From: AAAI Technical Report WS-97-05. Compilation copyright © 1997, AAAI (www.aaai.org). All rights reserved.

Suggestion Strategies for Constraint-Based Matchmaker Agents

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

Abstract

In this paper 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 (Clients or 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. We also show empirically that this paradigm supports "suggestion strategies" for finding acceptable solutions more efficiently *or* for increasing the amount of information obtained from the Customer. This work also indicates some ways in which the tradeoff between these two metrics for evaluating performance can be handled.

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 forthcoming Consumer's Edge website (Krantz 1997), where the primary mode of interaction is the query, made by the Customer to the Matchmaker. We propose a constraint-based paradigm, with a very different form of interaction.

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 well be a person) to the Matchmaker.

A small example will motivate and illustrate the basic paradigm. Suppose we are a matchmaker agent for an interior decorating firm, and we are planning a living room decor for a customer. As part of this exercise we propose that two chairs and a sofa be placed along one wall of the living room. The customer remonstrates; he wants a lamp next to the sofa. We add a constraint to our living room representation to limit the objects that may be placed beside the sofa. Solving the new problem, we propose a sofa with a lamp on one side and a chair on the other. We add that the sofa and the chair are both blue. At this point the customer objects to having two pieces of furniture side by side of the same color. Therefore, we add a color constraint to the representation and solve the new problem. Now we propose a sofa with a lamp on one side and a chair on the other; the sofa is blue and the chair coral, a solution the customer finds satisfactory.

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, and indeed in our experiments we use a computer agent to simulate a Customer. 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 the Customer's needs are both modeled as a network of constraints. A "suggestion" corresponds to a solution of a CSP. A "correction" specifies the 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 imple-

mented.

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 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 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. In this paper we present experiments that provide empirical evaluation of some simple suggestion strategies.

Ease of use is not the only evaluation criteria. In an environment in which the Matchmaker has an ongoing relationship with the Customer, it can be desirable for the Matchmaker to learn as much as possible about the Customer's constraints, to facilitate future interactions. In our implementation it is possible, in fact it proves experimentally the norm, for the Matchmaker to 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 performance metric when comparing suggestion strategies.

Notice that this latter metric is somewhat antithetical to the ease of use criteria. Acquiring many Customer constraints can be viewed as good, because it facilitates future interaction; however, it also might be viewed as bad, because it requires more Customer effort (in the form of corrections). There is a similarly double-edged situation, analogous to one encountered in the classical information-retrieval literature, that we plan to pursue in future research: we would like to model the situation in which some suggestions lead to a satisfactory solution quickly, while others lead to a more satisfactory solution, but at greater "cost" to the Customer. However, even with our simple initial model of the matchmaker process we encounter some interesting empirical behavior.

The contributions of this paper are:

- A new matchmaker agent paradigm.
- A constraint-based implementation of this paradigm.
- Basic suggestion strategies for complete and stochastic matchmakers.
- Basic metrics for strategy evaluation.
- Experimental evaluation of 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. The number of tuples in a constraint may refer to the number of acceptable tuples. More 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.

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.

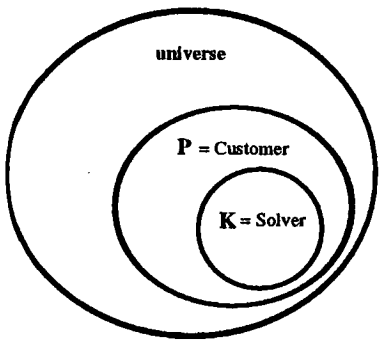

Figure 1: This figure shows the three classes of constraints that set the scene for a Matchmaking dialog: those initially known to both Solver and Customer (K), those known (perhaps implicitly) to the Customer (P), and the universal set of constraints, which includes all those that might have been part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

In the present work we simplify this Matchmaking Dialog between Solver and Customer in the following ways. We assume that both the Solver's and the Customer's constraints are drawn from the same universe; hence, when the Solver is apprised of a constraint violation it does not have to decide what the constraint actually is, i.e., the set of acceptable tuples. Further, constraints known to the Solver are assumed to be a proper subset of the Customer's (implicit) constraints (Figure 1). In terms of the CASP definition in Section 2, the former is set K, and the latter can be identified with P. (In an overall Customer-Solver Dialog, the constraints in K might be determined by preliminary questioning [what kinds of furniture are you looking for?, how large is your living room?] before the first solution is presented.) We also assume that on each iteration of the communication cycle, the Customer gives the Solver the complete set of constraints violated by the last solution.

Suggestion Strategies

Faced with a repeated cycle of communications between two agents, we would like to find ways to limit the length of this dialog. One approach is to try to find solutions that are more likely to satisfy constraints between variables, even though 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 and 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.

Fortunately, 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 promising method for maximizing satisfaction. 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 that are *most* likely to be in conflict. A corresponding hill-climbing strategy is to search each time from a new location, i.e., with a new set of initial values.

As suggested earlier, a second goal of Matchmaking that will sometimes be important is to learn as much as possible about the Customer during an interaction. Here, "learning about the Customer" means learning the Customer's constraints. 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.

We therefore have an important potential tradeoff to consider - between efficiency in finding a satisfactory solution and thoroughness in discovering Customer constraints. It would appear that satisfaction strategies would be subject to this tradeoff, while violation strategies might overcome it. But it is not clear *a priori* which kind of strategy will be most efficient in finding a satisfactory solution. If a satisfaction method is much better than any others, then it may be necessary to consider this tradeoff carefully. (This is not the only possible tradeoff;- others were encountered in our experimental work, as described in Section 5.2.)

Of course, our decisions will also reflect our overriding goals (as indicated in the Introduction). If we are only interested in a solution to the present problem, then efficiency may be our only concern, but if we are also interested in learning as much as we can about the Customer, we may want a procedure that can handle both criteria effectively.

To better understand how different procedures chosen with the above goals in mind will perform in prac-

tice and the degree to which the tradeoffs discussed here are important, we now turn to empirical investigations of suggestion strategies.

Experimental Evaluation of Suggestion Strategies

Methods

Tests were made with random problems; for brevity, we concentrate on one problem set and mention results for other sets of problems in passing. This set included ten problems with 50 variables and a constant domain size of 5. A fully connected graph of constraints was obtained in all cases by first generating a random spanning tree for the variable set. The density, in terms of edges added to the spanning tree, was fixed at 0.25 (giving 343 constraints). Relative tightness was allowed to vary, although there was a fixed probability of including a given tuple in a constraint; with this many constraints, the average tightness was almost exactly equal to the stipulated probability of 0.18. Problems with these parameters are in or near the critical region for computational complexity; those in the present set were the first ten generated that had complete solutions.

Two kinds of algorithms were tested: (i) a complete CSP algorithm, forward checking with conflict-directed backjumping and dynamic ordering by domain size (Prosser 1993), (ii) a heuristic repair method, min-conflicts augmented with a random walk strategy (Wallace 1996). In the latter procedure, after a variable in conflict has been chosen, a value is chosen at random from the domain with probability p, while the usual min-conflicts procedure is followed with probability $1 - p$.

For the complete algorithm, suggestion strategies were devised by ordering domain values in specific ways prior to search. To maximize constraint satisfaction, values in each domain were ordered by maximum averaged promise (max-promise), where "promise" is the relative number of supporting values in each adjacent domain, and these proportions are averaged across all adjacent domains. A violation strategy was obtained simply by reversing this order for each domain, which gave an ordering by minimum averaged promise (min-promise). Another violation strategy was to 'shuffle' the domains before each search, i.e., to order each domain by random sequential selection of its values. In addition, lexical ordering of values served as a control.

For the heuristic procedure, a possible maximization strategy is "solution reuse": each iteration after the first begins with the solution obtained at the end of the last iteration. Two kinds of suggestion strategy were tested as candidate violation strategies. In one case the walk probability was raised from 0.10 to 0.35; since the latter allows more random selection of values, a greater variety of solutions might be found, leading to more violations. The other kind of strategy was a "reset" strategy: each iteration begins with an initial assignment generated from scratch. In addition, resetting was combined with three different preprocessing strategies: (i) min-conflicts greedy preprocessing based on a single variable ordering, where for each successive variable a value is chosen that minimizes conflicts with existing assignments of values to variables, (ii) greedy preprocessing based on a different, randomly selected, variable ordering for each iteration, (iii) hill-climbing from an initial random assignment on each iteration, i.e., for each variable a value is chosen at random and its conflicts with previous assignments recorded. Since the walk strategy is independent of the reset strategies, all combinations of the two were tested.

For each Matchmaking Dialog, the constraint set of the original CSP was the universal set. From this universal set, constraints were chosen by random methods to be in K and P. At the beginning of each dialog, the full constraint set was scanned and with probability p_k, a given constraint was added to ***both*** the Solver's and the Customer's constraint sets. (These constraint, therefore, comprise K.) If the constraint was not chosen, then with probability p_p, it was added to the Customer's set, P. From this, it is easy to determine expected values for number of initial Solver constraints, Customer constraints not in K, and constraints that are in neither the Solver's nor the Customer's constraint sets.

Four sets of values were used for p_k and p_p, respectively, in the procedure just described: 0.2 and 0.4, 0.2 and 0.8, 0.4 and 0.4, 0.4 and 0.8.

Experimental Results

Representative results for the different value orderings used with the complete CSP algorithm are shown in Table 1. The same pattern of results was found with 0.4, 0.4 and with 0.4, 0.8 for p_k and p_p. And similar results were obtained for another set of random problems that were based on a different model of generation and that had greater variability in constraint tightness.

In one case (0.2, 0.4), the satisfaction strategy, max-promise, found acceptable solutions after fewer iterations in comparison with either the lexical ordering or the constraint violation strategies. But with more Customer constraints in relation to Solver constraints, a violation strategy, min-promise, was more efficient in this respect than max-promise. In both cases, min-promise was the most effective in uncovering violations quickly, as reflected in the measure of violations per iteration. On the other hand, the shuffling procedure found more violations across the whole dialog.

The tradeoff expected with max-promise was very much in evidence, since this procedure uncovered far fewer Customer constraints than any other; this was true for both pairs of p_k, p_p values. This tradeoff was also found for the shuffling procedure: while it was the least efficient ordering, it uncovered more constraints than any other procedure.

Interestingly, the min-promise ordering required relatively few iterations, while finding more violations per iteration on average than the other orderings, and in this respect it tended to overcome the tradeoff between efficiency and constraint discovery. In fact, when there were more Customer constraints not in the initial Solver set, this ordering was better than the satisfaction ordering, max-promise, on both metrics.

Table 1: Matchmaking Dialog Statistics for Different Value Orderings

constr probs*	val ord	iterats to sol	violat /iter	undis const	sol sim	time (sec)
.2, .4						
	lex	8	8	59	.72	.03
	max	6	5	85	.83	.02
	min	8	11	37	.61	.02
	shuf	14	8	15	.22	.02
.2, .8						
	lex	12	14	69	.54	.55
	max	15	9	105	.65	.43
	min	11	17	47	.48	.53
	shuf	19	11	21	.24	.41

Notes. Means based on 10 problems and five dialogs, or initial K and P, per problem. Algorithm is FC-CBJ with dynamic ordering by domain size. * prob(shared constraint), prob(tester constraint). *iterats* are number of iterations before a solution was found. *violat/iter* are mean number of violations discovered by tester on one iteration. *undis const* are mean number of undiscovered tester constraints at end of a run. *sol sim* is mean similarity (proportion of common values) of successive solutions found during a run. *time* is runtime for entire dialog. Expected numbers of shared and tester-only constraints are 69 and 110 for the 0.2, 0.4 condition and 69 and 220 for the 0.2, 0.8 condition.

From these results alone, one would conclude that max-promise should be chosen to maximize efficiency if the difference between the number of Customer and Solver constraints is small, but min-promise is preferrable when the difference is large. If one wants to maximize constraint discovery overall, then the shuffle should be chosen. But for the best tradeoff between efficiency and constraint discovery, min-promise should be chosen.

Unfortunately, perhaps, the picture changes when efficiency is measured in terms of time instead of iterations. For overall time to complete a dialog, max-promise is better than min-promise, especially when there are more Customer constraints (the reverse of what was found when we considered iterations). This difference may be due to the greater time required to find a solution when starting from the least supported values. Surprisingly, shuffle is essentially as fast as max-promise. (For the 0.4, 0.8 case, where the times were about twice as long as for 0.2, 0.8, shuffle was somewhat slower, but both were almost twice as fast as the other two orderings.)

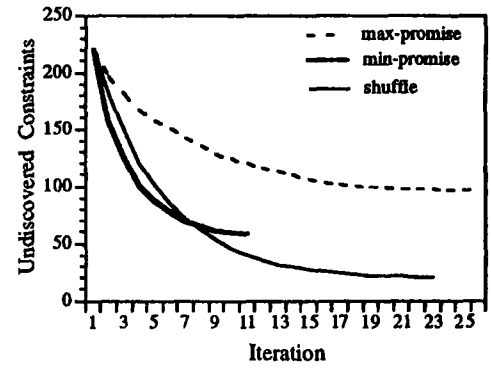

Figure 2: Undiscovered Customer constraints after successive iterations with three value orderings. Condition involved inclusion probabilities of 0.2 and 0.8 (cf. explanation under Methods). Means based on five dialogs with one problem.

These results show that, when efficiency is measured in units of time rather than iterations, max-promise is still the best strategy in terms of efficiency. However, now it is shuffle, rather than min-promise, that best overcomes the tradeoff between efficiency and constraint discovery. This does not leave us with a simple decision, since either measure of efficiency may be more appropriate in different contexts. In particular, a half-second increase in summed search time may be much more palatable than presentation of five additional solutions.

Although the summary data in Table 1 imply something about trends during a dialog, it is useful to consider curves for some of these measures across an entire dialog. Perhaps the most informative is the curve for number of undiscovered constraints (Figure 2). The curves in Figure 2 are for one problem, but the qualitative differences seen here were found with all problems tested. During the early iterations, min-promise finds the most constraints, but its curve levels out more quickly than the curve for shuffle, and it 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. The curve for max-promise remains well above the other two throughout the dialog.

These curves suggest yet another basis for choosing between strategies for constraint discovery, *if* one is willing to accept a partial solution. If a limit is put on the number of iterations in a dialog, then, if that limit is low (< 8 in Figure 2), min-promise is the strategy of choice. But if the limit is sufficiently high, the shuffling should be chosen.

Curves for solution similarity across successive iterations are also of interest, since solution similarity may be important psychologically in interactions with a human customer. This measure tends to increase in the

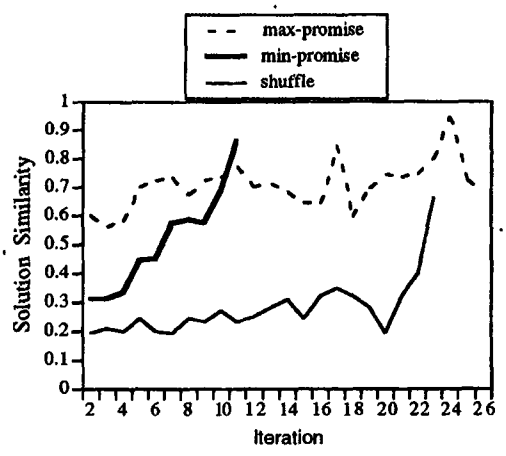

Figure 3: Similarity of successive solutions during a dialog, for three value orderings. Same condition and problem as in Figure 2.

course of a dialog regardless of the ordering (Figure 3; Again, the same trends were found for all ten problems.) For min-promise, successive similarity is fairly low at first, but it rises sharply. Similarity values for max-promise are high throughout the dialog. In contrast, similarity values for shuffle are fairly low through most of the dialog.

In experiments with heuristic repair (Table 2), some of the same differences are seen between satisfaction and violation strategies, and to some degree there are similar kinds of tradeoffs. Solution reuse, the satisfaction strategy, finds acceptable solutions after fewer iterations, and after less time, than any of the violation strategies. On the other hand, all of the latter are more successful in discovering Customer constraints. The most successful is the restart strategy that uses a different random value assignment at the beginning of each search; this procedure discovers as many constraints as the shuffled ordering with the complete algorithm. Unfortunately, success in discovering constraints is purchased with a considerable increase in runtime, and none of the procedures really overcomes this tradeoff. However, there is a partial mitigation in that the best restart strategies are just as successful with the lower walk probability as with the higher, although the runtime is much lower in the former case.

Concluding Comments

This work introduces a new strategy for Customer-Matchmaker interaction based on software agents performing the Matchmaker functions (and possibly playing the role of Customer as well). We have evaluated several strategies that may be useful in this context. We have also identified an important and interesting tradeoff between the goals of efficient problem solving and knowledge acquisition, and our experimental work already provides suggestions for handling it.

Table 2: Matchmaking Dialog Statistics for Hill-Climbing Strategies

restart proced	walk probs	iterats	violat / iter	undis const	sol sim	time (sec)
0.2, 0.4 constraint probs.						
no restart (solution reuse)						
	0.10	6	9	70	.83	.18
	0.35	7	9	65	.81	.22
restart - mincon preproc						
	0.10	9	7	61	.74	.60
	0.35	9	7	57	.70	.65
restart - random var mincon						
	0.10	14	6	35	.47	1.03
restart - random value						
	0.10	14	8	15	.22	1.26
0.2, 0.8 constraint probs.						
no restart (solution reuse)						
	0.10	14	12	76	.71	2.12
	0.35	15	13	47	.55	8.12
restart - mincon preproc						
	0.10	18	11	41	.43	17.0
	0.35	18	11	32	.36	28.6
restart - random var mincon						
	0.10	18	12	29	.31	15.3
restart - random value						
	0.10	18	12	24	.27	14.5

Notes. Same problems as in Table 1, five dialogs per problem. Minconflicts hill-climbing with random walk. Restart procedures described in text; for randomized orderings both walk probabilities gave almost identical results except for runtimes. Other abbreviations as in Table 1.

Work now in progress is beginning to examine the matchmaking process in a more realistic setting. This work will incorporate preferences on the part of the Customer, to replace the simple ok/not-ok decisions made in the present experiments, and will enlarge on the Solver's learning capacities, so it can determine the structure of the Customer constraints when these are not known *a priori*.

This work also carries the implication that a multi-agent system may be well-suited for solving the Constraint Acquisition and Satisfaction Problem. There is an obvious division of labor between the information acquisition and CSP-solving aspects of CASPs, and it is likely that this can be mapped directly onto different agents in many situations.

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

References

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

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.

Prosser, P. 1993. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence* 9:268–299.

Wallace, R. J. 1996. Analysis of heuristic methods for partial constraint satisfaction problems. In Freuder, E. C., ed., *Principles and Practice of Constraint Programming - CP'96*, volume 1118 of *Lecture Notes in Computer Science*. Berlin: Springer. 482–496.