Constraint-Based Strategies for Matchmakers *

Eugene C. Freuder and Richard J. Wallace
Department of Computer Science
University of New Hampshire
Durham, NH 03824
ecf,rjw@cs.unh.edu

Abstract

We describe a paradigm for content-focused matchmaking, based on a recently proposed model for constraint acquisition-and-satisfaction. Matchmaking agents are conceived as constraint-based solvers that interact with other, possibly human, agents (Customers). The Matchmaker provides potential solutions ("suggestions") based on partial knowledge, while gaining further information about the problem itself from the other agent through the latter's evaluation of these suggestions. The dialog between Matchmaker and Customer results in iterative improvement of solution quality, as demonstrated in simple simulations. This paradigm also supports "suggestion strategies" for finding acceptable solutions more efficiently *or* for increasing the amount of information obtained from the Customer.

Introduction

Intelligent matchmakers can be regarded as a third generation tool for Internet accessibility, where hypertext constitutes the first generation, and search engines the second. "Content-focused matchmaker" agents can provide advice to internet consumers (people or other agents) about complex products (Gomez *et al.* 1996). The reigning paradigm for such agents is the "deep interview", as embodied in the PersonaLogic website (Krantz 1997), where the primary mode of interaction is the Matchmaker query, which is essentially a multiple-choice question about product features. We propose a constraint-based paradigm, with a very different form of interaction. (Earlier discussions of this work can be found in (Freuder & Wallace 1997) (Eaton, Freuder, & Wallace 1998).)

In this paradigm, the primary mode of interaction is the *suggestion*, made by the Matchmaker to the Customer. The Matchmaker suggests a product to the Customer. The secondary mode of communication is the *correction*, made by the Customer to the Matchmaker, indicating how the suggestion fails to meets the Customer's needs. We believe this form of interaction is more natural and shifts more of the burden from the Customer (who may be a person) to the Matchmaker.

The Matchmaker could be an impartial matchmaker or a vendor. The product could be a physical product, e.g. a car, or an information source, e.g. a web page. The Customer could also be a computer agent. In future work on multi-agent systems we envision Matchmakers playing the role of Customer with other Matchmakers to procure information for their clients, and Matchmakers seeking compromise solutions for multiple clients.

We model the intelligent matchmaker paradigm using formal methods drawn from the study of constraint satisfaction problems (CSPs). The Matchmaker's knowledge base and Customer's needs are both modeled as a network of constraints. A suggestion corresponds to a solution of a CSP. A correction specifies Customer constraints that the proposed solution violates. Repeating the cycle of suggestion and correction allows the Matchmaker to improve its picture of the Customer's problem until a suggestion constitutes a satisfactory solution. The problem of both acquiring and solving a CSP has been termed the "constraint acquisition and satisfaction problem" (CASP) in (Freuder 1995), where the basic suggestion/correction model was suggested but not implemented.

The constraint network representation supports the computation of suggestions and easily incorporates corrections. In computing suggestions the constraint solving process infers the implications of corrections in a manner which avoids the need to make all constraints explicit. We believe that this form of model-based representation will be easier to build and maintain than the rule or decision tree based representation that presumably underlies a deep interview matchmaker.

The objective here is to model a situation in which Customers do not enter the interaction with a fully explicit description of their needs. They may be unfamiliar with what is available in the marketplace. They recognize their constraints during the interaction with the Matchmaker. They cannot list all their requirements up front, but they can recognize what they do

*This material is based on work supported by the National Science Foundation under Grant No. IRI-9504316.

not want when they see it. We believe this to be a common form of customer conduct. (Picture yourself browsing through a store or a catalogue, or interacting with a salesclerk.)

The Matchmaker can facilitate this process by an appropriate choice of suggestions (tentative solutions). For example, some suggestion strategies may lead to a satisfactory solution more easily for the user than others, e.g. with fewer iterations of the suggestion/correction cycle. On the other hand, ease of use is not the only possible goal. In some cases, it may be desirable for the Matchmaker to learn as much as possible about the Customer's constraints, to facilitate future interactions. In many cases, the Matchmaker can come up with a satisfactory solution *before* acquiring all of the Customer constraints. (Some constraints will be fortuitously satisfied by the suggestion.) Thus we use the number of Customer constraints acquired by the Matchmaker as another criterion when comparing suggestion strategies.

Background: CSPs and CASPs

A constraint satisfaction problem (CSP) involves assigning values to *variables* that satisfy a set of *constraints*. Each constraint is a relation based on the Cartesian product of the *domains*, or allowable assignments, of a subset of variables. In the present work all constraints are binary, i.e., they are based on the domains of two variables. A binary CSP is associated with a constraint graph whose nodes represent variables and arcs represent constraints.

CSPs have four basic types of parameter: number of variables, number of values in a domain or domain size, number of constraints, and number of value tuples in a constraint. In practice, if the domain size is the same for all variables, we refer to it as the value of a single domain size parameter. Otherwise, we often use an aggregate measure like the mean as a representative parameter value. Number of constraints is usually expressed in relation to the total number of possible constraints in a graph of n variables and is referred to as problem density. Most often constraint sizes are expressed in a complementary way, as the (relative) number of unacceptable tuples, or *tightness* of a constraint. Again, if tightness varies among constraints, we refer to average tightness as a representative value.

In a constraint acquisition and satisfaction problem (CASP) the constraint solver must acquire information about the constraints before it can solve the problem. The situation can be conceptualized by assuming some universe of constraints, i.e., all the constraints which can possibly be part of the CASP. (In the extreme case, this would be the complete graph based on the known variables.) A certain set of constraints within this universe forms the current problem, P. The CSP solver (here the Matchmaker) knows a subset of the constraints in P at the outset, call it K, but it must in fact solve problem P. It will, therefore, have to acquire knowledge about the remaining constraints in P before it can find a satisfactory solution in a reasonable amount of time.

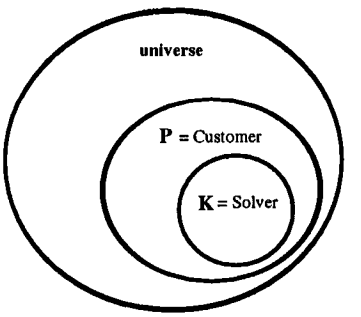
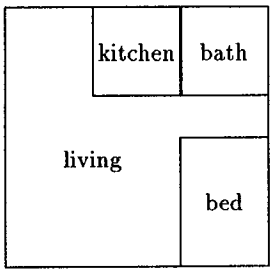


Figure 1: The three classes of constraints that set the scene for a Matchmaking dialog: those initially known to both Solver and Customer (K), those known (perhaps implicitly) to the Customer (P), and the universal set of constraints, which includes all those that might have been part of the problem.

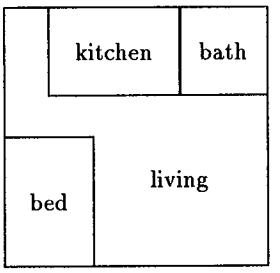
CASPs, Agents, and Matchmaking

Matchmaking based on the CASP paradigm involves two agents, the CASP Solver and the Customer. In this situation, the Customer 'knows' the problem to be solved, but not so explicitly that it can tell the Solver outright. The Solver elicits some of this Customer knowledge by suggesting a solution based on the constraints that it (the Solver) knows about. The Customer then evaluates the solution to determine whether there are constraints of concern that are violated. These violations are communicated to the Solver, which incorporates this information as constraints between the variables involved in each violation. The Solver then solves the new CSP and presents this solution as a new suggestion to the Customer. This communication cycle is repeated until the solution is fully satisfactory to the Customer, i.e., none of the latter's constraints are violated.

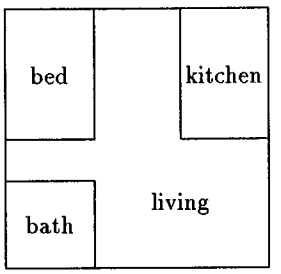
To get some idea of how this might work in practice, we will consider the following example. A customer comes into a real estate office looking for an apartment. She knows that she wants a one-bedroom apartment within a certain price range. The agent has several apartments available, so they decide to go and have a look at them. The first apartment they look at has this layout:



On looking it over, the customer immediately realizes that the kitchen is much too small. So they go to another apartment:



The customer finds the kitchen satisfactory, but now she realizes she prefers having the bathroom next to the bedroom, as it was in the first apartment. So they look at a third apartment:



Apartment three satisfies all the customer's constraints, so she deems it acceptable.

Suggestion Strategies

In the situation just described, the agent might have chosen to show the apartments in a different order. In particular, after showing Apt. 1, he might have taken the customer to see Apt. 3. In this case, the customer's problem would have been solved in two steps rather than three. However, in this case the constraint involving the locations of the bedroom and bathroom would have been satisfied fortuitously. By showing Apt. 2, the agent discovered this constraint, thus learning more about the customer's problem. This is, of course, an example of the tradeoff between the criteria of efficiency and information gain that were described in the Introduction.

Examples like this give us reason to expect that the two goals of efficiently solving the Customer's problem and making that problem explicit are potentially divergent. Because of this, we would like to examine heuristics from the CSP domain that bear on this question. In so doing we will consider the situation in more abstract terms, even if some of the strategies we consider are not directly applicable to an example such as the one above, e.g. because they put too great demands on the Customer.

If our goal is to limit the number of iterations in the suggestion-correction cycle, there are two approaches we might take. One is to try to find solutions more likely to satisfy constraints between variables, even if these constraints are not presently in the Solver's representation. This policy is, therefore, one of maximizing satisfaction, specifically, the number of satisfied constraints. An alternative, possibly more perverse, approach is to maximize constraint violations. Here, the policy is to find solutions that violate as many constraints as possible so that more constraints are incorporated into the Solver's set from the start.

Fairly straightforward methods for finding solutions under either policy can be derived from current knowledge of constraint satisfaction. These methods depend on the kind of procedure used in the solution process. For algorithms that use complete or exhaustive search, selecting values less likely to be in conflict with values in other variables is a reasonable method for maximizing satisfaction. To realize this strategy we can order the values in each domain by maximum averaged promise, where "promise" is the relative number of supporting values in each adjacent domain (Geelen 1992). For hill-climbing or heuristic repair methods, a strategy in the same spirit is solution reuse, i.e., starting each search with the solution obtained earlier, after revising the information about conflicts based on the last Customer communication. A complete search strategy that conforms to the policy of maximizing violations is the converse of the satisfaction strategy: choose values *most* likely to be in conflict (specifically, values in each domain are ordered by minimum promise). A corresponding hill-climbing strategy is to search each time from a new location, i.e., with a new set of initial values.

If our goal is to learn as much as possible about the Customer, where "learning about the Customer" means learning his or her constraints, then intuitively, maximization strategies do not appear well-suited to this goal, but violation strategies should serve this purpose at least as well as that of maximizing efficiency. This in turn suggests that satisfaction strategies will be subject to a tradeoff between the two different goals, while violation strategies might overcome it. On the other hand, it is not clear which kind of strategy will be most efficient in finding an acceptable solution. If a satisfaction method is much better than any others, then it may be necessary to consider this tradeoff carefully.

These ideas have been supported by experiments that simulate a Customer-Solver dialog under simplified conditions. An example is shown in Figure 2, using the complete methods described above. (These curves

are for one problem, but the qualitative differences seen here were found with all problems tested.) These tests involved a strategy for satisfying constraints as quickly as possible (max-promise) and two strategies for maximizing constraint violations: min-promise, described above, and a series of random value-orderings (called "shuffle"). Each curve ends when a solution is found that satisfies all of the Customer's constraints.

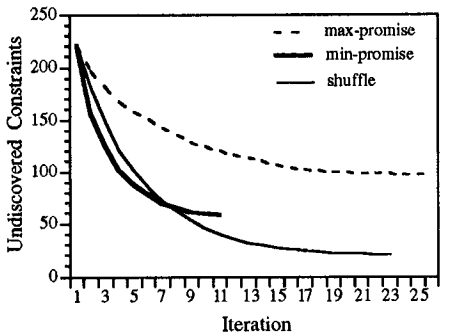


Figure 2: Undiscovered Customer constraints after successive iterations with three value orderings.

During early iterations, both min-promise and shuffle discover many constraints, but the curve for the former levels out more quickly, and this procedure also finds a completely satisfactory solution more quickly, so its curve is shorter. Consequently, the curve for shuffle falls below the other curve on the eighth iteration, but it continues on for some time before finding a satisfactory solution. The curve for max-promise remains well above the other two throughout the dialog. And it requires even more iterations than shuffle to find a fully satisfactory solution. Thus, for these problems, the strategy of violating constraints satisfies both goals, and therefore overcomes the tradeoff between efficiency and information gain.

References

Eaton, P. S.; Freuder, E. C.; and Wallace, R. J. 1998. Constraints and agents: Confronting ignorance. *AI Magazine* (to appear).

Freuder, E. C., and Wallace, R. J. 1997. Suggestion strategies for constraint-based matchmaker agents. In *Constraints and Agents: Papers from the 1997 AAAI Workshop*, 105–111.

Freuder, E. C. 1995. Active learning for constraint satisfaction. In *Active Learning. AAAI-95 Fall Symposium Series, Working Notes*, 34–35.

Geelen, P. A. 1992. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings ECAI-92*, 31–35.

Gomez, J.; Weisman, D. E.; Trevino, V. B.; and Woolsey, C. A. 1996. Content-focused matchmakers (excerpt). In *Money & Technology Strategies*, volume 2(3). Forrester Research, Inc.

Krantz, M. February 17, 1997. The web's middleman. *Time* 67–68.