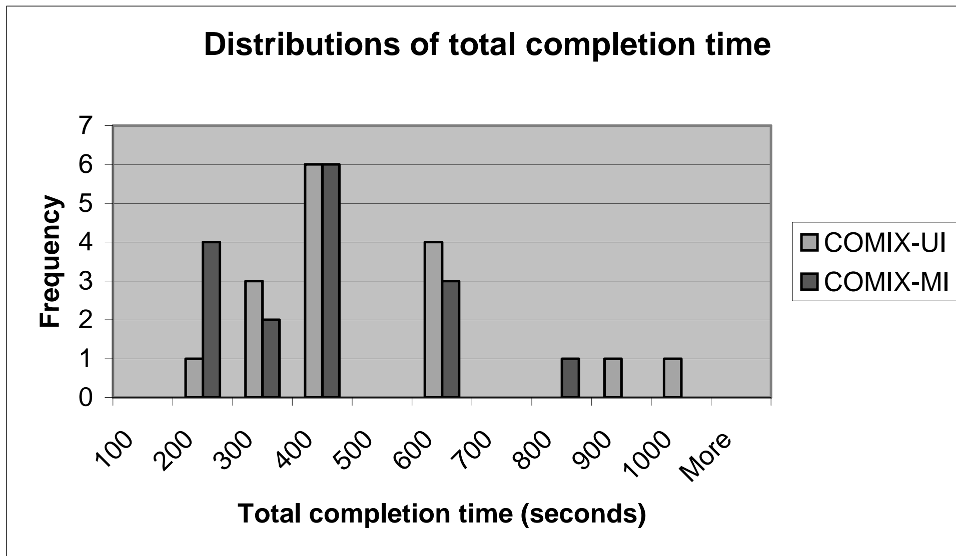


Figure 55: Distributions of total completion time for the over-constrained round-trip tasks.

Figure 55 presents the distributions of the total completion time for both system settings. A closer examination of the tasks shows that, with COMIX-MI, the total completion times for round-trips with one leg to relax all fall below 400 seconds, while the total completion times for tasks with two legs to relax exceed 300 seconds. This distinction resulted in the two modes in the distribution. With COMIX-UI, we do not see this distinction. While in general, tasks with two legs to relax require more time than tasks with one leg to relax, the time ranges for the two types of tasks overlap to a greater extent (between 200 seconds and 600 seconds for tasks with one leg to relax and between 300 seconds and 1000 seconds for tasks with two legs to relax). As a result, only one mode is observed for this distribution.

Figure 56 presents the distributions of system turns and users turns for the over-constrained round-trip tasks. We observe that the distributions of system turns and user turns are tighter for COMIX-MI, and that the distributions of system turns and user turns are more spread out for COMIX-UI. This suggests that COMIX-UI is influenced to a greater degree by the variability in user's capability in finding relaxation solutions, while COMIX-MI evens out the variability of user's capability in finding relaxation candidates by offering suggestions.

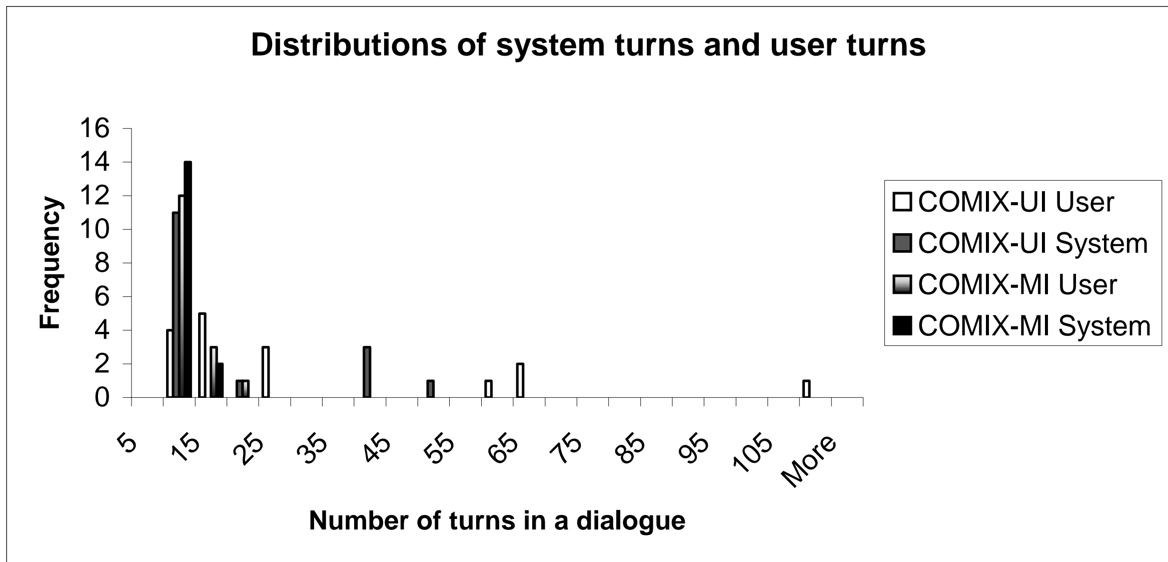


Figure 56: Distributions of system turns and user turns for the over-constrained round-trip tasks.

The subjective task success scores (i.e., **Task Success**) are very similar, as only one user gave up on one task. Task success as measured by Kappa, however, increased significantly by 13.5% from 0.693 by the COMIX-UI setting to 0.787 by the COMIX-MI setting (*p=0.035*). A look at Kappa score distributions (Figure 57) shows that although users achieve high Kappa scores for a majority of tasks with both system settings, the performance of the COMIX-UI setting is more spread out to the lower scores, while the performance of the COMIX-MI setting is clustered around the high scores. As there is not much difference in the subjective **Task Success** scores, it suggests that user's perception of task completion/success is not always reliable, and does not correlate well with the objective Kappa scores.

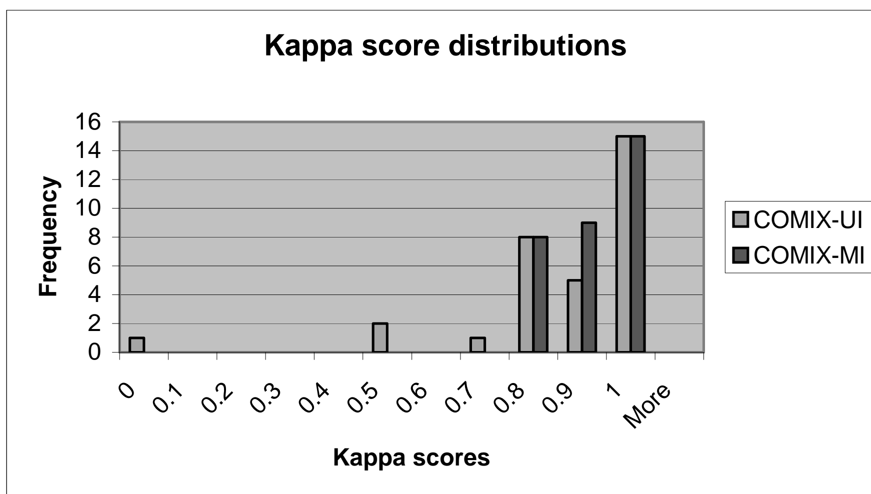


Figure 57: Kappa score distributions for the over-constrained round-trip tasks.

User satisfaction registers a small increase from 32.813 to 33.188. This increase, however, is not statistically significant.

In summary, our experiments show that the better performance we have observed with COMIX-MI in terms of dialogue efficiency and the Kappa scores for the round-trip tasks result primarily from the over-constrained round-trip tasks, for which COMIX-MI had the opportunities to offer cooperative relaxation suggestions. COMIX-MI also helps to even out the variability in user's capability in finding relaxation candidates. For the under-constrained round-trip tasks, solutions are found easily and COMIX-MI does not have much chance to offer any additional help; thus, no significant difference was observed between the two system settings. User's perception of task success (i.e., **Task Success**) and usability (i.e., **User Satisfaction**) is comparable for both system settings, but we have observed that user's perception of task success is not always reliable, and does not correlate well with the Kappa scores.

9.2.3. Comments from users

We collected users' feedback on both system settings through the post-task questionnaire and informal interviews. Users made comments with respect to the improvement of the interface and the system's task initiative-taking capabilities. General comments/suggestions related to system capabilities include:

- Both system settings are easy to work with for getting the right flights (e.g., "If you have experience with travel sites such as Travelocity, then there is no learning necessary to work with this system. Both systems are very easy to work with and very robust.")
- Users like the system suggestions when the tasks are over-constrained (e.g., when commenting on the COMIX-MI setting, a user said "It saves a lot of trial and error if the system can do the relaxation for you. I probably should have given this setting higher scores compared with the other system.")
- For easy tasks, the preference strength specification takes time and is not very useful (e.g., "Probably it is not necessary to specify the preference strength explicitly for every task. I keep them in mind when I do the search. Maybe the system can require that information only when it is necessary.")

General comments/suggestions related to interface design include:

- It would be desirable to have price as a determining factor in searching for flights (e.g., “When I book a flight, one of the most important factors for me is price. It would be desirable for a real system to use price as a factor in searching for flights.”)
- It would be desirable for the system to automatically guess the right city names in case of misspellings (e.g., “I don’t know how to spell Minneapolis. In addition to telling me that I had a misspelling, the system should provide a list of suggested city names.”)

Users did not show any difficulty in expressing preferences with strengths such as required or weak, even though a strong value in one problem might be different in preference strength from a strong value in another problem. This is possible because the users had no difficulty in establishing a ranking (or hierarchy) of the constraints in terms of their preference strengths. Other than the required constraints, constraints of other preference strengths were subsequently assigned symbolic values reflecting their respective ranks in the hierarchy.

Overall, the users believed that both system settings were robust and had the capability for carrying out the tasks and that COMIX-MI’s capability in providing suggestions in over-constrained situations was a desirable feature to use in information access tasks.

Chapter 10 Conclusions and Future Work

10.1. Conclusions

The thesis explores initiative-taking, cooperative dialogue behaviors in information dialogues, focusing on under-constrained and over-constrained dialogue situations. The focus of this work is on developing computational models that enable information systems to detect under- or over-constrained situations, and to take problem-solving initiative through cooperative dialogue behaviors appropriate for such situations. We have presented a constraint-based problem-solving model that supports the generation of initiative-taking actions in information-seeking dialogues. The model makes use of several AI techniques such as constraint satisfaction, solution synthesis, and constraint hierarchy to generate and maintain parallel partial solutions that can be exploited for solving under-constrained and over-constrained dialogue situations. Within this model, we have investigated different heuristic methods for selecting questions for resolving under-constrained situations and for selecting relaxation candidates for resolving over-constrained situations. Some of these methods make use of constraint preferential information or knowledge of dynamic solution structures from the solution synthesis network. In addition, we have investigated the interaction between the question selection methods and the relaxation selection methods. The effectiveness and efficiency of these heuristic methods have been evaluated empirically through simulated human-computer airline information-seeking dialogues and through a user study.

In the simulated information-seeking experiments, we have compared four question selection heuristics – question selection based on maximum branching, question selection based on maximum information gain, question selection based on constraint hierarchy, and question selection based on fixed ordering – with a baseline random question selection method. We also compared two relaxation candidate selection heuristics – relaxation candidate selection based on constraint hierarchy and relaxation candidate selection based on minimum solution size – with a baseline chronological backtracking method. These heuristics were evaluated in terms of dialogue efficiency and task success. To improve dialogue efficiency, we have aimed to reduce the number of questions the system needs to ask from the user in under-constrained situations and the number of relaxation suggestions the system initiates in over-constrained situations. Reducing both numbers is tantamount to reducing the dialogue turns between the user and the system. Another aspect of dialogue efficiency is the computation time required for the system to identify the question or relaxation candidates. We want to minimize the computation time as well.

To gain insight between problem structure and the performance of the heuristic question selection and relaxation selection methods, we have examined four experimental parameters: task difficulty, goal state size, interval size, and task complexity. The following observations drawn from the simulated experiments generalize across varying settings of these parameters.

For question selection, the heuristics based on maximum branching and constraint hierarchy are generally more efficient than the other methods in terms of the number of questions the system needs to elicit from the user. For example, when compared with the baseline random question selection method (the Random-Simple method) for the 3-destination flight reservation problems, the MaxBranch-Simple method reduces the number of questions by 25.7%, while the Const-Simple method reduces the number by 16.1%. Between these two heuristics, the maximum branching heuristic is generally more efficient than the one based on constraint hierarchy.

The heuristic based on maximum branching, however, takes significantly more computation time for selecting questions compared with either the Const-Simple method or the Random-Simple method.

In an information dialogue system, the questions the system elicits from the user are communicated through dialogue turns, thus taking communication time between the system and the user. The time for identifying the questions requires system computation time. Which heuristic method to use for question selection depends on the aspect of efficiency the system designer aims to maximize.

The heuristics based on maximum branching and constraint hierarchy are also generally more efficient than the other methods in terms of the number of relaxation candidates for resolving over-constrained dialogue situations. For example, compared with the baseline Random-Simple method for the 3-destinations flight reservation problems, the MaxBranch-Simple method reduces the number of relaxation candidates by 48.8%, while the Const-Simple method reduces the number by 43.6%. Corresponding to the reduction in the number of relaxation candidates, the time required for identifying the relaxation candidates is reduced by 63.4% for the MaxBranch-Simple method and by 50.7% for the Const-Simple method.

When the question selection method is fixed, we have not observed any advantage by introducing the knowledge sources such as constraint hierarchy or solution size for identifying relaxation candidates. Combined with the observations mentioned earlier on relaxation efficiency, we conclude that the efficiency of identifying relaxation candidates is dominated by the question selection method rather than by the relaxation method. That is, the more efficient the question

selection method is, the more efficient relaxation selection is. This could result from the fact that question selection determines the solution structure, which consequently determines the efficiency of relaxation candidate selection.

No method has demonstrated a significant advantage in improving task success scores. This suggests that the heuristic methods contribute more to the efficiency aspect of problem solving rather than task success.

Overall, the simulated experiments have demonstrated that heuristics that take advantage of the knowledge sources such as constraint hierarchy and partial solutions in the constraint-based problem solver can effectively improve dialogue efficiency in both under-constrained situations and over-constrained situations.

In the usability study, we have implemented a form-based information-seeking dialogue system and evaluated the usability of the system with two different task initiative settings: user initiative and mixed-initiative. In the mixed-initiative setting, we have selected the constraint hierarchy based method, the overall most effective method in question selection and relaxation candidate identification, and investigated the usability of the task initiative-taking behaviors of the system supported by the method. We have focused on task initiative-taking actions in over-constrained situations for the mixed-initiative setting, and kept the system behaviors for under-constrained situations the same for both settings.

Within the scope of this user study, we have observed that there was no statistically significant difference in the usability of both system settings and there was no statistically significant difference in the users' perception of task success. Users gave both system settings high scores for user satisfaction and task success. Interviews with the users suggested that when system was robust enough, the users were generally happy with the system's performance, and believed they had successfully completed the tasks.

When task success was measured with the objective kappa scores, however, we have observed that the mixed-initiative setting significantly outperformed the user-initiative settings for more complex problems such as round-trip flight reservation tasks by 13.5% ($p=0.035$). Such difference resulted primarily from over-constrained problems in which the system with the mixed-initiative setting had a chance to take task initiative and provide relaxation candidate suggestions to the user.

The mixed-initiative setting significantly outperformed the user-initiative setting in terms of dialogue efficiency for the more complex round-trip tasks. It reduced the average number of system turns by 35.2%, the average number of user turns by 49.6%, and reduced the average task completion time by 14%. System initiative also helped even out the variability in users' capability in solving the tasks. With user initiative, task completion time and dialogue turns demonstrated a larger range of variation, while with mixed initiative, the system initiative-taking actions made the performance more homogenous.

We have observed no statistical significant difference between the two initiative settings for the simpler one-way airline reservation tasks. This suggests that task initiative-taking actions are most effective with more complex problems.

10.2. Future Work

This work can be extended in several ways. Major extensions of this work can be carried out in several directions: adaptive mixed-initiative behavior, user profiling, domain expansion, scalability, application to spoken dialogue systems, effective selection of initiative-taking actions, and expanded usability study.

10.2.1. Initiative and dialogue strategy adaptation

We have observed in the usability study that the extent of the contributions of initiative taking by the system depends to a great extent on the complexity of the tasks and user's problem-solving expertise. With simple information-seeking tasks (e.g., finding one-way flights), users are generally proficient in finding the solutions by themselves, so initiative-taking actions by the system only make limited contributions. With harder problems (e.g., finding round-trip flights for over-constrained information requests), initiative-taking actions by the system contribute more toward a higher rate of task success and improved dialogue efficiency. Therefore, it is desirable for the system to adapt its initiative-taking behaviors to the varying degrees of expertise of the users (and even the same user across dialogues) and the complexity of the tasks.

Recently, there is growing interest on adaptive dialogue systems. Chu-Carroll (2000) describes an adaptive mixed-initiative dialogue system that employs initiative-oriented strategy adaptation to automatically adapt response generation strategies. The system automatically determines the initiative distribution for the system's dialogue turn, and selects a set of dialogue acts, based on the

initiative distribution, to satisfy certain communicative goals. Empirical evaluation has demonstrated that both the mixed initiative and automatic adaptation aspects led to improved user satisfaction, improved dialogue efficiency, and higher overall dialogue quality (Chu-Carroll and Nickerson, 2000). Litman and Pan (2000) present an adaptive spoken dialogue system that can predict whether a user is having speech recognition problems in the course of the dialogue and automatically adapt its dialogue strategies based on its prediction. Whenever the system classifies the dialogue as problematic as a dialogue progresses, the system adapts to a more conservative set of dialogue strategies. Empirical evaluation demonstrates that the adaptive version significantly outperforms the non-adaptive version in terms of task success. We believe adaptation is an essential aspect of natural and successful dialogues and is an important research direction to pursue.

10.2.2. User profiles and personalization

In the simulated tasks and the user study tasks discussed in the earlier chapters, the tasks are independent of each other. In real world situations, a user tends to have stable preferences over time that an information system can use as context for personalization. For example, a student in Pittsburgh who is a reward member of US Airways would generally require the departure city to be Pittsburgh and strongly prefer to use US Airways as the carrier. When user profiles are available, the system can generate information-access interfaces tailored to the individual users, thus reducing the time needed for users to specify their information needs.

One way to acquire such user preferences is for the system to explicitly ask the user to provide basic user profile information. Another way, which is less burdensome for the user, is for the system to acquire user constraints and their strength values based on users' information-access sessions with such a system. A system with such learning capabilities also has the advantage of updating user profiles automatically through time without the need of periodically requesting user updates.

10.2.3. Applications to other domains

In this thesis, we have used the airline reservation domain as the basis for information-seeking dialogues. It assumes a domain model of structured knowledge. Therefore, the observations we have obtained should extend naturally to other information-seeking tasks based on structured databases. One such domain is online shopping, where the users seek products that match certain

constraints. For example, Chai et al. (2001) present a dialogue system that helps users in finding IBM notebook computers that satisfy certain specification requirements. Their usability studies have demonstrated that by incremental refinement of the users' queries, the system is able to find more user constraints and recommends a product that best matches the user's criteria. Such dialogue behaviors have improved dialogue efficiency by reducing the number of clicks and the average interaction time. They do not present, however, how the system decides on what constraints to introduce or relax in refining the user's queries. The heuristics we have investigated in this thesis should be beneficial for such a system in adopting the optimal strategies to further improve dialogue efficiency. Other possible application domains include consultation dialogues (Chu-Carroll and Carberry, 1995a), scheduling, and network configuration.

10.2.4. Scalability

While we have addressed the scalability issue to a certain extent (e.g., we have evaluated the heuristics up to 3-leg flight reservation tasks), the number of attributes and constraints are still small compared to some other applications, such as network configuration, where hundreds, or even thousands of computers are connected together following certain connectivity or requirement constraints. The applicability of the framework presented in this work to such type of problems remains to be investigated.

10.2.5. Application to spoken dialogue systems

In this work, evaluation of the question selection heuristics and relaxation candidate selection heuristics are carried out in a noise-free dialogue channel. In spoken dialogue systems, speech recognition errors, disfluency, user corrections, etc. are very frequent. Confidence measures are generally required for assessing the quality of word recognition, out of domain utterance detection, and concept recognition (San-Segundo et al., 2001). To apply the findings in this work, we need to explore how to incorporate confidence measures from the spoken dialogue systems into the solution synthesis and constraint satisfaction framework explored in this thesis.

10.2.6. Expanded usability evaluation

In the usability study using form-based dialogues, we have chosen to implement only a limited set of strategies that are supported by the constraint-based problem solver and evaluated in the simulation experiments. First, we have restricted ourselves only to the implementation of the task

initiative-taking strategies in over-constrained situations; initiative-taking strategies for under-constrained situations are not implemented, and the behaviors of the two system settings (COMIX-UI and COMIX-MI) are kept as the same for under-constrained situations. Second, among the possible task initiative-taking dialogue actions for over-constrained situations, we have limited the implementation to only the dialogue acts ProposeNewVal. The ProposeNewVal&Answer act is not implemented, even though it could improve usability for certain problems.

These limitations in implementation in the usability study suggest a number of ways through which the current form-based dialogue system and the usability evaluation can be extended:

- Incrementally implement the task initiative-taking strategies (RequestVal, ProposeVal, and ProposeVal&Answer) in under-constrained situations.

Currently, the collection of user constraints by the system is through a *static* form with a *fixed* set of objects requesting information needs from the user. With the task initiative-taking strategies implemented *incrementally*, the system could start with a fixed form of objects for all the users, but adapt its behaviors based on the user's partial constraints and partial solution status from the constraint-based problem solver. For instance, if a user is requesting flights between two hub cities, there is likely to be many flights. In this case, the system will continue to expect additional constraints from the user, such as departure date and departure time. If, in contrast, the user requests flights between two cities with fewer flights, then it is possible that after the user specifies the departure city and the arrival city, the solution set found by the system is small enough that no additional constraints are required and the system is ready to present the solutions. Such implementation will allow the system to adapt its dialogue behaviors specific to a user's information needs and the information available from the domain data and the problem solver. As a result, it is possible that the system would provide users with solutions that satisfy their information needs even before the users complete specification of their constraints.

- Implement the task initiative-taking strategies ProposeVal&Answer in under-constrained situations and ProposeNewVal&Answer in over-constrained situations.

While we have implemented the task initiative-taking dialogue actions RequestVal, ProposeVal, and ProposeNewVal via form-based objects, we did not implement the dialogue actions ProposeVal&Answer and ProposeNewVal&Answer. These two actions could provide useful information to the users because, with these two actions, the users are able to evaluate the partial results that support the system's initiative and make relevant refinement of their information needs

based on their judgment of the partial results. It would be interesting to investigate the effect of this kind of additional information – the partial solutions – on usability of the system.

10.2.7. Selection of initiative-taking dialogue actions

While we provide general conditions on how the constraint-based model supports the generation of different task initiative dialogue actions, we did not specify which dialogue action the system should choose when several dialogue actions are possible. It is possible that one action is more effective and efficient than another in certain dialogue conditions. For example, the ProposeVal&Answer action could be more efficient than the ProposeVal action, since by carrying out the former action, the answers that satisfy the user's information needs could be readily available. One way for deciding which action to choose is through a combination of learning algorithms and empirical evaluation techniques proposed recently by Walker et al. (1998).

Appendix 1: Pre-Experiment Questionnaire

Subject #:

Computer experience:

1. How long have you been using a computer?

    a. less than 1 year
    b. 1-3 years
    c. over 3 years

2. How many hours a week do you use a computer??

    a. less than 10 hours
    b. 10-20 hours
    c. 20-40 hours
    d. over 40 hours

Task experience:

1. Have you ever made airline reservations?

    a. yes
    b. no

2. If you have ever made airline reservations before, which methods have you used? (choose all that apply)

    a. travel agent

    b. by phone directly with airline

    c. internet web sites

    d. others, please specify

3. If you have ever made airline reservations from an Internet Web site before, comment on your experience:

Appendix 2: Instructions for Usability Study

You will be given the task of finding appropriate flights and are asked to solve these tasks by retrieving flight information from an airline database. The systems you will use for solving these tasks are: COMIX-MI and COMIX-UI. These two systems will be assigned to you in a random order.

For each system (COMIX-MI or COMIX-UI)

Training session: You will be given 2 tasks to solve using this system. Each task is described on a worksheet. After you solve each task, record the flight information in the worksheet. A system may give you the option to specify the preference strengths of travel need in the given tasks. If the options are available, the task specification provides you with pre-assigned strength values. Here's how to interpret the strength values for a particular information need:

Required: the constraint is not changeable
Strong prefer: prefers not to be changed
Medium prefer: change if needed
Weak prefer: ok to change or doesn't matter

You want to find a flight that satisfies the travel preferences and their preference strengths as much as possible. If you cannot find such a solution, try to relax certain preferences. Answers are always available through constraint relaxation. In general, you want to relax the weaker preferences first before relaxing the stronger preferences. Preferences with required strengths should not be relaxed. If you find more than one solution, record the solution that best satisfies the original task. The system will be evaluated based on how well your solution satisfies the original task requirement.

Attached to the worksheet is a post-experiment questionnaire. Fill in the questionnaire after you finish each task. For the questionnaire, please choose only one answer for each multiple-choice question and select your answer based on your experience with solving the particular task at hand, not based on your overall impression of the system. If you have specific comments on your experience with the task (e.g., difficulties interacting with the system, suggestions for improvements), you can provide these comments in the Comments section in the questionnaire.

During the training session, you can ask questions on how to use the system for solving the task, how to fill in the worksheets, and how to answer the questions in the questionnaires.

Testing session: You will be given 8 new tasks to solve using this system. During the testing session, you are supposed to solve the tasks independently without consulting the experiment organizer. You're allowed to re-start the system if you feel the need to start over for a particular task. For each task, clearly write down your user ID (assigned by the experiment organizer), the system name, and the solution flight.

You can take a break between tasks, but do not take a break during a task session until you are finished with the task.

Questionnaire: After you finish with each task, please fill in the attached questionnaire. Again, do not forget to write down your user ID and the system name.

Appendix 3: Example Task Scenarios for Training Sessions

Task 100

You need to schedule a one-way trip from Beijing, P.R. China (PEK) to Pittsburgh, PA (PIT). You would want to take Northwest Airlines because you're a member of their reward program. The departure time should be around Sept. 27th, 2001 in the late afternoon after 5pm.

If the system allows you to specify the strengths of your travel preferences, the strength assignments should be as follows:

From PEK to PIT

Departure city: required (required value means "not changeable")
Arrival city: required (required value means "not changeable")
Departure date: medium prefer (medium value means "changeable if necessary")
Departure time: medium prefer (medium value means "changeable if necessary")
Airline: strong prefer (strong prefer means "prefer not to be changed")

Record your solution flight from PEK to PIT (if connecting, record both legs):

Flight #		Flight #	
Departure city		Departure city	
Arrival city		Arrival city	
Departure time		Departure time	
Arrival time		Arrival time	
Departure date		Departure date	
Airline		Airline	

Task 200

You need to schedule a round trip from Pittsburgh, PA (PIT) to Beijing, P.R. China (PEK). The travel time should be around September 6, 2001 in the late afternoon after 5pm. The return flight should be around September 15, 2001, but no later than that date.

If the system allows you to specify the strengths of your travel preferences, the strength assignments should be as follows:

<u>From PIT to PEK</u>

Departure city: required (required value means “not changeable”)
Arrival city: required (required value means “not changeable”)
Departure date: medium prefer (medium value means “changeable if necessary”)
Departure time: medium prefer (medium value means “changeable if necessary”)
Airline: weak prefer (weak value means “does not matter”)

<u>From PEK to PIT</u>

Departure city: required (required value means “not changeable”)
Arrival city: required (required value means “not changeable”)
Departure date: medium prefer (medium value means “changeable if necessary”)
Departure time: weak prefer (weak value means “does not matter”)
Airline: weak prefer (weak value means “does not matter”)

Record your solution flight from PIT to PEK (if connecting, record both flights):

Flight #		Flight #	
Departure city		Departure city	
Arrival city		Arrival city	
Departure time		Departure time	
Arrival time		Arrival time	
Departure date		Departure date	
Airline		Airline	

Record your solution flight from PEK to PIT (if connecting, record both flights):

Flight #		Flight #	
Departure city		Departure city	
Arrival city		Arrival city	
Departure time		Departure time	
Arrival time		Arrival time	
Departure date		Departure date	
Airline		Airline	

Appendix 4: Post-Task User Questionnaire

Please take a minute to answer the following questions.

Subject #: System Name: Task:

1. Did you complete the task and get the information you needed? (Task Success)
    a. Yes
    b. No

2. In this experiment, was it easy to find the flight you wanted? (Task Ease)
    a. Very easy
    b. Somewhat easy
    c. Neither easy or difficult
    d. Somewhat difficult
    e. Very difficult

3. How confident are you that you found all the relevant information? (Solution Confidence)
    a. Very confident
    b. Somewhat confident
    c. Undecided
    d. Not quite confident
    e. Not confident at all

4. In this experiment, did you know what you could do at each point of the dialogue? (User Expertise)
    a. Almost always
    b. Often
    c. Sometimes
    d. Rarely
    e. Almost never

5. In this experiment, did the system work the way you expected it to? (Expected Behavior)
    a. Almost always
    b. Often
    c. Sometimes
    d. Rarely
    e. Almost never

6. In this experiment, did the system provide you the functionality you needed to proceed at each point of the dialogue? (Behavior Completeness)
    a. Almost always
    b. Often
    c. Sometimes
    d. Rarely
    e. Almost never

7. In this experiment, did the system engage in the dialogue in an acceptable pace instead of being slow and sluggish? (System Speed)
    a. Almost always
    b. Often
    c. Sometimes
    d. Rarely
    e. Almost never

8. From your current experiment using the interface to get flight information, how likely will you use the interface regularly to obtain flight information in the future if the system is available? (Future Use)
    a. Definitely will
    b. Possibly will
    c. Undecided
    d. Possibly not
    e. Definitely not

General comments: comments on the system's usability. E.g., specify 2 or 3 things you like about the system and specify 2 or 3 things you don't like about the system? What features are useful to you? What features are not useful to you? What additional features you'd like to see implemented in the system?

Bibliography

Alicia Abella, Michael K. Brown, and Bruce Buntschuh. 1996. Development principles for dialogue-based interfaces. In *Proceedings of ECAI'96 Workshop on Dialogue Processing in Spoken Language Systems*, pages 1-7.

Alicia Abella and Allen L. Gorin. 1999. Construct algebra: analytical dialog management. In *Proceedings of the ACL'99*, pages 191-199.

James F. Allen and C. Raymond Perrault, 1980. Analyzing intention in utterances. *Artificial Intelligence*, 15:143-178.

Steve Beale. 1997. *Hunter-Gatherer: applying constraint satisfaction, branch-and-bound and solution synthesis to computational semantics*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. 1996. Constraint hierarchies. In Michael Jampel, Eugene Frender, and Michael Maher, editors, *Lecture Notes in Computer Science: Over-Constrained Systems*, pages 23-62.

Jaime J. Carbonell. 1983. Discourse pragmatics and ellipsis resolution in task-oriented natural language interfaces. In *Proceedings of the ACL'83*, pages 164-168.

Sandra Carberry. 1990. *Plan Recognition in Natural Language Dialogue*. ACL-MIT Press Series on Natural Language Processing. MIT Press, Cambridge, MA.

Jean C. Carletta. 1996. Accessing agreement on classification tasks: the kappa statistic. *Computational Linguistics*, 22(2):249-254.

Bob Carpenter. 1992. *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge.

Joyce Chai, Veronika Horvath, Nanda Kambhatla, Nicolas Nicolov, and Margo Stys-Budzikowska. 2001. A conversational interface for online shopping. In *Proceedings of HLT 2001: First International Conference on Human Language Technology Research*, pages 55-58.

Peter Pin-Shan Chen. 1976. The Entity-Relationship model – toward a unified view of data. *ACM TODS* 1(1).

Jennifer Chu-Carroll. 2000. MIMIC: An adaptive mixed initiative spoken dialogue system for information queries. In *Proceedings of the 6th Conference on Applied Natural Language Processing*, pages 97-104.

Jennifer Chu-Carroll and Michael K. Brown. 1997a. Initiative in collaborative interactions – its cues and effects. In *Technical Reports of AAAI-97 Spring Symposium on Computational Models for Mixed-Initiative Interaction*.

Jennifer Chu-Carroll and Michael K. Brown. 1997b. Tracking initiative in collaborative dialogue interactions. In *Proceedings of EACL-ACL'97*.

Jennifer Chu-Carroll and Sandra Carberry. 1995a. Generating information-sharing subdialogues in expert-user consultation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1243-1250.

Jennifer Chu-Carroll and Sandra Carberry. 1995b. Response generation in collaborative negotiation. In *Proceedings of the 33rd ACL*, pages 136-143.

Jennifer Chu-Carroll and Jill Suzanne Nickerson. 2000. Evaluating automatic dialogue strategy adaptation for a spoken dialogue system. In *Proceedings of the first Conference of the North American Chapter of the Association for Computational Linguistics*, pages 202-209.

Paul R. Cohen. 1995. *Empirical Methods for Artificial Intelligence*. The MIT Press, Boston.

M. Danieli and E. Gerbino. 1995. Metrics for evaluating dialogue strategies in a spoken language system. In *Technical Reports of the 1995 AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation*, pages 34-39.

Matthias Denecke and Alex Waibel. 1997. Dialogue strategies guiding users to their communicative goals. In *Proceedings of Eurospeech.*

Barbara Di Eugenio. 1987. Cooperative behavior in the FIDO system. *Information Systems*, 12(3):295-316.

Barbara Di Eugenio, Pamela W. Jordan, Richmond H. Thomason, and Johanna D. Moore. 2000. The agreement process: an empirical investigation of human-human computer-mediated collaborative dialogues. *International Journal of Human Computer Studies*. 53:1017-1076.

Toby Donaldson and Robin Cohen. 1997. A constraint satisfaction framework for managing mixed-initiative discourse. In *Technical Reports of AAAI-97 Spring Symposium on Computational Models for Mixed-Initiative Interaction*.

Bjorn Freeman-Benson and J. Maloney. 1989. The DeltaBlue algorithm: An incremental constraint hierarchy solver. In *Proceedings of the Eighth International Phoenix Conference on Computers and Communications*.

Bjorn Freeman-Benson, J. Maloney, and A. Borning. 1990. An incremental constraint solver. C*ACM*, 33(1):54-63.

Eugene C. Freuder. 1978. Synthesizing constraint expressions. *Communications of the ACM*, 21(11):958-966.

Eugene C. Freuder. 1995. The many path to satisfaction. In Manfred Meyer, editor, *Lecture Notes in Computer Science: Constraint Processing*, pages 103-119. SpringerVerlag.

Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence,* 58:21-70.

Eugene C. Freuder and Richard J. Wallace. 1997. Suggestion strategies for constraint-based MATCHMAKER agents. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, pages 105-111. Technical Report WS-97-05. Menlo Park, Calif.: AAAI Press.

Nancy Green and Sandra Carberry. 1999. A computational mechanism for initiative in answer generation. *User Modeling and User-Adapted Interaction*, 9(1&2):93-132.

H. P. Grice. 1975. Logic and conversation. In P. Cole and J. Morgan, editors, *Syntax and Semantics*. Academic Press, New York.

Barbara J. Grosz. 1977. *The Representation and Use of Focus in Dialogue Understanding*. Ph.D. thesis, University of California, Berkeley.

Barbara J. Grosz and Candace L. Sidner. 1986. Attention, intentions and the structure of discourse. *Computational Linguistics*, 12(3):175-204.

Curry I. Guinn. 1995. *Meta-Dialogue Behaviors: Improving the Efficiency of Human-Machine Dialogue - A Computational Model of Variable Initiative and Negotiation in Collaborative Problem-Solving*. Ph.D. thesis, Department of Computer Science, Duke University.

Peter A. Heeman and Susan E. Strayer. 2001. Adaptive modeling of dialogue initiative. In *Proceedings of Adaptation in Dialogue Systems Workshop*.

Pamela W. Jordan and Barbara Di Eugenio. 1997. Control and initiative in collaborative problem solving dialogues. In *Technical Reports of AAAI-97 Spring Symposium on Computational Models for Mixed-Initiative Interaction*.

Samuel Jerrold Kaplan. 1979. *Cooperative Responses from a Portable Natural Language Data Query System*. Ph.D. thesis, School of Computer and Information Science, University of Pennsylvania.

Diane J. Litman and Shimei Pan. 2000. Predicting and adapting to poor speech recognition in a spoken dialogue system. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'00)*.

Diane J. Litman, Shimei Pan, and Marilyn A. Walker. 1998. Evaluating response strategies in a web-based spoken dialogue agent. In *Proceedings of ACL/COLING'98: 36th Annual Meeting of the Association of Computational Linguistics*, pages 780-786.

John Maloney. 1991. *Using Constraints for User Interface Construction*. PhD thesis, Department of Computer Science and Engineering, University of Washington. Published as Department of Computer Science and Engineering Technical Report 91-08-12.

B. Raskutti and I. Zukerman. 1997. Generating queries and replies during information-seeking interactions. *International Journal of Human Computer Studies*, 47(6):689-734.

R. W. Smith. 1992. Integration of domain problem solving with natural language dialog: The missing axiom theory. In *Proceedings of Applications of AI X: Knowledge Based Systems*, pages 270-278.

Norman M. Sadeh and Mark S. Fox. 1995. Variable and value ordering heuristics for the job scheduling constraint satisfaction problem. Technical Report CMU-RI-TR-95-39, School of Computer Science, Carnegie Mellon University.

R. San-Segundo, B. Pellom, K. Hacioglu, W. Ward, and J.M. Pardo. 2001. Confidence Measures for Dialogue Systems. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-2001).*

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

Edward Tsang. 1993. *Foundations of Constraint Satisfaction*. Academic Press, London.

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

Marilyn A. Walker. 1993. *Information Redundancy and Resource Bounds in Dialogue*. PhD thesis, The Institute for Research in Cognitive Science, University of Pennsylvania, IRCS Report 93-45.

Marilyn A. Walker, Diane Litman, Candace Kamn, and Alicia Abella. 1997. PARADISE: A general framework for evaluating spoken dialogue agents. In *Proceedings of the 35th Annual Meeting of the Association of Computational Linguistics*, pages 271-280.

Marilyn A. Walker, Jeanne C. Fromer, and Shrikanth Narayanan. 1998. Learning optimal dialogue strategies: A case study of a spoken dialogue agent for email. In *ACL/COLING'98: Proceedings of the 36th Annual Meeting of the Association of Computational Linguistics*, pages 1345-1351.