Open Constraint Satisfaction

Boi Faltings and Santiago Macho-Gonzalez

Artificial Intelligence Laboratory (LIA),
Swiss Federal Institute of Technology (EPFL),
IN-Ecublens, CH-1015 Ecublens, Switzerland,
`boi.faltings|santi.macho@epfl.ch`,
`http://liawww.epfl.ch/`

Abstract. Traditionally, constraint satisfaction has been applied in *closed-world* scenarios, where all choices and constraints are known from the beginning and fixed. With the Internet, many of the traditional CSP applications in resource allocation, scheduling and planning pose themselves in *open-world* settings, where choices and constraints are to be discovered from different servers in a network.
We examine how such a distributed setting affects changes the assumptions underlying most CSP algorithms, and show how solvers can be augmented with an information gathering component that allows open-world constraint satisfaction. We report on experiments that show strong performance of such methods over others where gathering information and solving the CSP are separated.

1 Constraint Satisfaction in Distributed Information Systems

Constraint satisfaction has been applied with great success to resource allocation, scheduling, planning and configuration. Traditionally, these problems are solved in a *closed-world* setting: first all variables, domains, constraints and relations are defined, then the CSP is solved by a search algorithm.

With the increasing use of the Internet, many of the problems that CSP techniques are good at now pose themselves in a distributed setting. For example, in personnel allocation, it is possible to obtain additional staff on short notice. In configuration, it is possible to locate additional suppliers of parts through the internet.

This change in setting makes a fundamental difference to the underlying constraint satisfaction problem. Most successful CSP methods, in particular constraint propagation, are based on the closed-world assumption that the domains of variables are completely known and fixed. In an open setting, this assumption no longer holds, making completeness and termination of search algorithms are more complex issue.

In an open world, there are also fundamental questions about the semantics of a CSP. When we combine options for a CSP from different sources, one might suspect that the right way to form the combination would be to form a CSP whose solutions are the union of the solutions of the CSPs being combined. However, this would not make sense if we are trying to add options to make an unsolvable problem solvable.

Finally, the criteria for performance shift as each information gathering step is orders of magnitude more expensive than searching for solutions themselves. Consequently, good heuristics for information gathering are more important than efficient search heuristics.

In this paper, we define *Open Constraint Satisfaction Problems* (OCSP). We examine relevant related work in the CSP and database communities, and then propose solutions for two major issues: the semantics of combining CSP, and integrating information gathering and solving of OCSP. Finally, we report on some initial experiments that show the strong performance gains that OCSP bring over carrying out information gathering and CSP solving sequentially.

2 Open Constraint Satisfaction Problems

In this section, we define a formal framework for open constraint satisfaction problems.

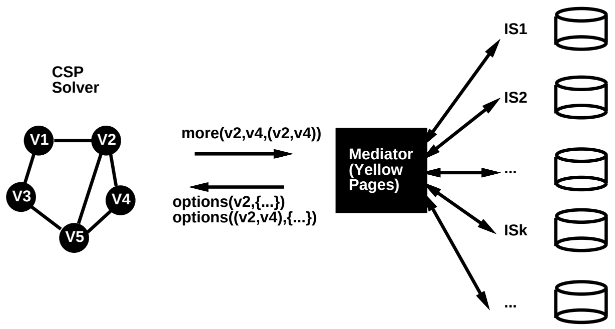

Fig. 1. *Elements of an open constraint satisfaction problem*

We consider the setting shown in Figure 1 which reflects the important elements that occur in an open setting. The CSP solver can access an unbounded set of information servers through a mediator. The mediator is a directory that indexes the information that can be found on the information servers. Such directories already exist in unstructured form (Yahoo), and industry is working towards formal models based for example on the UDDI standard. For the purposes of our research, we consider that this technology implements a functionality whereby the CSP solver can obtain additional domain values:

- using the `more(vi,...,(vi,vj),...)` message, it can request the mediator to gather more values for these variables. The mediator will then contact randomly selected information servers. While servers are selected randomly, any server on the network will eventually be contacted.

- using `options(vi,...)` and `options((vi,vj),...)` messages, the mediator informs the CSP solver of additional domain values or constraint tuples found in the network. This response will return what was found on the servers, so it may be empty or not contain any new values.
- when there are no more values to be found, the mediator returns `nomore(vi,...)` to inform the problem solver of this.

Note that as we assume an unbounded number of information sources, it is not even possible to gather all information in a single database and then start the problem-solving process, as has been implemented in many distributed information systems ([1, 2, 4]).

We now define:

Definition 1. *An* open constraint satisfaction problem *(OCSP) is a possibly unbounded, partially ordered set $\{CSP(0), CSP(1), ...\}$ of constraint satisfaction problems, where CSP(i) is defined by a tuple $< X, C, D(i), R(i) >$ where*

- *$X = \{x_1, x_2, ..., x_n\}$ is a set of n variables,*
- *$C = \{(x_i, x_j, ...), (x_k, x_l, ...), ...\}$ is a set of m constraints, given by the ordered sets of variables they involve,*
- *$D(i) = \{d_1(i), d_2(i), ..., d_n(i)\}$ is the set of domains for CSP(i), with $d_k(0) = \{\}$ for all k.*
- *$R(i) = \{r_1(i), r_2(i), ..., r_m(i)\}$ is the set of relations for CSP(i), each giving the list of allowed tuples for the variables involved in the corresponding constraints, and satisfying $r_k(0) = \{\}$ for all k.*

The set is ordered by the relation $\prec$ where $CSP(i) \prec CSP(j)$ if and only if $(\forall k \in [1..n])d_k(i) \subseteq d_k(j)$, $(\forall k \in [1..m])r_k(i) \subseteq r_k(j)$, and either $(\exists k \in [1..n])d_k(i) \subset d_k(j)$ or $(\exists k \in [1..m])r_k(i) \subset r_k(j)$.

A solution *of an OCSP is a combination of value assignments to all variables such that for some i, each value belongs to the corresponding domain and all value combinations corresponding to constraints belong to the corresponding relations of $CSP(i)$.*

3 Related work

Within the CSP community, the work that is closest to ours is *interactive constraint satisfaction* (ICSP), introduced in [5]. Similarly to our work, in ICSP domains are acquired incrementally from external agents. The forward checking algorithm is modified so that when domains become empty, it launches a specific request for additional values that would satisfy the constraints on that variable. In earlier work ([6]), the same authors also show how arc consistency algorithms can be adapted with the right dependency structures so that consistency can be adapted to values that might be added later. However, ICSP has a strong focus on the efficiency of the CSP search algorithm rather than on minimzing information gathering; it typically gathers significantly more values than necessary. It also does not address the problems of an open environment, in particular it limits itself to finite domains and assumes that variable domains can be exhaustively retrieved from the information agents.

Open constraint satisfaction bears some ressemblance to the dynamic constraint satisfaction problem (DCSP), where constraints are added and removed over time. Bessiere ([7]) has shown methods for dynamically adapting consistency computations to such changes. However, dynamic CSP methods require that the set of all possible domain values is known beforehand, and thus do not apply to the OCSP problem. Another major difference is that OCSPs are restricted to a monotonic ordering of domains and values, while DCSP allow adding and removing variables in any order.

Another related area is distributed CSP(DisCSP), investigated in particular by Yokoo ([9]) and more recently also other researchers. DisCSP does not require agents to announce the complete variable domains beforehand, so by its formulation it would also allow them to be open. However, all known search algorithms for solving DisCSP rely on closed-world assumptions over variable domains for initiating backtracks. As DisCSP also require each variable to be controlled by a single agent, they also assume that all domain values are known and fixed by a single agent during the search, and so they do not address the context posed by OCSP either.

There has been some research into using constraints as a formalism for representing and integrating information, in particular the KRAFT project ([10]) and the FIPA CCL content language ([11]). These address in particular issues of how to represent constraints in languages such as XML so that it is easy to carry out composition. They will be important for practical implementations of OCSP.

Research in the database community has addressed issues of information gathering and information retrieval, starting with federated databases ([12]), then dynamic information integration ([13, 1]), and finally multi-agent information systems such as InfoSleuth ([2, 3]). Significant work has gone into matchmaking between queries and information sources. In our research, we use an ontology-based classification similar to that of [14]. There are significantly more complex matchmaking techniques such as the Information Manifold ([4]). Decker and Sycara ([15]) investigate the efficiency of middle-agent systems, and Sycara ([16]) elaborates on their use as information agents. Techniques such as LARKS ([17]) show that much more complex classification than simply ontologies are possible. Thus, there is a sufficient technology base for implementing the mediator functionality we assume in this paper.

Recently, researchers in information retrieval have paid more attention to driving information retrieval from the task that users are trying to solve. Systems such as Watson and I2I ([18]) and just-in-time information retrieval ([19]) automatically retrieve information from databases, mail archives and other information sources by matching it with keywords that occur in the current activity of the user - for example, a document being prepared.

4 Using CSP for Information Integration

Solving OCSP implies that CSPs from different sources must be integrated. We assume that the mediator performs the necessary translations so that it returns additional options as domains and relations of a CSP whose variables and constraints are compatible with those of the problem being solved.

There are two different ways that the new domains and relations might be integrated:

- *conjunctive combination*, meaning that the new information represents additional constraints that any solution must satisfy. This occurs for example when we want to schedule a meeting between many participants and need to integrate all their time constraints.
- *disjunctive combination*, meaning that the new information represents additional options and thus enables additional solutions. This occurs for example when a problem-solver obtains schedule information for the same route from different airlines.

Conjunctive combination is a common case in constraint satisfaction algorithms and handled simply by having all relations present simultaneously. Dynamically adding and removing constraints using conjunctive combination has been addressed for example in dynamic constraint satisfaction ([8, 7]). For solving OCSP, we particularly need *disjunctive* combination, which so far has not received much attention in the CSP community (but see [10, 11]).

CSP formulations have commonly been compared through the solutions they admit ([20]). This works well for conjunctive combination, where the solutions of the combined problem is the intersection of the solutions of the components, but is too restrictive for disjunctive combination: when we are looking for additional options to make a scheduling or configuration problem solvable, we are often combining subproblems that have no solutions to obtain a problem that has solutions.

Thus, in accordance with the definition of OCSP, we define the disjunctive combination of two CSP as follows:

Definition 2. *The* disjunctive combination *of CSP1* $= < X_1, D^1, C_1, R^1 >$ *and CSP2* $= < X_2, D^2, C_2, R^2 >$ *is CSP3* $= < X_3, D^3, C_3, R^3 >$ *where*

- $X_3 = X_1 \cup X_2$
- $D^3 = \{d_j^1 | x_j \in X_1 \wedge x_j \notin X_2\} \cup \{d_j^2 | x_j \in X_2 \wedge x_j \notin X_1\} \cup \{d_j^1 \cup d_j^2 | x_j \in X_1 \cap X_2\}$
- $C_3 = C_1 \cup C_2$
- $R^3 = \{r_j^1 | c_j \in C_1 \wedge c_j \notin C_2\} \cup \{r_j^2 | c_j \in C_2 \wedge c_j \notin C_1\} \cup \{r_j^1 \cup r_j^2 | c_j \in C_1 \cap C_2\}$

The difficulty with disjunctive combination is that it is incompatible with the pruning and constraint propagation techniques that have been fundamental to most constraint satisfaction algorithms, and requires additional restrictions on the model in order to give the desired results. The example in Figure 2 illustrates the difficulties.

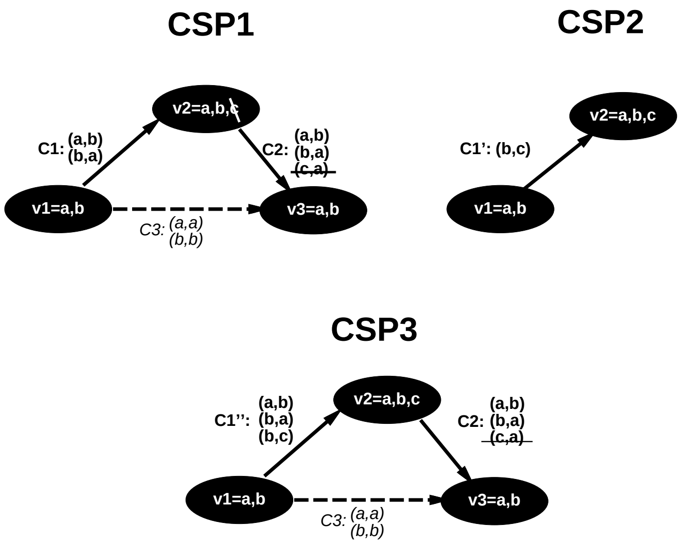

Fig. 2. *Possible problems with disjunctive combination of two CSP.*

In CSP1, we can use constraint propagation to prune values c from variable $x2$ and then tuple (c, a) from constraint $c2$. When we later combine with CSP2, we do not obtain any additional solutions. But if we had combined with CSP1 without pruning, we would have also obtained the solution $x1 = b, v2 = c, v3 = a$. The problem here is that pruning relies on a closed-world assumption: it does not eliminate valid solutions, but only under the condition no additional options are added. While it is easy to recognize this situation when solving a CSP, we have to avoid cases where pruning has already been incorporated in the constraint model. In order to characterize this property, we define:

Definition 3. *A CSP is* redundantly expressed *if it contains a relation or domain whose validity depends on other relations or domains in the CSP.*

A constraint model can also pose problems in the other direction. Consider that CSP1 might have originally had an additional constraint C3. In the database model, this constraint could have been removed as being redundant: it already follows from C1 and C2. When we now combine with CSP2, we would also obtain the new solution $x1 = b, v2 = c, v3 = a$, which would violate the original constraint C3. To avoid this situation, we define:

Definition 4. *: A CSP is* completely expressed *if and only if for any pair of variables x and y and any pair of values $x = a$ and $y = b$ allowed by the constraint (if any) between them, there is a way to change the rest of the CSP so that they become part of a valid solution.*

In order to be composable, CSP models have to be exactly expressed:

Definition 5. *A CSP is* exactly expressed *if and only if it is completely but not redundantly expressed.*

as is shown by the following Theorem:

Theorem 1. *The disjunctive combination of two exactly expressed CSPs is itself exactly expressed.*

Proof. Disjunctive combination does not involve constraint propagation nor elimination of redundancies, so it does not affect the property of being correctly expressed.

This defines a restriction that the CSPs provided by the mediator as well as the CSP representations maintained while solving an OCSP must satisfy. Unfortunately, it turns out that this condition rules out the use of many interesting constraint propagation and preprocessing techniques. However, we will now see that there are other techniques that can be applied.

5 Algorithms for solving OCSP

We now consider algorithms for actually solving OCSP. The simplest algorithm is the brute-force algorithm: first collect all values from all information sources, and then run an efficient CSP solver to generate a solution to the problem. However, this can be very costly: imagine contacting all PC part manufacturers to figure out how to fix your video card!

It is clearly preferable to only gather information as needed, as shown by function `o-search`(Algorithm 1). If there is a solution within the values available

```
Function o-search(CSP)
$s \leftarrow$ solve($CSP$)
if $s \neq \{\}$ then
  return $s$ as a solution
$CSP_{incr} \leftarrow$ more($X \cup C$)
if nomore($X \cup C$) then
  return failure
$CSP_{new} \leftarrow$ disjunctive combination of $CSP$ and $CSP_{incr}$
o-search($CSP_{new}$)
```

Algorithm 1: *o-search: an incremental algorithm for solving OCSP.*

from the servers, this algorithm will eventually find it, since the mediator will eventually return every value or value combination that the servers can provide. However, it is not very efficient, since it blindly gathers values for any part of the CSP without focussing on those parts that caused the failure.

To reduce the amount of useless server accesses, information gathering must focus on finding additional options for the minimal unsolvable subproblems of the current instantiation of the CSP. Thus, we now show how to improve Algorithm 1 by identifying variables that participate in minimal unsolvable subproblems and gathering values for these individual variables. We then show that the resulting algorithm is complete, and how it can be generalized to the case where values for entire subproblems can be obtained from the information servers.

5.1 Integrating search and information gathering

In order to simplify the algorithms, we make the assumption that all domains of the OCSP are discrete, and that furthermore all constraints (including binary ones) have been encoded using the *hidden variable encoding* ([20, 21]). In this encoding, all constraints are represented as additional variables with tuple-valued domains representing the corresponding relations. The only remaining constraints are then equality constraints that ensure consistency between assignments to the original variables and the corresponding elements of the constraint tuples. Besides the fact that the hidden variable encoding has been shown to have desirable computational properties ([22]), it simplifies the OCSP formulation since now all constraints are fixed, and information gathering only concerns variable domains.

When a CSP has no solution, it is often the case that it contains a smaller subproblem that already has no solution. It will not be possible to create a solution by information gathering unless values are added to variables and relations of that subproblem. This fact can be used to more effectively drive information gathering. The idea is to find a variable that must be part of an unsolvable subproblem as a promising candidate for adding extra values. To develop this into a general and complete algorithm, we need to address two issues: how to identify unsolvable subproblems, and how to select all variables in turn to avoid unbounded information accesses while missing a feasible solution.

The following lemma provides the basis for identifying variables that are part of unsolvable subproblems:

Lemma 1. *Let a CSP be explored by a failed backtrack search algorithm with static variable ordering $(x_1, ..., x_n)$, and let x_k be the deepest node reached in the search with inconsistency detected at x_k. Then x_k, called the* failed variable*, is part of every unsolvable subproblem of the CSP involving variables in the set $\{x_1..x_k\}$.*

Proof. In order to reach x_k, the search algorithm has constructed at least one valid assignment to $x_1, ..., x_{k-1}$, so this set of variables does not contain any unsolvable subproblem. However, there is no consistent assignment to $x_1, ..., x_k$, so this set does contain unsolvable subproblem(s). Since the only difference is x_k, x_k must be part of all of these unsolvable subproblems. □

On the basis of this proposition, we can use the results of a failed CSP search process to determine for which variable additional values should be collected. These are then passed to the mediator, which will search for relevant information on the network. When there are no additional values for this variable, the mediator returns a `nomore` message, and other variables are then considered. The resulting algorithm **fo-search** (failure-driven open search) is shown in Algorithm 2.

```
1: Function **fo-search**(X,D,C,R,E)
2: $i \leftarrow 1, k \leftarrow 1$
3: **repeat** {backtrack search}
4:   **if** $exhausted(d_i)$ **then** {backtrack}
5:     $i \leftarrow i - 1$, $reset - values(d_i)$
6:   **else**
7:     $k \leftarrow max(k, i), x_i \leftarrow nextvalue(d_i)$
8:     **if** $consistent(\{x_1, ..., x_i\})$ **then** {extend assignment}
9:       $i \leftarrow i + 1$
10:    **if** $i > n$ **then**
11:      **return** $\{x_1, ..., x_n\}$ as a solution
12: **until** $i = 0$
13: **if** $e_k = closed$ **then**
14:   **if** $(\forall i \in 1..k-1) e_k = closed$ **then**
15:     **return failure**
16: **else**
17:   $nv \leftarrow$ **more**$(x_k)$
18:   **if** $nv =$ **nomore**$(x_k)$ **then**
19:     $e_k \leftarrow closed$
20:   $d_k \leftarrow nv \cup d_k$
21: reorder variables so that $x_k$ becomes $x_1$
22: **fo-search**(X,D,C,R,E) {search again}
```

Algorithm 2: *Function* **fo-search** *for solving OCSP.*

Algorithm 2 makes the assumption that variables are ordered by the index i. It assumes that no consistency techniques are used in the search, although the chronological backtracking can be replaced with backjumping techniques to make it more efficient.

We are now going to show that **fo-search** is a complete algorithm for solving OCSP. We start by defining:

Definition 6. *An* unsolvable subproblem *of size k of an instance $CSP(i)$ of an OCSP is a set of variables $S = \{x_{s1}, x_{s2}, ..., x_{sk}\}$ such that there is no value assignment $(x_{s1} \in d_{s1}, ..., x_{sk} \in d_{sk})$ that satisfies all constraints between these variables.*

and showing the following property:

Lemma 2. *Let $\mathcal{S} = \{S_1, ..., S_m\}$ bet the set of unsolvable subproblems of instance $CSP(i)$ of an OCSP. Then for any instance $CSP(j), CSP(j) \succ CSP(i)$, the set of unsolvable subproblems $\mathcal{S}'$ it contains is a subset of $\mathcal{S}$.*

Proof. Suppose that $CSP(j)$ contains a subproblem $S' \notin \mathcal{S}$, and let $S' = \{x_{t1}, ..., xtk\}$. By Definition 1, the domains $d_{t1}(i) \subseteq d_{t1}(j), ..., d_{tk}(i) \subseteq d_{tk}(j)$.

S' is solvable in $CSP(i)$, and the values used in its solution must also be part of the corresponding domains for $CSP(j)$. Thus, S' cannot be unsolvable in $CSP(j)$. □

This Lemma shows that the set of unsolvable subproblems in successive instances of an OCSP is monotonically non-increasing.

We now consider more closely Algorithm 2, and in particular the sequence of instances it solves in subsequent iterations. We have the following lemma:

Lemma 3. *Assume that the last $k+1$ calls to Algorithm 2 have been with instances $CSP(i_0, ..., CSP(i_k)$, that the algorithm has last searched variables in the order $x_{j1}, ..., x_{jk}$ and identified the k-th variable x_{jk} as the failed variable, and that the each of the instances $CSP(i_0), ..., CSP(i_k)$ has identical unsolvable subproblems. Then:*

- *in the last k calls, Algorithm 2 has called the mediator (function* **more***) exactly once for each of the variables $x_{j1}, ..., x_{jk}$;*
- *$S = \{x_{j1}, ..., x_{jk}\}$ is a minimal unsolvable subproblem of $CSP(i_k)$;*
- *the algorithm will continue to call* **more** *on each of the variables in S in turn until S becomes solvable.*

Proof. As the algorithm always puts the variable for which it has called more values as the first in the search, the first claim follows directly from the function of the algorithm.

Furthermore, S is unsolvable as no solution could be found by complete search. Suppose that it was not minimal, i.e. that there was a variable x_{il} such that $S' = S - x_{il}$ was also unsolvable. x_{il} was the failed variable when **fo-search** was run on $CSP(i_l)$, and that search must have included variable x_k. By Lemma 1, x_{il} was part of every unsolvable subproblem of $CSP(i_l)$ that also involves x_{ik}. But as x_{ik} is the failed variable of subproblem S, by Lemma 1, it is part of every unsolvable subproblem involving variables in S. Consequently, every unsolvable subproblem within S must also involve x_{il}.

The third claim follows from the fact that as long as S is unsolvable, running **fo-search** on $CSP(i_k)$ gives an identical result as running it on $CSP(i_0)$. □

We can now show completeness of Algorithm 2:

Theorem 2. *Supposed that OCSP is solvable, i.e. by calling* **more** *on every variable a sufficient number of times we eventually reach an instance $CSP(j)$ such that for all $CSP(m) \succ CSP(j)$, $CSP(m)$ contains no unsolvable subproblems. Then Algorithm 2 will eventually terminate with a solution. Thus, the algorithm is complete.*

Proof. $CSP(1)$ has finitely many variables and thus finitely many unsolvable subproblems of size at most n. Assume that the algorithm never finds a solution; then since by Lemma 2, the set of unsolvable subproblems is monotonically non-increasing, there must exist an infinite sequence of calls to **fo-search** such that the unsolvable subproblems are always identical. By Lemma 3, in such a sequence the algorithm will eventually call for additional values for each variable of the same unsolvable subproblem S. But since the OCSP is solvable, these calls must eventually return values that will make S solvable. Thus, the sequence of calls where subproblems remain unsolvable cannot be infinite. □

An interesting consequence of Theorem 2 is that if a problem is unsolvable and the set of available values is finite, the algorithm will stop while identifying a minimal unsolvable subproblem. This can be useful when it is possible to obtain information for several variables in parallel.

For efficiency reasons, it may be advantageous for the mediator to obtain values not only for single variables, but entire subproblems with a single query. Algorithm 2 can be modified for this case in two ways:

- it can not gather additional values until a minimal unsolvable subproblem is completely identified, and then call the mediator on that subproblem or subproblems that have a maximal overlap with that subproblem. However, it is important that every variable in the subproblem is eventually queried, for otherwise completeness is no longer ensured.
- it can gather additional values for all subproblems that include the last failed variable. This would preserve completeness of the algorithm, but may be less efficient than focussing search on the minimal unsolvable subproblem itself.

In all cases, it is important that the problem formulation used by the information servers is by CSP that are exactly expressed so that composition will produce correct results.

5.2 Efficiently integrating new values

Once additional values have been obtained, the search algorithm can be restarted to see if the problem now has a solution. We would of course like this search to avoid reexploring the search space that had already been explored unsuccessfully earlier. The most obvious solution, reusing the nogoods observed from the earlier search, is not practical since these nogoods may be invalidated by the new values.

The technique of decomposing a CSP into subproblems proposed by Freuder and Hubbe in [24] turns out to be useful here. They proposed to decompose a CSP to factor out unsolvable subproblems, thus limiting search effort to a smaller and solvable part. This idea applies well to OCSP: the new, combined problem can be decomposed into the old problem just searched (which is known to be unsolvable) and a new one based on the values just obtained. However, if we limit subsequent searches only to the newly found values for x_i, we loose the benefit of Lemma 1 and the algorithm is no longer complete. We can nevertheless use the subproblem decomposition to make algorithms more efficient by ordering the values so that the new values are explored first. In this way, the algorithm starts search with the new subproblem, and only revisits earlier assignments when this subproblem has been found to have no solution.

6 Experimental results

We tested the performance of the techniques we described on synthetic, randomly generated constraint satisfaction problems. As an example, we used resource allocation (equivalent to list coloring), which can be modelled as a CSP whose variable domains are resources and whose constraints are all inequalities (expressing the fact that the same resources cannot be used for different tasks at the same time).

Comparison metrics We compare the algorithms on two aspects. The first is the number of accesses to information sources required to find a solution to the OCSP, and measure the network traffic generated. Several metrics have been developed in the field of database selection ([25]). Since each variable must have at least one value, solving the CSP requires at least one information source access per variable, and this is the theoretical optimum. We measure performance by the ratio:

$$R = \frac{Number\ of\ variables\ of\ the\ CSP}{Number\ of\ access\ to\ IS\ until\ a\ solution\ is\ found}$$

Since each variable must have at least one value, solving the CSP requires at least one information source access per variable, so that the ideal value for R is 1. Smaller values of R mean low efficiency. We consider R a good measure of the relative amount of information gathering effort generated by the different methods, but it does not take into account possible parallization or buffering.

Experiments and results The experiments followed the following steps:

1. Generate a random coloring problem, with between 3 to 10 variables, 3 to 13 values per variable, and random inequality constraints so that the graph is at least connected and at most complete.
2. Distribute the values of the variables in a fixed set of n information sources. The results reported here are for 12 information sources, but do not change substantially when the number of sources is increased or decreased.
3. Use different algorithms to find the first solution to the problem, and measure the efficiency ratio described above.

We compare the performance of different combinations of algorithms in the mediator and the problem solver. For the mediator, we consider the following algorithms:

– *Brute Force:* gather all values from all relevant information sources into a single database, then search for a solution with these values.
– *Random:* The mediator considers only information sources indexed under the given property and concept, and treats them in random order.
– *Size:* The mediator considers the same information sources as above, but in decreasing order of the number of values they carry for the different properties.

For the problem solver, we consider the two algorithms given earlier, namely **OS** for `o-search`, obtaining new values for variables randomly, and **FO** for `fo-search`, where search for new values is driven by the failures of backtrack search. Furthermore, we also compare the algorithms with interactive CSP ([5]).

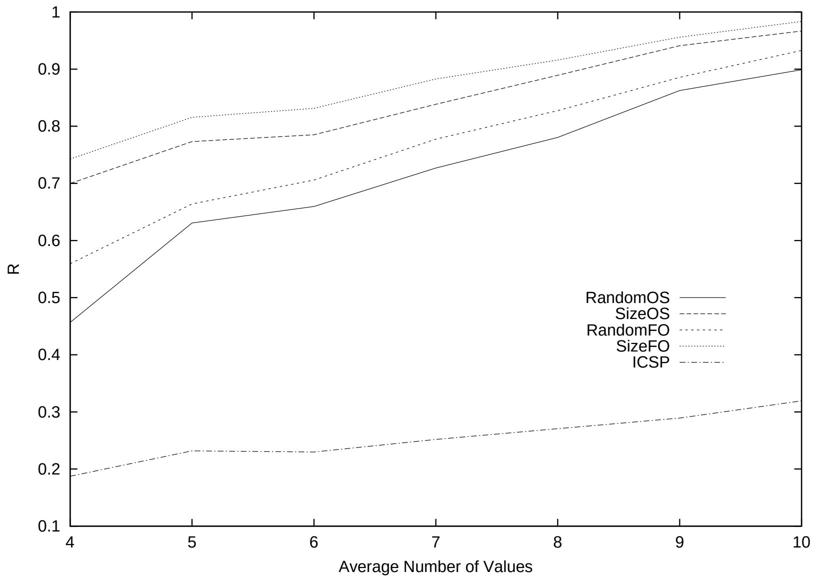

Fig. 3. *Efficiency ratio against number of values for several combination of search/mediator algorithms.*

Figure 3 plots the efficiency ratio against the average number of values available for each variable for a setting in which there are a total of 12 information servers. The more values there are for each variable, the easier the problem is to solve, and we can see that the average efficiency in general increases with the number of available values. On the other hand, efficiency can be observed to decrease slightly with the size of the problem. We have not observed any significant dependency of information server accesses on constraint density.

When problem-solving and information gathering are not coupled at all, problem-solving would require accessing all the servers, resulting in an efficiency ratio of 12; this curve is not shown in the graph in Figure 3. Thus, the idea of coupling the two processes definitely provides very large efficiency improvements.

We can also observe a significant improvement over interactive constraint satisfaction as described in [5], which is natural as this algorithm does not provide a method for choosing a variable and preselecting values by constraints is not feasible in an open environment.

The best method is, as expected, a combination of failure-driven open search combined with an indexing of the database on size; this gives an efficiency approaching the theoretical optimum. It appears furthermore that information about the size of information servers plays a bigger role than directing the search for the right variable, as the next runner-up is the algorithm combining size with open-search (OS). In practice, this information is very difficult to provide, so that the improvements obtained by the CSP search algorithm are of great pratical interest.

7 Conclusions

Many new and exciting applications in open information systems, in particular the WWW, address problems which CSP techniques are very good at solving. Such applications will appear increasingly with the emergence of web services and the semantic web. We have defined Open Constraint Satisfaction Problems (OCSP) as a formulation that addresses this open setting.

The first contribution of this paper is the definition of semantic properties that allow incrementally combining values in an OCSP while maintaining a correct solution set. These result in conditions that the problem formulations must satisfy in order to be combinable.

The second contribution is to have shown an effective method for identifying minimal unsolvable subproblems and thus focussing information gathering. Based on this, we have given an algorithm that is provably complete even for unbounded variable domains, and demonstrated that on random coloring problems, it achieves a performance very close to the theoretical minimum as far as accesses to information servers is concerned.

In particular, the gains tend to increase with both the number of information servers and the number of values they provide. Thus, the technique is likely to be particularly useful to improve the scalability of intelligent information systems based on constraint satisfaction techniques.

References

1. Genesereth, M. R., Keller, A. M., Duschka, O.: "Infomaster: An Information Integration System", Proceedings of 1997 ACM SIGMOD Conference, May 1997.
2. Marian Nodine, Jerry Fowler, Tomasz Ksiezyk, Brad Perry, Malcolm Taylor and Amy Unruh: Active Information Gathering in InfoSleuth In International Journal of Cooperative Information Systems 9:1/2, 2000, pp. 3-28.
3. Jerry Fowler, Brad Perry, Marian Nodine, and Bruce Bargmeyer: Agent-Based Semantic Interoperability in InfoSleuth SIGMOD Record 28:1, March, 1999, pp. 60-67.
4. Alon Y. Levy , Anand Rajaraman , Joann J. Ordille: "Querying Heterogeneous Information Sources Using Source Descriptions," *Proceedings of the 22nd VLDB Conference*, Bombay, India, 1996
5. Rita Cucchiara, Marco Gavanelli, Evelina Lamma, Paola Mello, Michela Milano, and Massimo Piccardi: "Constraint propagation and value acquisition: why we should do it interactively," *Proceedings of the 16th IJCAI*, Morgan Kaufmann, pp.468-477, 1999
6. R. Cucchiara, E. Lamma, P. Mello, M. Milano: "Interactive Constraint Satisfaction,", Technical Report DEIS-LIA-97-00, University of Bologna, 1997

7. Christian Bessière: "Arc-Consistency in Dynamic Constraint Satisfaction Problems," *Proceedings of the 9th National Conference of the AAAI*, pp. 221-226, 1991
8. Sanjay Mittal and Brian Falkenhainer: "Dynamic constraint satisfaction problems," *Proceedings of the 8th National Conference of the AAAI*, pp. 25-32, 1990
9. Makoto Yokoo: "Asynchronous Weak-commitment Search for Solving Large-Scale Distributed Constraint Satisfaction Problems," *Proceedings of the First International Conference on Multi–Agent Systems*, p. 467, MIT Press, 1995.
10. Peter M.D. Gray, Suzanne M. Embury, Kit Y. Hui, Graham J.L. Kemp: "The Evolving Role of Constraints in the Functional Data Model", *Journal of Intelligent Information Systems* **12**, pp. 113-137, 1999.
11. Monique Calisti, Boi Faltings, Santiago Macho-Gonzalez, Omar Belakhdar and Marc Torrens: "CCL: Expressions of Choice in Agent Communication," *Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, Boston MA, USA., July, 2000
12. A. Sheth and J.A. Larson: "Federated Database Systems," *ACM Computing Surveys* **22**(3), 1990
13. S. Chawathe, H. Garcia Molina, J. Hammer, K.Ireland, Y. Papakostantinou, J. Ullman and J. Widom: The TSIMMIS project: Integration of heterogeneous information sources. In *IPSJ Conference*, Tokyo, Japan, 1994
14. José Luis Ambite and Craig Knoblock: "Flexible and scalable cost-based query planning in mediators: A transformational approach," *Artificial Intelligence* **118**, pp. 115-161, 2000
15. Keith Decker, Katia Sycara and Mike Williamson: "Middle-Agents for the Internet," *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, Morgan Kaufmann, 1997, pp. 578-583
16. Sycara, K. "In-Context Information Management Through Adaptive Collaboration of Intelligent Agents." In Intelligent Information Agents: Cooperative, Rational and Adaptive Information Gathering on the Internet. Matthias Klusch (Ed.), Springer Verlag, 1999.
17. Katia Sycara, Seth Widoff, Matthias Klusch and Jianguo Lu: "LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace." Autonomous Agents and Multi-Agent Systems, 5, 173-203, 2002.
18. J. Budzik, S. Bradshaw, X. Fu, and K. Hammond: "Supporting Online Resource Discovery in the Context of Ongoing Tasks with Proactive Assistants," *International Journal of Human-Computer Studies* **56**(1) Jan 2002, pp. 47-74
19. B.J. Rhodes and P. Maes: "Just-in-time information retrieval agents," *IBM Systems Journal* **39**, pp. 685-704, 2000
20. F. Rossi, C. Petrie and V. Dhar: "On the equivalence of constraint satisfaction problems," *Proceedings of ECAI-90*, pp. 550-556, 1990
21. K. Stergiou and T. Walsh: "Encodings of Non-binary Constraint Satisfaction Problems," *Proceedings of AAAI-99*, ppp. 163-168, AAAI Press, 1999
22. N. Mamoulis and K. Stergiou: "Solving Non-binary CSPs Using the Hidden Variable Encoding," *Proccedings of CP 2001*, LNCS 2239, Springer-Verlag, pp. 168-182, 2001
23. P. Prosser: "Hybrid Algorithms for Constraint Satisfaction Problems," *Computational Intellligence* **9**(3), pp. 268-299, 1993
24. Eugene Freuder and Paul Hubbe: "Extracting Constraint Satisfaction Subproblems," *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 548-555, 1995
25. James C. Freanch and Allison L. Powell :Metrics for Evaluating Database Selection Techniques. 2000