A BDD-Based Polytime Algorithm for Cost-Bounded Interactive Configuration

Tarik Hadzic and **Henrik Reif Andersen**
Computational Logic and Algorithms Group
IT University of Copenhagen, Denmark
{tarik,hra}@itu.dk

Abstract

Interactive configurators are decision support systems assisting users in selecting values for parameters that respect given constraints. The underlying knowledge can be conveniently formulated as a Constraint Satisfaction Problem where the constraints are propositional formulas. The problem of interactive configuration was originally inspired by the *product configuration* problem with the emergence of the mass-customization paradigm in product manufacturing, but has also been applied to other tasks requiring user interaction, such as specifying services or setting up complex equipment. The user-friendly requirements of *complete*, *backtrack-free* and *real-time* interaction makes the problem computationally challenging. Therefore, it is beneficial to compile the configuration constraints into a tractable representation such as Binary Decision Diagrams (BDD) (Bryant 1986) to support efficient user interaction. The compilation deals with the NP-hardness such that the online interaction is in polynomial time in the size of the BDD.

In this paper we address the problem of extending configurators so that a user can interactively limit configuration choices based on a maximum cost (such as price or weight of a product) of any valid configuration, in a complete, backtrack-free and real-time manner. The current BDD compilation approach is not adequate for this purpose, since adding the total cost information to the constraints description can dramatically increase the size of the compiled BDD. We show how to extend this compilation approach to solve the problem while keeping the polynomial time guarantees.

Introduction

Interactive configuration problems are special applications of Constraint Satisfaction Problems (CSP) where a user is assisted in interactively assigning values to variables by a software tool. The key user functionality delivered by this tool is the *calculation of valid domains* (*CVD*) for each unassigned variable. Application areas include customizing physical products (such as PC's and cars) and services (such as airplane tickets and insurances).

The domains calculated by the *CVD* functionality should be *complete* (all valid configurations should be reachable through user interaction), *backtrack-free* (a user is never forced to change an earlier choice due to incompleteness in the logical deductions), and the *CVD* feedback should be in *real-time*. The requirement for backtrack-freeness and completeness makes the problem of calculating valid domains NP-hard. Therefore, in order to provide real-time guarantees, current approaches use off-line compilation of the CSP model into a tractable datastructure representing the solution space of all valid configurations (Møller, Andersen, & Hulgaard 2004; Amilhastre, Fargier, & Marquis 2002; Madsen 2003). Although having exponential worst-case size, in the real industrial applications that are using BDDs, the compiled representations can often be kept small (Hadzic *et al.* 2004; Subbarayan *et al.* 2004).

In this paper we look at the problem of *cost-bounded interactive configuration*, where every selectable value is assigned a cost. The extended *CVD* functionality, denoted as *max-bounded CVD*, should allow a user to limit the maximum cost for any final valid configuration. This problem corresponds closely to an important product-configuration functionality where a user can interactively limit the maximum cost of any configurable specification, such as price or weight of the configured product.

We first describe *CVD* functionality for the standard BDD-based configuration. Then we show that the immediate approach to cost bounded configuration can lead to exponential explosion in the size of the BDD representation. We then construct algorithms that augment the BDD representation to deliver max-bounded *CVD* with polynomial-time guarantees. Finally, we show that it is not possible to extend polynomial-time guarantees for cost-bounded configuration where domains are restricted by both minimum and maximum bounds.

The remainder of the paper is organized as follows. In the next section we formalize the concept of interactive configuration, afterwards we describe the BDD-based configuration approach. The main section describes a solution approach to cost-bounded configuration. Before we conclude we discuss the complexity of some extensions to the cost-bounded approach.

Interactive Configuration

The input *model* describing the knowledge about valid variable assignments is a special kind of a *Constraint Satisfaction Problem* (CSP) (Dechter 2003):

A *configuration model* C is a triple (X, D, F) where X is

a set of variables $\{x_0, \ldots, x_{n-1}\}$, $D = D_0 \times \ldots \times D_{n-1}$ is the Cartesian product of their finite domains $D_0, \ldots, D_{n-1}$ and $F = \{f_0, ..., f_{m-1}\}$ is a set of propositional formulae over atomic propositions $x_i = v$, where $v \in D_i$, specifying conditions on the values of the variables.

Concretely, every domain can be viewed as $D_i = \{0, \ldots, |D_i| - 1\}$. An *assignment* of values $v_0, \ldots, v_{n-1}$ to variables $x_0, \ldots, x_{n-1}$ is denoted as a set of pairs $\rho = \{(x_0, v_0), \ldots, (x_{n-1}, v_{n-1})\}$. The domain of the assignment $dom(\rho)$ is the set of variables which are assigned: $dom(\rho) = \{x_i \mid \exists v \in D_i.(x_i, v) \in \rho\}$ and if $dom(\rho) = X$ we refer to ρ as a *total assignment*. We say that a total assignment ρ is *valid* if it satisfies all the rules, which is denoted as $\rho \models F$. A partial assignment $\rho, dom(\rho) \subseteq X$ is *valid* if there is at least one total assignment $\rho' \supseteq \rho$ that is valid, $\rho' \models F$, i.e. if there is at least one way to successfully complete the existing configuration process.

In product configuration, the knowledge about product components and product rules is usually modelled by representing all the choices for a component as values in a variable domain. Then, a valid total assignment ρ completely specifies a configurable product.

User Interaction

A configurator assists a user interactively in reaching a valid product specification, i.e. in reaching a valid total assignment. The key operation in this interaction is that of computing, for each unassigned variable $x_i \in X \setminus dom(\rho)$, the *valid domain* $D_i^\rho \subseteq D_i$. The domain is *valid* if it contains those and only those values with which ρ can be extended to become a total valid assignment ρ', i.e. $D_i^\rho = \{v \in D_i \mid \exists \rho' : \rho' \models F \wedge \rho \cup \{(x_i, v)\} \subseteq \rho'\}$. The significance of this demand is that it guarantees the user backtrack-free assignments to variables as long as he selects values from valid domains. This reduces cognitive effort during the interaction and increases usability.

At each step of the interaction, the configurator reports the valid domains to the user, based on the current partial assignment ρ resulting from his earlier choices. The user then picks an unassigned variable $x_i \in X \setminus dom(\rho)$ and selects a value from the calculated valid domain $v_i \in D_i^\rho$. The partial assignment is then extended to $\rho \cup \{(x_i, v_i)\}$ and a new interaction step is initiated.

BDD Based Configuration

In (Møller, Andersen, & Hulgaard 2004; Hadzic *et al.* 2004; Subbarayan *et al.* 2004) the interactive configuration was delivered by dividing the computational effort into an *offline* and *online* phase. First, in the offline phase, the authors compiled a BDD (Bryant 1986) representing the solution space of all valid configurations $Sol = \{\rho \mid \rho \models F\}$. Then, the functionality of *calculating valid domains* (*CVD*) was delivered online, by efficient algorithms executing during the interaction with a user. The benefit of this approach is that the BDD needs to be compiled only once, and can be reused for multiple user sessions.

An important requirement for online user interaction is the guaranteed real-time experience of the user configurator interaction. Therefore, the algorithms that are executing in the online phase must be provably efficient in the size of the BDD representation. This is what we call the *real-time guarantee*. As the *CVD* functionality is NP-hard, and the online algorithms are polynomial in the size of the generated BDD, there is no hope of providing polynomial size guarantees for the worst-case BDD representation. However, it suffices that the BDD size is small enough for all the configuration instances occurring in practice (Subbarayan *et al.* 2004).

Compiling the Configuration Model

To compile a configuration model into a BDD, each of the finite domain variables x_i with domain $D_i = \{0, \ldots, |D_i| - 1\}$ is encoded by $k_i = \lceil log|D_i| \rceil$ Boolean variables $x_0^i, \ldots, x_{k_i-1}^i$, referred to as *log encoding variables*. Each $v \in D_i$, corresponds to a binary encoding $v_0 \ldots v_{k_i-1}$ denoted as $v_0 \ldots v_{k_i-1} = enc(v)$. Also, every combination of bits $v_0 \ldots v_{k_i-1}$ represents some integer $v \leq 2^{k_i} - 1$, denoted as $v = dec(v_0 \ldots v_{k_i-1})$ such that $dec(enc(v)) = v$. Hence, the atomic proposition $x_i = v$ is encoded as a Boolean expression $x_0^i = v_0 \wedge \ldots \wedge x_{k_i-1}^i = v_{k_i-1}$. In addition, *domain constraints* are added to forbid those bit assignments $v_0 \ldots v_{k_i-1}$ which do not translate to a value in D_i, i.e. where $dec(v_0 \ldots v_{k_i-1}) \geq |D_i|$.

Assuming the ordering $x_0 < \ldots < x_{n-1}$, a solution space Sol is represented by a Reduced Ordered Binary Decision Diagram $B(V, E, X_b, R, var)$, where V is a set of nodes ranged over by u, including the terminal nodes T_0, T_1. E is the set of edges ranged over by e and $X_b = \{0, 1, \ldots, |X_b| - 1\}$ is set of variable indices, labelling every non-terminal node u with $var(u) \leq |X_b| - 1$ and labelling the terminal nodes T_0, T_1 with index $|X_b|$. The set of variable indices X_b is constructed by taking the union of the log-encoding variables $\bigcup_{i=0}^{n-1} \{x_0^i, \ldots, x_{k_i-1}^i\}$ and ordering them in a natural layered way, i.e. $x_{j_1}^{i_1} < x_{j_2}^{i_2}$ iff $i_1 < i_2$ or $i_1 = i_2$ and $j_1 < j_2$.

Every directed edge $e = (u_1, u_2)$ has a starting vertex $u_1 = \pi_1(e)$ and ending vertex $u_2 = \pi_2(e)$. R denotes the root node of the BDD.

Standard CVD Algorithms

In this section we will briefly revisit how the calculation of valid domains is executed in the standard configuration case. The description is based on the Clab (Jensen online) configuration framework, and a more detailed formalization can be found in (Hadzic, Jensen, & Andersen 2006 online).

Before describing the algorithms, we first introduce the appropriate notation. Let x_j^i be the $j+1$-st Boolean variable encoding the finite domain variable x_i. Then, if x_j^i corresponds to an index $k \in X_b$ we define $var_1(k) = i$ and $var_2(k) = j$ to be the appropriate mappings. Now, given the BDD $B(V, E, X_b, R, var)$, V_i denotes the set of all nodes $u \in V$ that are labelled with a BDD variable encoding the finite domain variable x_i, i.e. $V_i = \{u \in V \mid var_1(u) = i\}$. We think of V_i as defining a layer in the BDD. We define In_i to be the set of nodes $u \in V_i$ reachable by an edge originating from outside the V_i layer, i.e. $In_i = \{u \in$

$V_i| \ \exists(u', u) \in E.\ var_1(u') < i\}$. For the root node R, labelled with $i_0 = var_1(R)$ we define $In_{i_0} = V_{i_0} = \{R\}$.

Now we will more precisely denote the interaction process in which a user extends the partial assignment by assigning values from valid domains to unassigned variables. We assume that we are in an interaction step where in the previous user assignment, a user fixed a value for a finite domain variable $x = v, x \in X$, extending the old partial assignment ρ_{old} to the current assignment $\rho = \rho_{old} \cup \{(x_i, v)\}$. For every variable $x_i \in X$, old valid domains are denoted as $D_i^{\rho_{old}}, i = 0, \ldots, n-1$, and the old BDD $B^{\rho_{old}}$ is reduced to a restricted BDD, $B^{\rho}(V, E, X_b, R, var)$. Note that for assigned variables $x_i \in dom(\rho_{old})$, $D_i^{\rho_{old}} = \{\rho(x_i)\}$ and the log-encoding variables X_b correspond only to variables $X \setminus dom(\rho_{old})$.

The *CVD* functionality is to calculate valid domains D_i^{ρ} for the remaining unassigned variables $x_i \notin dom(\rho)$ by extracting values from the newly restricted BDD $B^{\rho}(V, E, X_b, R, var)$. To simplify the following discussion, we will analyze the isolated execution of the *CVD* algorithm over a given BDD $B(V, E, X_b, R, var)$. The task is to calculate valid domains VD_i from the starting domains D_i. The user-configurator interaction can be modelled as a sequence of these executions over restricted BDDs B^{ρ}, where the valid domains are D_i^{ρ} and the starting domains are $D_i^{\rho_{old}}$.

A value $v \in D_i$ is in VD_i if there is a path in B from the root to the terminal T_1 consistent with the binary encoding of v. The *CVD* functionality is implemented by observing two key ideas on when we can conclude if there is such a path. First, this happens if there is a *long edge* $e = (u_1, u_2)$ crossing over layer V_i, i.e. $var_1(u_1) < i < var_1(u_2)$ such that $u_2 \neq T_0$. Then we can include all the values from D_i into a valid domain, i.e. $VD_i = D_i$. Detecting and copying of all "skipped" variable domains can be achieved in $O(|E| + n)$ where $|E|$ is the number of edges in the compiled BDD B and $n = |X|$ is the number of finite domain variables. For more details check (Hadzic, Jensen, & Andersen 2006 online).

For the remaining variables x_i, whose layer V_i is not crossed by any long edge, we check for each value v in a domain D_i whether it can be part of the domain VD_i. The second key observation is that if $v \in VD_i$ then there must be $u \in V_i$ such that traversing the BDD from u with log-encoding of v will lead to a node other than T_0. This is sufficient since from any non-zero node $u \in V \setminus \{T_0\}$ it is possible to reach the terminal T_1, i.e. possible to satisfy the BDD B.

Traversing $u \in V_i$ with value $j \in D_i$ is depicted by the algorithm in Fig. 1. It is important to note that the marking scheme for each $j \in D_i$ prevents processing the same node more then once. Hence, to calculate VD_i it takes at most $O(|V_i| \cdot |D_i|)$ steps. Now, the total worst-case running time to implement the classical *CVD* functionality is $O((\sum_{i=0}^{n-1} |V_i| \cdot |D_i|) + |E| + n) = O(\sum_{i=0}^{n-1} |V_i| \cdot |D_i| + n)$.

```
Traverse(u, v)
1:   i ← var_1(u)
2:   v_0, ..., v_{k_i - 1} ←  enc(v)
3:   repeat
4:    if Marked[u] = v return T_0
5:    Marked[u] ← v
6:    s ← var_2(u)
7:    if v_s = 0 then u ← low(u)
8:    else u ← high(u)
9:   until var_1(u) ≠ i
10:  return u
```

Figure 1: For fixed $u \in V, i = var_1(u)$, $Traverse(u, v)$ iterates through V_i and returns the node in which the traversal ends up according to the value v. $Marked[u]$ is initialized to -1.

Interactive Maximum Cost Bounded Configuration

We now consider an extension of the configuration model where the selection of each choice $v \in D_i$ is associated with a cost.

A *cost bounded configuration model* C_c is a quadruple (X, D, F, c) where $C(X, D, F)$ is a standard configuration model and c is a cost function such that $c_v^i \in \mathbf{Z}$ denotes the integer cost of choice $x_i = v$, $x_i \in X$, $v \in D_i$.

The cost of assignment ρ is defined as $c(\rho) = \sum_i c^i_{\rho(x_i)}$. Given the starting domains D_i, partial user assignment ρ and a user-designated maximum cost C_{max}, the configurator should calculate and display the valid domains involving only those choices that can be extended to a configuration of maximum price C_{max}. The cost-bounded valid domains are: $D_i^{\rho, C_{max}} = \{v \in D_i \mid \exists \rho'.(\rho' \models F \ and \ \rho \cup \{(x_i, v)\} \subseteq \rho' \ and \ c(\rho') \leq C_{max})\}$. In the following subsections we will discuss how to implement this functionality.

Encoding Costs in the BDD

To simplify the following discussion we assume that the costs c_v^i are non-negative integers. An immediate approach to deliver max-bounded *CVD* is to create a new configuration model C_y by introducing a new variable $y \in D_y$ to the configuration model C and adding appropriate propositional constraints enforcing for any solution ρ

$$y = \sum_i c^i_{\rho(x_i)}. \tag{1}$$

We then generate a new BDD representation B^y and let the user manipulate the cost in the standard configuration process. The user can for example specify the exact price of the final product $y = v$ and in return get valid options D_i for other unselected components x_i. A constraint $y \leq C_{max}$ can be encoded as $y = 0 \vee y = 1 \vee \ldots \vee y = C_{max}$.

We claim that this approach is inadequate since the size of B^y can be exponential in the model encoding C_y even for the practical instances where B is of manageable size. This would make efficient *CVD* algorithms unusable since they operate over the exponentially large structure.

Let $x_0 < \ldots < x_{i-1} < y < x_{i+1} < \ldots < x_n$ be a variable ordering where the variable y is in the i-th place ($x_i = y$). Let $C^{Sol}(X')$ denote the set of all different costs over all valid combinations of variables in $X' \subseteq X$, i.e. $C^{Sol}(X') = \{\sum_{x_i \in X'} c^i_{\rho(x_i)} \mid \rho \in Sol\}$. Let V^y denote the set of vertices in BDD B^y. We have the following:

Theorem 1.
$|V^y| \geq$
$max(|C^{Sol}(\{x_0, \ldots, x_{i-1}\})|, |C^{Sol}(\{x_{i+1}, \ldots, x_n\})|)$ [1]

Proof. It suffices to notice the following: For every cost of a valid partial assignment $c(\rho) \in C^{Sol}(\{x_0, \ldots, x_{i-1}\})$ there must be at least one node $u \in In_y$. Namely, if there were more costs than nodes then there is a node $u' \in In_y$ for which there are at least two partial assignments ρ_1, ρ_2 of different costs $c(\rho_1) \neq c(\rho_2)$ such that traversing from R through both ρ_1 and ρ_2 ends up in u'. This means that for any valid partial assignment ρ_3 traversing from u' to T_1 we have the same total price $\rho_3(y)$ (from equation (1)) equal to two different numbers $c(\rho_1) + \sum_{k=i+1}^{n} c^k_{\rho_3(x_k)}$ and $c(\rho_2) + \sum_{k=i+1}^{n} c^k_{\rho_3(x_k)}$, which is not possible.

With similar reasoning, we conclude that for cost of a valid partial assignment $c(\rho) \in C^{Sol}(\{x_{i+1}, \ldots, x_n\})$ there must be at least one node in $Out_y = \{u_2 \in V \mid \exists (u_1, u_2) \in E.(var(u_1) = i \; and \; var(u_2) > i)\}$. Hence, $|V^y| \geq max(|In_y|, |Out_y|) \geq max(|C^{Sol}(\{x_0, \ldots, x_{i-1}\})|, |C^{Sol}(\{x_{i+1}, \ldots, x_n\})|)$ □

Now, it is easy to construct instances where B_y explodes while B is small. For example, if all the variables are Boolean except y, and each $c^i_1 = 2^i$ and $c^i_0 = 0$, and we have no constraints $F = \{\}$ (leading to always true BDD), we have $|B| = 1$ but $|B^y| > 2^{\lfloor n/2 \rfloor}$.

An Extended BDD Compilation Approach

The basic idea behind our approach is not to introduce the price variable y into the model, but to process a user restriction $c(\rho) \leq C_{max}$ by additional filtering during *online* execution of *CVD* algorithms. By doing this, we do not risk an exponential blow-up of the BDD representation. We can therefore construct the *CVD* algorithms without sacrificing the processing guarantee.

In order to calculate valid domains VD_i, we need to be able to determine for a value $v \in D_i$ with designated cost c^i_v whether it is possible to find an assignment $\rho \models F$, such that $\rho(x_i) = v$ and $c(\rho) \leq C_{max}$. In other words, we need to check if there is an assignment $\rho = \rho_1 \cup \{(x_i, v)\} \cup \rho_2$ such that $c(\rho_1) + c^i_v + c(\rho_2) \leq C_{max}$, $dom(\rho_1) = \{x_0, \ldots, x_{i-1}\}$, $dom(\rho_2) = \{x_{i+1}, \ldots, x_{n-1}\}$.

To illustrate the key idea, assume that a path σ in layer V_i (referred to as *i-path* σ), is encoding a value $v \in D_i$, i.e. $\sigma = \{(u_1, u_2), \ldots, (u_{k-1}, u_k)\}$ such that $u_k = Traverse(u_1, v)$ and $var_1(u_k) = i + 1$. The cost of path σ is $c(\sigma) = c^i_v$. Let $\rho^{u_1}_{up}$ be the cheapest assignment to variables $\{x_0, \ldots, x_{i-1}\}$ that when traversed from root R ends up in u_1. Also, let $\rho^{u_k}_{down}$ be the cheapest assignment to variables $\{x_{i+1}, \ldots, x_{n-1}\}$ that traverses from u_k and ends up in T_1. Let us denote $c(\rho^{u_1}_{up})$ as $U[u_1]$ and $c(\rho^{u_k}_{down})$ as $D[u_k]$. Now to test whether path σ allows value v to be added to VD_i it suffices to check whether $U[u_1] + c(\sigma) + D[u_k] \leq C_{max}$. This is illustrated in Fig. 2.

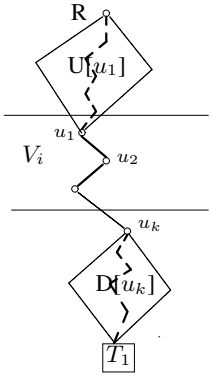

Figure 2: An i-path $\sigma = \{(u_1, u_2), \ldots, (u_{k-1}, u_k)\}$ encoding the value $v \in D_i$. $U[u]$ and $D[u]$ denote the costs of cheapest paths from u to R and T_1 respectively. If the cost $U[u_1] + c_v(\sigma) + D[u_k] \leq C_{max}$ then v is in the valid domain.

The key idea is to *efficiently calculate* the values $U[u_1]$ and $D[u_k]$ for every i-path $\sigma = \{(u_1, u_2), \ldots, (u_{k-1}, u_k)\}$. Note that this requires $u_1 \in In_i$ and that the last edge $(u_{k-1}, u_k) \in E$ that is crossing out of the layer V_i can be a long edge such that for $i_2 = var_1(u_2)$ it holds $i_2 > i + 1$. Also, a path σ could encode more than one value $v \in D_i$ since some of the internal edges (u_j, u_{j+1}) might be skipping variables, allowing for more then one value. The cost of σ encoding the value v is denoted as $c_v(\sigma) = c^i_v + \sum_{l=i+1}^{i_2-1} c^l_{min}$ where c^l_{min} is the cheapest value in D_l, i.e. $c^l_{min} = min_{j=0}^{k_l-1}\{c^l_j\}$. Now, for every i-path $\sigma = \{(u_1, u_2), \ldots, (u_{k-1}, u_k)\}$ encoding value $v \in D_i$, it holds:

$$U[u_1] + c_v(\sigma) + D[u_k] \leq C_{max} \Rightarrow v \in D_i.$$

Augmented Labelled Graph In order to efficiently compute labels $U[u]$, $D[u]$ we introduce an *Augmented Labelled Graph* $G'(V', E', w)$. It is a directed acyclic graph constructed from BDD $B(V, E, X_b, R, var)$ that allows efficient computation of these labels by running the efficient single-source shortest-path algorithms.

The key idea is to add *augmented edges*, $e' = (u_1, u_2)$ between any two nodes $u_1 \in In_i$, $var_1(u_2) > i$ when there exists an i-path σ between u_1 and u_2 encoding a value $v \in D_i$. In other words, $E'_i = \{(u_1, u_2) \mid u_1 \in In_i \; and \; \exists v \in D_i.u_2 = Traverse(u_1, v)\}$ and $E' = \cup_{i=0}^{n-1} E'_i$. Let $w_{min}(u_1, u_2)$ denote the cost of the cheapest assignment c^i_v among all $v \in D_i$ such that $u_2 = Traverse(u_1, v)$.

[1] From Theorem 1 we see that unless many values in the domains share identical costs, the BDD B^y is likely to explode. In fact, if any of $|C^{Sol}(\{x_0, \ldots, x_{i-1}\})|$ or $|C^{Sol}(\{x_{i+1}, \ldots, x_n\})|$ is exponentially large, it is guaranteed that $|B_y|$ will be exponentially large.

The cost of the entire edge is $w(u_1, u_2) = w_{min}(u_1, u_2) + \sum_{k=i+1}^{i_2-1} c^i_{min}$. Actually, $w(u_1, u_2)$ is equal to the price of the cheapest i-path σ between u_1 and u_2. The construction algorithm is shown in Fig. 3.

```
ConstructGraph(B)
1:  E' ← {}, V' ← {R}
2:  for i = 0 to n − 1
3:    for each v ∈ D_i
4:      for each u_1 ∈ In_i
5:        u_2 ← Traverse(u_1, v)
6:        if (u_1, u_2) ∉ E' then w(u_1, u_2) ← ∞
7:        E' ← E' ∪ {(u_1, u_2)}, V' ← V' ∪ {u_2}
8:        i_2 ← var_1(u_2)
9:        t ← c^i_v + Σ_{k=i+1}^{i_2−1} c^k_min
10:       if w(u_1, u_2) > t then
11:         w(u_1, u_2) ← t, w_min(u_1, u_2) ← c^i_v
12:   return (V', E', w)
```

Figure 3: Adds augmented edges (u_1, u_2) to the graph and labels them with the cost of the cheapest path from u_1 to u_2.

The running time is $O(\sum_{i=0}^{n-1} |V_i| \cdot |D_i|)$ where the marking scheme in $Traverse(u_1, v)$ ensures that for every $v \in D_i$ at most $|V_i|$ vertices are traversed. The induced graph $G'(V', E')$ has at most $\sum_{i=0}^{n-1} |In_i| \cdot |D_i|$ edges, since we add at most $|D_i|$ edges for every node $u \in In_i$. Since we do not introduce any new nodes, and possibly delete some internal nodes it holds: $|V'| \leq |V|$.

Augmented CVD Algorithms

In this section we show how to calculate new domains $VD_0, \ldots, VD_{n-1}$ that also satisfy the maximum cost restriction.

The first step is to label the nodes $u \in V'$ of the augmented graph G' with $U[u]$, $D[u]$. It suffices to run two single-source shortest path algorithms, from root R and terminal T_1 each. The resulting distances are exactly the values for our U and D labels. The shortest-path algorithms are well described in details in (Cormen *et al.* 2001). Since G' is a directed acyclic graph the running time is $O(|E'|+|V'|)$.

Then, in order to calculate valid domains of variables skipped by a long edge, we execute the algorithm in Fig. 4 which calculates for every variable $x_i \in X$ the cost of the cheapest path skipping over layer V_i. In other words, the algorithm calculates the price of the cheapest path involving an edge $(u_1, u_2) \in E'$ such that $var_1(u_1) < i < var_1(u_2)$. The price of such minimal cost path is stored in $P[i]$.

Topological sorting takes $O(|E'| + |V'|)$ time. For every long edge (u, u') we execute line 4, $var_1(u') - var_1(u) - 1$ times. Hence, if we denote $X((u, u')) = \{var_1(u) + 1, \ldots, var_1(u') - 1\}$, then the worst-case execution time is $O(\sum_{e \in E'} |X(e)|) + O(|E'| + |V|) = O(|E'| \cdot n) = O((\sum_{i=0}^{n-1} |In_i| \cdot |D_i|) \cdot n) = O((\sum_{i=0}^{n-1} |V_i| \cdot |D_i|) \cdot n)$.

After calculating cheapest skipping paths, the algorithm in Fig. 5 efficiently extracts valid domains with respect to interactively provided maximum-cost limit C_{max}.

```
CheapestPaths(G')
1:  for all i = 0, ..., n − 1, P[k] ← ∞,
2:  for all (u, u') ∈ E' in increasing
      topological order, u' ≠ T_0
3:    for l = var_1(u) + 1 to var_1(u') − 1
4:      P[l] ← min{P[l], U[u]+w(u, u')+D[u']}
```

Figure 4: For every variable x_k the algorithm calculates $P[k]$, the cheapest path in the augmented graph spanning across the V_k, i.e. allowing for any assignment to x_k. Obviously, such a path might not exist, in which case $P[k] = \infty$.

```
CVDRuntime(B, C_max)
1:  for each i = 0 to n − 1
2:    max ← C_max + c^i_min − P[i]
3:    VD_i ← {v ∈ D_i | c^i_v < max}
4:    for each v ∈ D_i \ VD_i
5:      for each u_1 ∈ In_i
6:        u_2 ← Traverse(u_1, v)
7:        if u_2 ≠ T_0 and U[u_1] + w(u_1, u_2)−
            w_min(u_1, u_2) + c^i_v + D[u_2] ≤ C_max
8:        then
9:          VD_i ← VD_i ∪ {v}, skip to next v
```

Figure 5: The algorithm extracts valid domains from BDD B, based on maximum cost C_{max}. It uses all the auxiliary labels $U[u]$, $D[u]$, $P[i]$, $w(u_1, u_2)$, $w_{min}(u_1, u_2)$ that were precalculated in previous steps.

If $P[i] - c^i_{min} + c^i_v \leq C_{max}$ then the cheapest path $P[i]$ crossing the layer V_i allows to include the cost of value $v \in D_i$ instead of the minimal cost c^i_{min}. The maximum bound for such c^i_v that can be included is calculated in line 2, and then all the values cheaper then this maximum bound are added to VD_i in line 3. Note that if there is no cheapest path then $P[i] = \infty$ and no values are added. For each i this step executes in $O(|D_i|)$ time.

For the rest of the values $v \in V_i \setminus VD_i$ it is checked whether there is an induced edge (u_1, u_2) that is compatible with v such that the price of traversal $c^i_v + \sum_{k=var_1(u_1)}^{var_1(u_2)-1} c^k_{min} = w(u_1, u_2) - w_{min}(u_1, u_2) + c^i_v$ is smaller than $C_{max} - U[u_1] - D[u_2]$. If this test succeeds (line 7) then v is part of the valid domain. For each i, traversals take $O(|V_i| \cdot |D_i|)$ time.

Hence, the algorithm in Fig. 5 executes in worst case $O(\sum_{i=0}^{n-1} |D_i| + \sum_{i=0}^{n-1} (|V_i| \cdot |D_i|))$ time.

Finally, the entire user interaction step is illustrated in Fig. 6. The complexity is dominated by the $Cheapest - Paths(G')$ algorithm in line 6, which executes in $O(|E'| \cdot n)$ time.

Extensions and Infeasibility Results

In a similar way, we can efficiently implement the user functionality of restricting the *minimal cost* of valid solutions, i.e. given the user restriction C_{min} and partial assignment ρ we can in polynomial time w.r.t. to compiled representation B calculate the valid domains $D_i^{\rho, C_{min}} = \{v \in$

$UserInteraction(B, C_{max}, \rho)$

```
1:  if user assigns x_i = v
2:    ρ ← ρ ∪ {(x_i, v)}
3:    B ← B^ρ
4:    G' ← ConstructGraph(B)
5:    Compute U[u], D[u]
6:    CheapestPaths(G')
7:    CVDRuntime(B, C_max)
8:    Display all VD_j
9:  else if user changes C_max
10:   CVDRuntime(B, C_max)
11:   Display all VD_j
```

Figure 6: An interaction step in the user configuration process.

$D_i \mid \exists \rho'.(\rho' \models F \text{ and } \rho \cup \{(x_i, v)\} \subseteq \rho' \text{ and } c(\rho') \geq C_{min})\}$. One way to implement this is to reduce this min-bounding $c(\rho) \geq C_{min}$ to the already described max-bounding $-c(\rho) \leq -C_{min}$ by replacing the cost function c with $-c$. We have proven the following:

Theorem 2. *Given the configuration model $C(X, D, F)$ and its compiled solution space B, and the partial user assignment ρ, the problem of calculating valid domains for max-restriction $D_i^{\rho, C_{max}}$ and calculating valid domains for min-restriction $D_i^{\rho, C_{min}}$ is of polynomial complexity in the size of input B.*

Consider the *CVD* functionality where a user gets valid domains by restricting both the maximum and minimum value of any final configuration. We show that it is not possible to deliver this functionality with polynomial time guarantees in the size of the compiled BDD representation.

Theorem 3 (Infeasibility result). *Given the configuration model $C(X, D, F)$ and its compiled solution space B, and the partial user assignment ρ, the problem of calculating valid domains $D_i^{\rho, C_{max}, C_{min}} = \{v \in D_i \mid \exists \rho'.\rho' \models F \text{ and } \rho \cup \{(x_i, v)\} \subseteq \rho' \text{ and } C_{min} \leq c(\rho) \leq C_{max}\}$ is NP-hard in the size of input $|B|$, where $|B| = |V| + |E| + |X_b|$.*

Proof. We will prove our claim by reducing the NP-hard *subset-sum problem* (Cormen *et al.* 2001) to the problem of calculating valid domains $D_i^{\rho, C_{max}, C_{min}}$ over a linear size BDD B. The input to the subset-sum problem is defined with a set of constants $S = \{c_0, \ldots, c_{n-1}\}$ and a constant A. We create an input to *CVD*^{min,max} problem by constructing a Boolean configuration model $C_c(X, D, F, c)$ in the following way: all the variables are Boolean ($D_i = \{0, 1\}$), there are no constraints ($F = \{\}$) and costs satisfy $c_0^i = 0$, $c_1^i = c_i$. The BDD B corresponding to configuration model $C(X, D, F)$ has just one node T_1.

Now, the problem of answering whether there is a subset $S' \subseteq S$ such that $\sum_{c_i \in S} c_i = A$ corresponds to setting $C_{min} = C_{max} = A$ and checking if all domains are non-empty.

If there was an algorithm solving *CVD*^{min,max} in polynomial time w.r.t. $|B|$ then we could also solve the subset-sum problem in polynomial time because the *CVD*^{min,max} would allow us to efficiently check if domains are non-empty. Hence, the *min-max bounding problem* is NP-hard in the size of input B. □

Conclusions and Future Work

In this paper we presented a polynomial algorithm for cost-bounded interactive configuration functionality, where a user is allowed to restrict the maximum cost for any final valid configuration. We showed that it is NP-hard to extend this functionality to allow simultaneous restrictions of both the maximum and minimum costs of valid configurations. We have shown how this problem is used to allow an important product-configuration functionality of interactive price manipulation.

In the future we plan to test the applicability of this approach to large, real-world configuration benchmarks such as those in CLib (CLib online). We also plan to investigate other classes of cost-based valid domain restrictions that can be computed efficiently.

References

Amilhastre, J.; Fargier, H.; and Marquis, P. 2002. Consistency restoration and explanations in dynamic CSPs-application to configuration. *Artificial Intelligence* 1-2.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 8:677–691.

CLib. online. Configuration benchmarks library. `http://www.itu.dk/doi/VeCoS/clib/`.

Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. 2001. *Introduction to Algorithms*. The MIT Press, Second edition.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Hadzic, T.; Subbarayan, S.; Jensen, R. M.; Andersen, H. R.; Møller, J.; and Hulgaard, H. 2004. Fast backtrack-free product configuration using a precompiled solution space representation. In *PETO Conference*. DTU-tryk.

Hadzic, T.; Jensen, R.; and Andersen, H. R. 2006, online. Notes on Calculating Valid Domains. `http://www.itu.dk/~tarik/cvd/cvd.pdf`.

Jensen, R. M. online. CLab: A C++ library for fast backtrack-free interactive product configuration. `http://www.itu.dk/people/rmj/clab/`.

Madsen, J. N. 2003. Methods for interactive constraint satisfaction. Master's thesis, Department of Computer Science, University of Copenhagen.

Møller, J.; Andersen, H. R.; and Hulgaard, H. 2004. Product configuration over the internet. In *Proceedings of the 6th INFORMS*.

Subbarayan, S.; Jensen, R. M.; Hadzic, T.; Andersen, H. R.; Hulgaard, H.; and Møller, J. 2004. Comparing two implementations of a complete and backtrack-free interactive configurator. In *CP'04 CSPIA Workshop*, 97–111.