Constraint-Based Preferences via Utility Hyper-Graphs

Rafik Hadfi and Takayuki Ito
Department of Computer Science and Engineering
Graduate School of Engineering, Nagoya Institute of Technology
Gokiso, Showa-ku, Nagoya 466-8555, Japan
rafik@itolab.nitech.ac.jp, ito.takayuki@nitech.ac.jp

Abstract
Real-world decisions involve preferences that are nonlinear and often defined over multiple and interdependent issues. Such scenarios are known to be challenging, especially in strategic encounters between agents having distinct constraints and preferences. In this case, reaching an agreement becomes more difficult as the search space and the complexity of the problem grow.

In this paper, we propose a new representation for constraint-based utility spaces that can tackle the scalability problem by efficiently finding the optimal contracts. Particularly, the constraint-based utility space is mapped into an issue-constraint hyper-graph. Exploring the utility space reduces then to a message passing mechanism along the hyper-edges by means of utility propagation. We experimentally evaluate the model using parameterized random nonlinear utility spaces. We show that it can handle a large family of complex utility spaces by finding the optimal contract(s), outperforming previous sampling-based approaches.

1 Introduction
Complex decision making problems involve multiple and interdependent issues, which usually yields nonlinear utility spaces. Finding the contract(s) that maximize a given preference profile becomes then challenging as the search cost increases with the increasing complexity of the utility space.

In this paper, we propose a new, modular representation for constraint-based utility spaces that can efficiently tackle the complexity of the search process. We address this problem by adopting a representation that allows a modular decomposition of the issues-constraints given the intuition that constraint-based utility spaces are nonlinear with respect to issues, but linear with respect to the constraints. This allows us to rigorously map the utility space into an issue-constraint hyper-graph. Exploring the utility space reduces then to a message passing mechanism along the hyper-edges by means of utility propagation.

Adopting a graphical representation while quantitatively reasoning about preferences is not new in the multi-attribute utility literature. In fact, the idea of utility graphs could potentially help decomposing highly nonlinear utility functions into sub-utilities of clusters of inter-related items, as shown in (Chajewska and Koller 2000; Bacchus and Grove 1995). Similarly, (Robu, Somefun, and Poutre 2005) used utility graphs for preferences elicitation and negotiation over binary-valued issues.

In another work, (Marsa-Maestre et al. 2009) adopts a weighted undirected graph representation of the constraint-based utility space. Particularly, a message passing algorithm is used to find the highest utility bids by finding the set of unconnected nodes which maximizes the sum of the nodes’ weight. However, restricting the graph and the message passing process to constraints’ nodes does not allow the representation to be descriptive enough to exploit any potential hierarchical structure of the utility space through a quantitative evaluation of the interdependencies between both issues and constraints. In (Fujita, Ito, and Klein 2010), issues’ interdependency is captured by means of similar undirected weighted graphs where a node represents an issue. This representation is restricted to binary interdependencies while real-world scenarios involve “bundles” of interdependent issues under one or more constraints. In our approach, we do not restrict the interdependency to lower-order constraints but we allow p-ary interdependencies to be defined as an hyper-edge connecting p issues. Adopting such representation with its underlying utility propagation mechanism comes from the intuition that preferences elicitation, when operating on a graphical model, could be seen as some form of Bayesian inference (Kwisthout and van Rooij 2013). Therefore, we chose to adopt a graphical representation that serves more as an adequate framework for any preference-based space.

The advantage of using this representation is its scalability in the sense that the problem becomes harder for a large number of issues and constraints. However, if we can decompose the utility space, we can exploit it more efficiently. That is, the decomposition will isolate independent components, thus, allowing them to be treated independently from the rest of the problem. This could allow a layered, hierarchical view of any given decision making problem. Given such architecture, it is possible to reason over the different layers of the problem according to a top-down approach. Even the idea of issue could be generalized to include sub-issues, located in sub-utility spaces and represented by cliques in the hyper-graph. Consequently, the search process could identify sub-contracts that are subject to improvement at each
The utility function (1) is nonlinear in the sense that the utility does not have a linear expression against the contract (Ito, Hattori, and Klein 2007). This is true to the extent that the linearity is evaluated with regard to the contract $\vec{c}$. However, from the same expression (1) we can say that the utility is in fact linear, but in terms of the constraints $\{c_1, \ldots, c_j, \ldots, c_m\}$. The utility space is therefore decomposable according to these constraints. This yields a modular representation of the interactions between the issues and how they locally relate to each other. In fact, $\text{hyp}(c_j)$ reflects the idea that the underlying contracts are governed by the bounds defined by $c_j$ once the contracts are projected according to their issues’ components. In this case, the interdependence is not between issues but between constraints. For instance, two constraints $c_1$ and $c_2$ can have in common one issue $i_k$ taking values respectively from an interval $I_{k,c_1}$ if it is involved in $c_1$, and values in $I_{k,c_2}$ if it is involved in $c_2$, with $I_{k,c_1} \neq I_{k,c_2}$. Finding the value that maximizes the utility of $i_k$ while satisfying both constraints becomes harder due to fact that changing the value of $i_k$ in $c_1$ changes the instance of $i_k$ in $c_2$ in a cyclic manner. This nonlinearity gets worse with an increasing number of issues, domains’ sizes, and the non-monotonicity of the constraints.

Next, we propose to transform (1) into a modular, graphical representation. Since one constraint can involve one or more multiple issues, we will adopt a hyper-graph representation.

2.3 From Utility Space To Utility Hypergraph

We assign to each constraint $c_j$ a factor $\Phi_j$, yielding a factors’ set $\Phi = \{\Phi_1, \ldots, \Phi_j, \ldots, \Phi_m\}$. The utility hypergraph $G$ is defined as $G = (\mathbb{I}, \Phi)$. Nodes in $\mathbb{I}$ define the issues and the hyper-edges in $\Phi$ are the factors (constraints). The neighbors’ set $N(\Phi_j) \subset \mathbb{I}$ of factor $\Phi_j$ represents the issues connected to $\Phi_j$ (involved in constraint $c_j$), with $|N(\Phi_j)| = \varphi_j$. In case $\varphi_j = 2 \forall j$ the problem collapses to a constraints satisfaction problem in a standard graph.

To each factor $\Phi_j$ corresponds a $\varphi_j$-dimensional matrix, $\mathcal{M}_{\Phi_j}$, where the kth dimension is $I_k = [a_k, b_k]$, the domain of issue $i_k$. This matrix contains all the values that could be taken by the issues in $N(\Phi_j)$. Each factor $\Phi_j$ has a function $u_j$ defined as a sub-utility function of the issues in $N(\Phi_j)$, defined as in (2).

$$u_j : N(\Phi_j)^{\varphi_j} \rightarrow \mathbb{R}$$

$$u_j(\vec{x}) \mapsto w(c_j, \vec{x}), \vec{x} = (i_1, \ldots, i_j, \ldots, i_{\varphi_j})$$

As we are dealing with discrete issues, $u_j$ is a mapping defined by the matrix $\mathcal{M}_{\Phi_j}$. That is, $u_j(i_1, \ldots, i_k, \ldots, i_{\varphi_j})$.
is simply the \((1, \ldots, k, \ldots, \varphi_j)^{th}\) entry in \(M_{\Phi_j}\), corresponding to the value \(w(c_j, \vec{x})\) described in (2). It is possible to extend the discrete case to the continuous one by allowing continuous issue values and defining \(u_c\) as a continuous function. Next, we provide few examples about the representation.

**Example 1.** We illustrate the mapping between a 2-dimensional utility space (Figure 2a) and its corresponding hyper-graph \(G_2\) (Figure 2b). The issues’ domains are \(I_1 = I_2 = [0, 9]\). \(G_2\) consists of \(m = 10\) constraints (red squares) where each constraint involves at least 2 issues (white circles). We note 6 cubic constraints, 3 plane constraints and 1 bell constraint.

![Utility space](image1)

![Utility hyper-graph](image2)

**Figure 2:** 2-dimensional utility space and its hyper-graph

**Example 2.** Now, consider a 10-dimensional utility space mapped into the hyper-graph \(G_{10} = (I, \Phi)\) with \(I = \{i_1, \ldots, i_9\}\) and \(\Phi = \{\Phi_1, \ldots, \Phi_7\}\), shown in Figure 3.

Each issue \(i_k\) has a set \(I_k = \bigcup_{\nu \in N(i_k)} I_{k, \nu}\) where \(I_{k, \nu}\) is an edge connecting \(i_k\) to its neighbor \(\nu \in N(i_k) \in \Phi\). For example, \(I_1 = \bigcup_{\nu \in \{\Phi_1, \Phi_3, \Phi_4\}} I_{1, \nu} = \{[5, 9], [3, 4], [3, 6]\}\). The constraints are cubic \((\Phi_1, \Phi_2, \Phi_3, \Phi_4)\), plane \((\Phi_5, \Phi_6)\) and bell \((\Phi_7)\). Each constraint \(c_j\) (resp. factor \(\Phi_j\)) is assigned a sub-utility function \(u_{\nu}\) to find the utility of a contract if it satisfies \(c_j\) by being located in \(hyp(c_j)\). Depending on its type, each constraint’s sub-utility is defined as in (3).

\[
\begin{align*}
  u_j &= \begin{cases} 
  \beta_j + \sum_{k=1}^{\varphi_j} \alpha_{j, k} i_k & \text{if Plane} \\
  v_j & \text{if Cube} \\
  \alpha_{j, k} i_k & \text{if Bell}
  \end{cases}, \\
  \beta_j + \sum_{k=1}^{\varphi_j} \alpha_{j, k} i_k & \text{if Plane} \\
  v_j & \text{if Cube} \\
  \alpha_{j, k} i_k & \text{if Bell}
\end{align*}
\]

That is, the plane constraint \(\Phi_j\) is defined using its \(\varphi_j\)-dimensional equation, while a cubic constraint is assigned a value \(v_j\). The sub-utility \(V_j\) of a bell shaped constraints is defined in (4). Herein, \(\delta\) is the Euclidean distance from the center of the bell constraint to the contract point. Distances are normalized in \([-1, 1]\).

\[
V_j = \begin{cases} 
  \beta_j (1 - 2\delta^2) & \text{if } \delta < 1/2 \quad \beta_j \in \mathbb{Z} \\
  2\beta_j (1 - \delta)^2 & \text{if } \delta < 1 \quad \beta_j \in \mathbb{Z} \\
  0 & \text{else}
  \end{cases}
\]

**3 Optimal Contracts**

The exploration of the utility hyper-graph is inspired from the sum-product message passing algorithm for loopy belief propagation (Pearl 1988). However, the multiplicative algebra is changed into an additive algebra to support utility accumulation necessary for the assessment of the contracts. The messages circulating in the hyper-graph are nothing other than the contracts we are attempting to optimize through utility maximization. Next, we develop the message passing (MP) mechanism.

**3.1 Message Passing**

We consider the issues set \(I\) and a contract point \(\vec{c} = (i_1, \ldots, i_k, \ldots, i_n) \in I\). We want to find a contract \(c^*\) that maximizes the utility function (1). Assuming that \(u_{\nu}\) is the local sub-utility of constraint \(\Phi_{j}\), we distinguish two types of messages: messages sent from issues to constraints, and messages sent from constraints to issues.

**From issue \(i_k\) to constraint \(\Phi_j\):** In (5), each message \(\mu_{i_k \rightarrow \Phi_j}\) coming from \(i_k\) to \(\Phi_j\) is the sum of the constraints’ messages to \(i_k\) coming from constraints other than \(\Phi_j\).

\[
\mu_{i_k \rightarrow \Phi_j} (i_k) = \sum_{\Phi_{j'} \in N(i_k) \setminus \Phi_j} \mu_{\Phi_j \rightarrow i_k} (i_k)
\]

**From constraint \(\Phi_j\) to issue \(i_k\):** Each constraint message is the sum of the messages coming from issues other than \(i_k\), plus the sub-utility \(u_{\nu} (i_1, \ldots, i_k, \ldots, i_n)\), summed over all the possible values of the issues (connected to the constraint \(\Phi_j\)) other than the issue \(i_k\).

\[
\mu_{\Phi_j \rightarrow i_k} (i_k) = \max_{i_1} \cdots \max_{i_{k'}} \cdots \max_{i_n} \left[ u_j (i_1, \ldots, i_k, \ldots, i_n) \right.
\]

\[
+ \sum_{i_{k'} \in N(\Phi_j) \setminus i_k} \mu_{i_{k'} \rightarrow \Phi_j} (i_k) \right]
\]

The MP mechanism starts from the leaves of the hyper-graph, i.e. the issues. At \(t = 0\), the content of the initial messages is defined according to (7), with \(u_{ij} (i_k)\) being the partial evaluation of \(i_k\) in the factor \(\Phi_j\).

\[
\mu_{i_k \rightarrow \Phi_j} (i_k) = 0
\]

\[
\mu_{\Phi_j \rightarrow i_k} (i_k) = u_{ij} (i_k)
\]

At any time, the optimal contract \(c^*\) is found by collecting the optimal issue-values as in (8a).

\[
c^* = (i_1^*, \ldots, i_k^*, \ldots, i_n^*)
\]

\[
i_k^* = \arg \max_{i_k} \sum_{\Phi_{j} \in N(i_k)} \mu_{\Phi_{j} \rightarrow i_k} (i_k)
\]

In a strategic encounter between agents, it is more common that an agent requires a collection, or bundle, of the optimal contracts rather than one single optimum. In order to find such collection, we should endow (8b) with a caching mechanism allowing each node in the hyper-graph to store the messages that have been sent to it from the other nodes. That is, the cached messages will contain the summed-up utility values of the underlying node’s instance. This is performed every time \(max\) is called in (6) so that we can store the settings of the adjacent utility (and contract) that led to the maximum. Once ordered, such data structure could, for instance, be used in a bidding process. In the next section, we algorithmically provide the MP mechanism.
3.2 Utility propagation algorithm

Main algorithm. Algorithm 1 operates on the hypergraph nodes by triggering the MP process. Despite the fact that we have two types of nodes (issues and constraints), it is possible to treat them abstractly using graph nodes by triggering the MP process. Despite the fact that a result is a collection of optimal contracts with utility greater or equal to the agent’s reservation value \( rv \).

Algorithm 1: Main Algorithm

```
Algorithm: Utility Propagation
Input: \( G = (\Phi, RV, mode, \rho) \)
Output: Optimal contracts (bundle)
begin
  for \( i = 1 \rightarrow (\rho \times |\Phi|) \) do
    if \( mode \ is \ synchronous \) then
      foreach \( \nu_{src,i} \in \Phi \) do
        foreach \( \nu_{dest,i} \in \nu_{src,i}.Neighbors() \) do
          \( bundle[i] \leftarrow \emptyset \)
          foreach \( j \in \nu_{dest,i}.Neighbors() \) do
            \( i \leftarrow \nu_{dest,i}.Neighbors()[j] \)
            \( \mu^* \leftarrow i.getmax() \)
            if \( \mu^* < \mu[k] \) then
              \( \mu \leftarrow \mu[k] \)
            if \( \mu^* \geq rv \) then
              \( bundle[i] \leftarrow bundle[i] \cup i[k^*] \)
    return \( bundle \)
end
```

Issue to Constraint. The issue’s message to a factor (or constraint) is the element-wise sum of all the incoming messages from other factors, as shown in Algorithm 2.

Constraint to Issue. In Algorithm 3, the factor’s message to a targeted issue is done by recursively enumerating over all variables that the factor references (6), except the targeted issue. This needs to be performed for each value of the target variable in order to compute the message. If all issues are assigned (\( \hat{x}_i : \alpha[i] = -1 \)), the values of the factor and of all other incoming messages are determined, so that their summation is compared to the prior maximum, as in Algorithm 4. The resulting messages, stored in \( bundle \), contain the values that maximize the factors’ local utility functions.

Optimal issue-values. At any time of the utility propagation process, it is possible to collect the current optimal contract(s) by individual concatenation of all the optimal issue-values \( I_{k,i} \), defined in (8b). Particularly, the summation in (8b) is performed as to only include the overlapping evaluations depending on how the issue domains are defined for different factors. For instance, Figure 4 shows how issue \( i_k \) has three possible evaluations depending on \( I_{k,1}, I_{k,j} \) and \( I_{k,m} \).

The maximization objective (8b) will attempt to find the
performed backwardly through random generation of values attempting to have an tract(s) search. The SA optimizer will be randomly sampling approach in (Ito, Hattori, and Klein 2007) for optimal con-

We start with the generation of the hyper-graph using Algorithm 4. Sum: recursive summing

combination(s) of $v_{1,i}$, $v_{k,i}$ and $v_{m,i}$ that maximize the sum. An optimal combination is an optimal issue-value $i_k^*$. Evaluations of issue $i_k$ $i_k \in \text{hyp}(c_j)$

Figure 4: Finding the optimal issue-values

4 Experiments

4.1 Settings

Before evaluating the utility propagation algorithm, we identify the criteria that could affect the complexity of the utility space and thus the probability of finding optimal contract(s). Other than $n$ and $m$, we distinguish $p$, defined as the maximal number of issues involved in a constraint. $p$ can be unary ($p = 1$), binary ($p = 2$), ternary ($p = 3$), or $p$-ary in the general case. The parametrized generation of a utility space (or utility hyper-graph) must meet the consistency condition $p < n \leq m \times p$, with $n, m, p \in \mathbb{N}^+$, to avoid problems like attempting to have an $8$-ary constraints in a $5$-dimensional utility space.

from $I_k, j \forall i_k \in \mathcal{N}(\Phi_j)$. Our comparison criteria is based on the utility/duration performed on a set of profiles of the form $(n, m, p)$, with 100 trials for each profile. The first version of the message passing process, SynchMP, is synchronous. That is, all the combinations of issue evaluations are generated deterministically. Figure 5 illustrates the performance of SynchMP for $(10, [10, 20, 30], 5)$.

Indeed, the deterministic aspect of SynchMP makes it very slow ($\Delta_{SA} < \Delta_{SynchMP}$) compared to its SA counterpart which exploits the randomization, allowing it to perform “jumps” in the search space. To avoid the enumeration over local nodes of $G$, it is possible to add randomization to the way nodes are selected. To introduce an asynchronous mode, AsynchMP, we add a new mode after the synchronous mode condition in Algorithm 1, as follows:

if mode is Asynchronous then

\[ \nu_{\text{src}}, \nu_{\text{dest}} \leftarrow \text{rand}_2([1, |V|]), \nu_{\text{dest}} \neq \nu_{\text{src}} \]

For $(40, [20, \ldots, 100], 5)$, Figure 6 shows the resulting difference in the performance of AsynchMP compared to SA.

5 Conclusion

We proposed a new representation of constraint-based utility spaces based on hyper-graphs that allows a modular decomposition of the constraints and issues. The exploration and search for optimal contracts is performed based on a message passing mechanism outperforming the sampling based optimizers.

As future work, we intend to exploit the structure of the hyper-graphs to induce a hierarchical type of automated bi-
lateral negotiation. Additionally, we think about studying the issues’ importance and how they might affect the final outcome of an automated negotiation.

References


