Open Constraint Optimization

Boi Faltings and Santiago Macho-Gonzalez

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

Abstract. Constraint satisfaction has been applied with great success in closed-world scenarios, where all options 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 options and constraints must be gathered from different agents in a network. We define open constraint optimization as a model of such tasks.

Under the assumption that options are discovered in decreasing order of preference, it becomes possible to guarantee optimality even when domains and constraints are not completely known. We propose several algorithms for solving open constraint optimization problems by incrementally gathering options through the network. We report empirical results on their performance on random problems, and analyze how to achieve optimality with a minimal number of queries to the information sources.

1 Constraint Optimization in Distributed Systems

Constraint satisfaction and optimization has been applied with great success to resource allocation, scheduling, planning and configuration. Traditionally, these problems are solved in a closed-world setting: all variable domains and constraints are assumed to be completely known, then the problem is solved by a search algorithm.

With increasing use of the internet, many of the problems that constraint programming techniques are good at now pose themselves in a distributed setting. For example, in personnel allocation, it is possible to obtain staff from partner companies. In configuration, it is possible to locate part suppliers through the internet. Furthermore, problems may also involve agents with different and possibly conflicting interests, for example when allocating production resources among different factories.

Figure 1 illustrates the context we assume; a set of m agents wish to find an assignment to a set of variables that is optimal with respect to their preferences. A central CSP solver is tasked to find this solution, and queries the agents for their options and preferences using queries more (xi, di). Agents will return their options starting with the one they would most prefer as a solution, and then in
Fig. 1. Context of an open constraint optimization problem. The problem solver uses more messages to ask agents for additional values for a variable $x_i$ that extend its current domain $d_i$. Agents reply with option messages that indicate new values $v_j$ and their cost $r(v_j)$.

In a non-decreasing order of preference using option($x_i, v_j, r(v_j)$) messages, where $v_j$ is a new value and $r(v_j)$ is its cost, a numerical value that reflects the degree of undesirability.

Such a scenario would arise for example when optimizing scheduling and resource allocation in a consortium of enterprises, planning travel using options for transportation and lodging found through search on the internet, or when configuring a product from parts found through electronic catalogs in the internet.

A straightforward way of solving such problems is to ask agents to first report all their values, and then use standard techniques such as branch-and-bound to compute the optimal solution. However, for reasons of privacy, computation or communication effort, agents may not wish to reveal more of their options and preferences than necessary to compute a solution. In this case, it will be better to query for values only as they are required to prove optimality of the solution. In this paper, we propose the framework of open constraint optimization as a model of such problems along with general algorithms for solving it.

In comparison with classical constraint optimization, open constraint optimization poses three challenges:

- since the domains and constraints are not completely known, many standard CSP and optimization methods such as consistency or branch-and-bound cannot be directly applied to such a problem. Also, in general it is not clear how to give any optimality guarantee, since the best options may not have been discovered yet.

- besides minimizing computation time, the primary objective is now to minimize the number of values and costs that the agents have to provide. This is important because communication through a network is orders of magnitude slower than computation, because there may be a significant cost involved in getting the extra options, and because agents would prefer to keep this information private.
when agents are self-interested, there needs to be a mechanism to ensure that agents will report values and costs truthfully, since otherwise the optimization can not produce the correct result.

The first obstacle is that in an open environment, it is in general not possible to prove optimality of any solution, since options that are yet to be discovered may lead to a better solution. However, following an idea of Conen and Sandholm [9] for preference elicitation in auctions, optimality can be ensured if we assume that options are reported in decreasing order of preference. This is a key assumption that we will enforce as part of the definition of open constraint optimization problems. It is also very realistic in real-world settings: usually, agents will want to give their most preferred options first.

We present a model of open constraint optimization problems, show several sound and complete algorithms for solving them, and compare their performance. Provided that agents report information truthfully, our algorithms are sound and complete. We also show that they allow computing the required taxes for a Vickrey-Clark-Grove (VCG) tax mechanism that makes it in each agent's best interest to report information truthfully.

2 Related Work

The most closely related work is that on interactive constraint satisfaction ([1]) and open constraint satisfaction ([2]) which has addressed the issue of efficiently solving constraint satisfaction problems in open, distributed settings. In this paper, we are interested in constraint optimization problems, where the goal is not just to find a solution but also to prove that this solution has the minimal cost. This requires considerably different algorithms.

Open constraint optimization is related to dynamic constraint satisfaction, which allows constraints to be dynamically added and removed. Bessiere ([3]) 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 an open setting. Another major difference is that in open constraint optimization changes 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 ([4]) 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. The DisCSP formalism has also been extended to constraint optimization problems, with the most recent result being the ADOPT algorithm [5]. However, search algorithms for solving DisCSP rely on closed-world assumptions over variable domains for initiating backtracks and thus cannot be applied in an open-world context.
3 Assumptions and Notation

3.1 Constraint Optimization Problems

There exists a large variety of formalisms for modeling constraint optimization problems with soft constraints. In this paper, we assume that the problems are formulated as discrete weighted constraint optimization problems (WCOP). In a weighted constraint optimization problem, constraints assign each combination of assignments to its variables a cost, and the goal of the optimization is to find an assignment to all variables that minimizes the sum of the costs of all constraints:

**Definition 1.** A discrete weighted constraint optimization problem (WCOP) is a tuple $< X, D, C, R >$ where:

- $X = \{x_1, \ldots, x_n\}$ is a set of $n$ variables.
- $D = \{d_1, \ldots, d_n\}$ is a set of domains of the variables, each given as a finite set of possible values.
- $C = \{c_1, \ldots, c_m\}$ is a set of constraints, where a constraint $c_i$ is given as the list $(x_{i_1}, \ldots, x_{i_k})$ of variables it involves.
- $R = \{r_1, \ldots, r_m\}$ is a set of relations, where a relation $r_i$ is a function $d_{i_1} \times \ldots \times d_{i_k} \to \mathbb{R}^+$ giving the cost of choosing each combination of values.

A solution is a combination of values $v_1 \in d_1, \ldots, v_n \in d_n$ such that the sum of the cost of the relations is minimal.

Note that in a WCOP, any value combination is allowed, but might lead to a very high cost.

As an example, consider the scenario illustrated in Figure 2. A consortium of companies has to decide on three features $x_1, x_2$ and $x_3$ of a new product. It considers the options $x_1 \in \{A, B, C\}, x_2 \in \{A, C\}$ and $x_3 \in \{B, C\}$ (but this space may be extended later). The value chosen for each feature carries a certain cost to the consortium such that $A$ has a cost of $0$, $B$ a cost of $1$ and $C$ a cost of $2$. The features have interaction with parts provided by three members $S_1, S_2$ and $S_3$, where $S_1$ influences features $x_1$ and $x_2$, $S_2$ influences $x_1$ and $x_3$, and $S_3$ all three variables. The members make their proposals one variant at a time in strictly non-decreasing order of cost (as would be normal in a negotiation situation). In this example, we assume that they would eventually report the following costs:

$$S_1 \Rightarrow r_4 \Rightarrow x_4'; \quad S_2 \Rightarrow r_5 \Rightarrow x_5'; \quad S_3 \Rightarrow r_6 \Rightarrow x_6';$$

<table>
<thead>
<tr>
<th>$(x_1, x_2)$</th>
<th>Cost</th>
<th>$(x_1, x_3)$</th>
<th>Cost</th>
<th>$(x_1, x_2, x_3)$</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B,C)</td>
<td>0</td>
<td>(A,C)</td>
<td>0</td>
<td>(A,A,B)</td>
<td>0</td>
</tr>
<tr>
<td>(C)</td>
<td>0</td>
<td>(B,C)</td>
<td>0</td>
<td>(B,A,B)</td>
<td>0</td>
</tr>
<tr>
<td>(A,C)</td>
<td>1</td>
<td>(A,B)</td>
<td>1</td>
<td>(A,A,C)</td>
<td>2</td>
</tr>
<tr>
<td>(B,A)</td>
<td>3</td>
<td>(A,C)</td>
<td>3</td>
<td>(A,C,B)</td>
<td>2</td>
</tr>
<tr>
<td>(C,A)</td>
<td>3</td>
<td>(B,C)</td>
<td>3</td>
<td>(B,A,C)</td>
<td>2</td>
</tr>
<tr>
<td>(A,A)</td>
<td>5</td>
<td>(B,B)</td>
<td>5</td>
<td>(B,C,B)</td>
<td>2</td>
</tr>
</tbody>
</table>

The entire problem can thus be modelled as a WCOP as follows:
Fig. 2. Constraint optimization problem (top) and its hidden variable encoding (bottom).

\[ X = \{x_1, x_2, x_3\} \]
\[ D = \{(A, B, C), (A, C), (B, C)\} \]
\[ C = \{(x_1), (x_2), (x_3), (x_1, x_2), (x_1, x_3), (x_1, x_2, x_3)\} \]
\[ R = \{r_1, \ldots, r_6\} \]

Hidden Variable Encoding

The WCOP formulation is inconvenient for open settings since we need to distinguish relations and variable domains which may both be discovered incrementally, which considerably encumbers the algorithms. We therefore transform the problem into an equivalent dual one where the constraints are encoded as tuple-valued, “hidden” variables and all constraints are binary relations ensuring that compatible instantiations are chosen for these tuples. As shown in ([6]), this transformation is possible for any constraint satisfaction problem. When we add to this representation the original variables and also link them to the tuples, we obtain the hidden variable encoding ([7]) of the problem.

As observed in [8], the hidden variable encoding generalizes in a straightforward way to weighted constraint optimization problems:

Definition 2. A hidden-variable weighted constraint optimization problem (HW-COP) is a tuple \(<X, D, C, R>\) obtained from a WCOP \(<X', D', C', R'>\) in the following way:

- \(X = \{x_1, \ldots, x_m\}\) corresponds to the constraints \(C'\). We assume that for each variable, there is a unary constraint for each variable giving the costs of its values so that this set also includes all variables of the original problem.
- $D = \{d_1, ..., d_m\}$ are the domains of the variables in $X$. They correspond to the domains of the relations of the original WCOP, where we assume that each domain is ordered so that the options with lowest cost come first.
- $C = \{(x_i, x_j, t) | x_i \in x_j\}$ is a set of hard constraints where $x_i$ is a single variable, $x_j$ is a tuple-valued variable containing $x_i$, and the type $t$ indicates that the constraint is a $c_t, t \in 1..k$ that enforces equality between the individual variable and the $t$-th element of the tuple.
- $R = R'$ are the cost functions as in the original WCOP.

Figure 2 shows how the example WCOP is transformed into an HWCOP by introducing new variables, $x_4$, $x_5$, and $x_6$, to model the constraints of the three agents. These are linked to the original variables by constraints of type $c_1$, $c_2$ and $c_3$, enforcing equality between the variable and the corresponding element of the tuple. The solutions to the transformed HWCOP are identical to those of the original WCOP as far as the shared variables $x_1$ to $x_3$ are concerned.

We further define:

**Definition 3.** An assignment of an HWCOP is an assignment of a value to each variable and represented as a vector $A = (i_1, ..., i_n)$ giving the position of each value in the respective domain. To simplify notation, we denote by $c(v(x_i)) = r_i(v(x_i))$ the cost of the assignment $x_i = v$. The cost of the assignment is the sum of the costs for each variable, i.e. cost($A) = \sum_{j=1}^{n} c(d_j(i_j(A)))$.

**Definition 4.** An assignment of a HWCOP is optimal if and only if it is consistent with the constraints and there is no other assignment that has a lower cost. An optimal assignment is a solution to the HWCOP.

We denote by $v^*_x(x_i) (x_k)$ the value assigned to $x_i$ in the solution that is optimal for the subproblem $x_1, ..., x_k$ (of which $x_i$ is part); when there are several solutions we choose the lexicographically smallest one.

### 3.3 Open HWCOP

As stated earlier, we are interested in optimization problems where options are obtained from agents dispersed through a network. We model this as an open HWCOP (OHWCOP), where each variable is contributed by an agent and models either a set of options that that agent contributes, or a constraint/preference that the agent wants to impose. Each agent applies the hidden variable encoding locally to the constraints/preferences it has on the joint problem, and communicates to the central solver its variables as well as the constraints that need to be enforced between them and other variables of the combined problem. The constraint solver then solves the problem and finally communicates the solution for each variable to the agents.

We require agents to follow the following rules of the protocol, where violators will be excluded from the optimization:

- agents always return their best values first.
- for each variable, costs are normalized so that the cost of the best tuple is always 0.
and formalize the problem posed to the central solver as:

**Definition 5.** An open hidden-variable weighted constraint optimization problem (OHWCOP) is a possibly unbounded, partially ordered set \( \{P^0, P^1, \ldots\} \) of HWCP, where \( P^i \) is defined by a tuple \( <X, D^i, C, R^i> \). The set is ordered by the relation \( \prec \) where \( P^i \prec P^j \) if and only if \( (\forall k \in [1..m]) d^i_k \subseteq d^j_k \), and \( (\exists k \in [1..m]) d^i_k \subset d^j_k \), i.e. all domains of \( P^j \) are at least as large as for \( P^i \) and at least one is larger.

Costs are assumed to be stable, i.e. if for some value \( v \in d^i_k \), \( r^i_k(v) = c \) then \( r^j_k(v) = c \) for all \( j \) such that \( P^i \prec P^j \).

As agents return their best options first, domains are ordered in non-increasing order of cost, i.e. \( r^i_k(d^i_k(t-1)) \leq r^i_k(d^i_k(t)) \) for all \( t \leq |d^i_k| \).

A solution to an OHWCOP is an assignment that is a solution to some instance HWCP and optimal for all higher instances, i.e. \( P^j \) such that \( P^i \prec P^j \).

The assumption of non-increasing cost is necessary for any form of open constraint optimization:

**Proposition 1.** When the assumption that domains are returned in order of non-decreasing cost does not hold, there is no general algorithm for solving OHWCOP that is guaranteed to terminate with the optimal solution without querying the entire domains of all variables.

**Proof.** Consider a problem that has a consistent solution with cost 0 involving the values returned last and other consistent assignments with cost > 0 involving other values. Here the optimal solution can only be found after querying all values. Since a general algorithm cannot know whether such a case is present, for any problem that does not have a solution with cost=0 it will have to check all domains completely to guarantee optimality.

4 Solving open constraint optimization problems

A first approach to solving OHWCOP would be to use standard branch-and-bound search algorithms coupled with a test for whether sufficient values are known to guarantee optimality also with larger domains.

The following proposition shows how far domains have to be known in order to guarantee optimality of the solution. We call an instance that satisfies this proposition subset domain-sufficient:

**Proposition 2.** The optimal solution to an instance HWCP(i) of an OHWCOP is guaranteed to be optimal for the OHWCOP itself if for any subset of variables \( S = x_{s1}, \ldots, x_{sk} \) and \( T = X - S \), we have that:

\[
\max_{v \in \mathbb{D}_S} c(v) \geq \sum_{x_i \in X} c(v^*_X(x_i)) - \sum_{x_i \in X - S} c(v^*_{X-S}(x_i))
\]

and \( v^*_{X-S} \) is optimal for the subproblem HWCP involving only variables in \( X - S \).
Proof. Let there be a better solution in an instance \(COP(j)\), and let \(S\) be the set of variables that have values not in \(COP(i)\). Then all the variables \(x_{sd} \in S\) must have a cost greater than \(\max_{v \in D_{sd}} c(v)\) for otherwise they would have been in \(COP(i)\). But then the cost of this solution would have to be at least:

\[
\sum_{x_{sd} \in S} \max_{v \in D_{sd}} c(v) + \sum_{x_i \in X-S} c(v^*_X(x_i)) \geq \sum_{x_i \in X} c(v^*_X(x_i))
\]

so it could not be optimal. Thus, the algorithm achieves optimality in \(COP(i)\).

This condition can be used to prove optimality of a solution without knowing the complete domains of all variables by recursively applying it to show optimality of \(v^*_X\) for the remaining sets \(X - S\) until this set becomes empty. However, it does not provide an operational criterion for constructing an algorithm that makes a minimal number of queries, since the sequence of sets \(S\) that would allow a proof with a minimal number of values cannot be computed without knowing the domains completely. In fact, we can show:

**Proposition 3.** There is no general algorithm for solving \(OHW\) that is guaranteed to solve all instances with a minimal number of queries.

Proof. Consider a problem with 3 variables \(x_1, x_2, x_3\) with identical domains \(\{a, b\}\) and constraints that require all variables to have equal values. Assume further that \(x_1\) has costs \(a/0, b/2\), \(x_2: a/1, b/0\) and \(x_3: a/0, b/1\). Then the optimal solution \(a\) can be found by with querying only the first value for \(x_3\) when variables are taken in the order \(x_1, x_2, x_3\), but requires querying all values when variables are taken in order \(x_3, x_2, x_1\). Since an algorithm cannot distinguish \(x_1\) and \(x_3\) before querying the second value, it could not always choose the optimal order.

It is however possible to show the following, more operational condition, which we call **singleton domain-sufficiency**:

**Proposition 4.** The optimal solution to an instance \(HW\) of an \(OHW\) is guaranteed to be optimal for the \(OHW\) itself if the domain of each variable \(x_k\) is known up to at least one value whose cost is not smaller than \(\overline{c_k}\), where \(\overline{c_k}\) is given as:

\[
\overline{c_k} = \max_{X' \subseteq X, x_k \in X'} \left( \sum_{x_i \in X'} c(v^*_X(x_i)) - \sum_{x_i \in X' \setminus x_k} c(v^*_X \setminus x_k(x_i)) \right)
\]

Proof. Omitted for lack of space.

This condition, slightly stronger than subset domain-sufficiency, can be used to build an algorithm that incrementally queries values only for variables that are not singleton domain-sufficient. However, as the condition has to be checked for each subset of variables, such an algorithm is very inefficient, and for lack of space we don’t give it in detail in this paper. We did however implement it and use it as a benchmark later in the paper.
4.1 Basic incremental search algorithm

Another approach is to use best-first algorithms based on A* that incrementally generate all possible assignments. While this approach can use a lot of memory, it provides a good basis for addressing the problems posed by open domains.

Function $\text{o-opt(OHWCOP)}$

For all $x_i, d_i \leftarrow \text{more}(d_i)$

$OPEN \leftarrow \{(1, ..., 1)\}$

loop

$M \leftarrow \{a \in OPEN | \text{cost}(a) = \min_{d \in OPEN} \text{cost}(d)\}$

$a \leftarrow \text{lexicographically smallest element of } M$

$OPEN \leftarrow OPEN - \{a\}$

if $\text{consistent}(a)$ then

return $a$

else

for $j = 1..n$ do

$b \leftarrow (a(1), ..., a(j) + 1, ..., a(n))$

if $b \notin OPEN$ then

$OPEN \leftarrow OPEN \cup \{b\}$

if $|d_j| < b(j)$ then

$d_j \leftarrow \text{append}(d_j, \text{more}(d_j))$

Algorithm 1: $\text{o-opt: an incremental algorithm for solving OHWCOP}$.

Algorithm 1 is inspired by the preference elicitation algorithm of Conen and Sandholm ([9]) for combinatorial auctions and systematically enumerates all possible assignments in the order of increasing weight. Whenever necessary, it moves to a higher HWGCP by extending the domains of one or more variables. The first assignment that is consistent with all constraints is an optimal solution, since all assignments of lower weights have already been searched and found inconsistent, as in the A* algorithm. We can show the following:

**Theorem 1.** Algorithm 1 is sound and complete.

**Proof.** Soundness is guaranteed by the fact that the algorithm only returns consistent assignments and systematically explores all assignments in strictly non-decreasing order of cost, so that the one returned is also the one with the lowest cost. Completeness is guaranteed by the fact that the algorithm systematically enumerates all assignments.

4.2 Failure-driven search

The solution found by Algorithm 1 on our example is the assignment:

$x_1 = A, x_2 = A, x_3 = B$
with a total cost of 7. It is interesting to consider what values are queried by the algorithm in order to reach this conclusion. In fact, to establish optimality of this solution, the algorithm will query all values for all variables. And the problem can get much worse: consider a problem with 100 variables where the optimal solution is one where each variable has a cost of 1. The algorithm would, however, query for values up to a cost of 100 for each variable, since a combination of one assignment with a cost of 100 with 99 assignments of cost 0 would have optimal cost. The problem here is that the algorithm generates numerous candidates where the cost of subproblems is far lower than in the optimal solution.

Function \text{f-o-opt}(OHWCOP) 
For all \( x_i, d_i \leftarrow \text{more}(d_i) \)
\text{OPEN} \leftarrow \{(1,\ldots,1)\}
\text{loop}
M \leftarrow \{a \in \text{OPEN}|\text{cost}(a) = \min_{d \in \text{OPEN}}\text{cost}(d)\}
\text{a} \leftarrow \text{lexicographically smallest element of } M
\text{OPEN} \leftarrow \text{OPEN} \cup \{a\}
\text{if consistent}(a) \text{ then}
\quad \text{return } a
\text{else}
\quad \text{c} \leftarrow \text{first violated constraint in } a, \text{i.e. the violated constraint involving an } x_k \text{ with the smallest } k.
\quad \text{for } j \in \text{vars}(c) \text{ do}
\quad \quad b \leftarrow (a(1),\ldots,a(j)+1,\ldots,a(n))
\quad \quad \text{if } b \not\in \text{OPEN} \text{ then}
\quad \quad \quad \text{if } |d_j| < b(j) \text{ then}
\quad \quad \quad \quad d_j \leftarrow \text{append}(d_j, \text{more}(d_j))
\quad \quad \quad \text{OPEN} \leftarrow \text{OPEN} \cup \{b\}
\quad \text{end loop}
\text{Algorithm 2: f-o-opt: an failure-driven algorithm for solving OHWCOP.}

Algorithm 2 significantly improves on this based on the observation that it is not necessary to generate all successors to an assignment \( a \):

\textbf{Proposition 5.} Let \( a = (a(1),\ldots,a(n)) \) be an inconsistent assignment and let the pair of variables \( x_i \) and \( x_j \) such that \( d_i(a(i)) \) and \( d_j(a(j)) \) are not consistent with a constraint \( c \) between them. Now consider the direct successors \( b_k = (a(1),\ldots,a(k)+1,\ldots,a(n)) \). All direct successors except \( b_i \) and \( b_j \) are redundant and can be pruned in Algorithm 1 without affecting its soundness and completeness.

\textbf{Proof.} When \( k \neq i, j \) we have that:

\begin{itemize}
\item \( b_k \) is inconsistent, as it contains the same conflict with \( c \) as \( a \).
\item all direct or indirect successors to \( b_k \) that do not change the values for \( x_i \) or \( x_j \) are inconsistent, as they still contain the same conflict with \( c \) as \( b_k \).
\end{itemize}
- all direct (indirect) successors to $b_k$ that change the value for $x_i$ or $x_j$ are also a direct (indirect) successor of either $b_i$ or $b_j$.

where we use the term indirect successor for sequences of direct successor relationships. Thus, all $b_k, k \neq i, j$ cannot be themselves solutions or lead to solutions that would not be generated from $b_i$ and $b_j$ already.

**Theorem 2.** Algorithm 2 is sound and complete.

**Proof.** Follows from Theorem 1 and Proposition 5.

On the example, Algorithm 2 finds the same solution as Algorithm 1, but queries significantly less values. For example, for variable $x_0$ it only queries three rather than all 12 values.

Thus, in Algorithm 2, instead of generating all successors to an assignment, we only generate those successors that are involved in the first conflicting constraint. This turns out to have the effect that the algorithm only explores assignments to $x_1, \ldots, x_{k+1}$ once it has found a consistent assignment to $x_1, \ldots, x_k$. This behavior is very similar to that of Russian Doll Search (RDS) ([10]), an algorithm that generates optimal solutions to increasingly large subproblems and uses their costs as bounds for solving larger problems.

A potential problem with Algorithm 2 is that it keeps a complete list $OPEN$ of all currently best nodes, so that memory consumption can become a problem. If necessary, the $OPEN$ list can be incrementally regenerated by a depth-first search process such as IDA* ([11]), limiting memory consumption at the expense of additional computation time. It is straightforward to obtain the same behavior by querying additional values for the variables involved in the first conflict of the candidate tightest to the current cost limit.

Below we consider what guarantees can be given for Algorithm 2 regarding the number of values queried.

### 4.3 Optimality of the number of value queries

Proposition 2 shows that any algorithm that guarantees optimality of the solution to an OHWCOP must query at least a certain number of values. This bound can be compared to the queries that Algorithm 2 actually makes. For this, we have the following propositions:

**Proposition 6.** Algorithm 2 queries at most one value for $x_k$ with cost $\geq c$ until it has examined all assignments to $x_1, \ldots, x_n$ with total cost $< c \sum_{j=1}^{k-1} \sum_{a(x_j) \neq 0} c(x_1, \ldots, x_{k-1}, a(x_j))$.

**Proof.** Queries for $x_k$ can only result from generating a successor to an assignment $a$ that is consistent up to $x_{k-1}$, for otherwise, there would be an earlier violated constraint. Now assume that the algorithm had already queried a value for $x_k$ with cost $\geq c$, then since assignment $a$ must have $x_k$ assigned to the highest cost value so far, the cost of assignment $a$ must thus be at least the optimum for $x_1, \ldots, x_{k-1}$ plus $c$. For this assignment to reach the head of the OPEN queue, all assignments with lower cost must have already been examined,
Proposition 6 gives us the following bound on the queries per variable:

**Proposition 7.** For variable $x_k$, Algorithm 2 queries values up to a cost threshold that differs from the optimum $\overline{c}_k$ by at most:

$$\sum_{x_i \in X \setminus x_k} c(v^*_{X \setminus x_k}(x_i)) - \sum_{j=1}^{k-1} c(v^*_{x_1 \ldots x_{k-1}, x_j})$$

Thus, in particular, for $x_n$ it queries the minimally possible number of values.

**Proof.** When the algorithm terminates, it has examined all assignments with a cost less than that of the solution, i.e., $\sum_{j=1}^{n} c(v^*_X(x_j))$. By Proposition 6, it will thus have queried at most one value for $x_k$ that is beyond the limit:

$$\max_k = \sum_{j=1}^{n} c(v^*_X(x_j)) - \sum_{j=1}^{k-1} c(v^*_{x_1 \ldots x_{k-1}, x_j})$$

Subtracting the limit given in Proposition 2 for $S = \{x_k\}$ results in the given relation. Its instantiation for the last variable $x_n$ gives a difference of 0.

In practice, the empirical results in Figure 3 show that Algorithm 2 is actually very close to optimal.

### 4.4 Achieving incentive-compatibility

Optimization is useless unless the costs are reported truthfully. As already proposed in [12], using the Vickrey-Clark-Groves (VCG) tax mechanism we can make the mechanism *incentive-compatible*, i.e., make it optimal for all agents to tell the truth. In the VCG tax mechanism, each agent pays a tax equal to the difference between the cost of the solution to the optimization problem when it is present and the cost when it is not. More formally, let agent $A$ have imposed the constraints modelled by the set of variables $X_A$, then the tax, called the Clark tax, is:

$$\text{payment}(A) = \sum_{x_k \in X \setminus X_A} c(v^*_X(x_k)) - c(v^*_{X \setminus X_A}(x_k))$$

In the example of Figure 2, the optimal solution is $(x_1, x_2, x_3) = (A, A, B)$ with a cost of 6. Without the constraint $x_4$, it is still $(A, A, B)$, without $x_5$, $(B, A, B)$ and without $x_6$, $(A, C, B)$. Using the notation $C_X(.)$ to denote the cost of a solution to agent $X$, the tax paid by the agents would be:

<table>
<thead>
<tr>
<th>Payment</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \cdot c_B(ABA) + c_C(ABA) - c_B(ABA) - c_C(ABA)$</td>
<td>$1-1 = 0$</td>
</tr>
<tr>
<td>$B \cdot c_A(ABA) + c_C(ABA) - c_A(BAB) - c_C(BAB)$</td>
<td>$5-3 = 2$</td>
</tr>
<tr>
<td>$C \cdot c_A(ABA) + c_B(ABA) - c_A(ACB) - c_B(ACB)$</td>
<td>$6-2 = 4$</td>
</tr>
</tbody>
</table>
Note that these taxes reflect the cost of reaching agreement on the variable values; the agents would usually receive other payments for their contributions to the consortium elsewhere.

The following argument shows why this tax makes it optimal for an agent to tell the truth:

- suppose that an agent overstates its costs. In all cases where this influences the solution, this will result in a tax that outweighs the benefit of the manipulated solution.
- suppose that an agent understates its costs. In all cases where this influences the solution, the loss incurred by not having its optimal solution outweighs the savings in tax.

It is important to note that the domains and costs required to compute these payments are the same as those required by Proposition 2. In fact, any algorithm for open constraint optimization that can guarantee optimality is also guaranteed to query all required values and costs for computing the VCG tax payments.

5 Empirical Results

![Graph](image)

**Fig. 3.** Comparison of Algorithms 1 (A*) and 2 (foopt) with respect to the average number of values queried in relation to average domain size. Also shown for comparison is a minimal algorithm based on Proposition 4.

Figure 3 compares the number of values queried for the two algorithms on random OHWCOP problems, where the constraint graph was generated randomly and constraints were assigned a random valuation from a uniform distribution between 0 and 1. We can see that both Algorithms 1 and 2 come
close to the benchmark performance, with Algorithm 2 being somewhat better. In order to get an idea of what the minimal number of queries is, we also implemented a method that incrementally finds optimal solutions to larger and larger subproblems. It maintains the property of singleton domain-sufficiency as in Proposition 4 and can be shown to query a minimal number of values to maintain this property. It can be seen that Algorithm 2 is almost optimal with respect to that method as well.

In comparison to this performance, the classical method of gathering all values and then solving the problem would query all domain values and can become arbitrarily worse than these algorithms, depending on the looseness of the problem.

Figure 4 compares the number of constraint checks and shows that only Algorithm 2 can be applied to problems of realistic size; in practice it can handle problems of up to 25 variables before constraint checks start to become excessive.

![Graph showing the comparison of Algorithms 1 (A*) and 2 (foopt) with respect to the number of constraint checks in relation to the number of problem variables. Also shown for comparison is a minimal algorithm based on Proposition 4.](image)

**Fig. 4.** Comparison of Algorithms 1 (A*) and 2 (foopt) with respect to the number of constraint checks in relation to the number of problem variables. Also shown for comparison is a minimal algorithm based on Proposition 4.

6 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 presented an extension of the constraint optimization framework that allows solving such systems in an open framework while still guaranteeing
optimality of the overall solution. This shows that it is possible to solve optimization problems even without the closed-world assumption, and opens up new possibilities for intelligent systems in open environments.

However, it turns out to be difficult to adapt the depth-first branch-and-bound methods commonly used for constraint optimization to this open framework, as the conditions that are needed to guarantee that they find the optimal solutions are very complex to evaluate. We have shown instead how it is possible to obtain good results by adapting a framework based on best-first search algorithms. By exploiting the local nature of failure in constraint satisfaction problems, we have shown how it is possible to greatly improve the behavior of such an algorithm. However, the size of problems that can be solved efficiently is still far from what can be handled when the closed-world assumption can be made, and we hope to stimulate further research for better algorithms that work in open settings.

Acknowledgements
We like to thank Thomas Sandholm for interesting discussions and insights.

References
8. Xavier Larrous, Rina Dechter: "On The Dual Representation of Non-Binary Semiring-Based CSPs” Workshop 1 (Soft Constraints) of CP 2000, September, 2000