

Improving the Performance of Interactive Configuration with Regular String Constraints

Esben Rune Hansen, Peter Tiedemann, Henrik Reif Andersen

IT University of Copenhagen,
{esben,petert,hra}@itu.dk

Abstract

A generalization of the problem of interactive configuration has previously been presented in [1]. This generalization utilized decomposition to extend the standard finite domain interactive configuration framework to deal with unbounded string variables and provided features such as prefix autocompletion.

In this paper we present several significant improvements to the core data structures and algorithms in [1] as well as the first implementation of an interactive configurator on string variables.

The primary improvement is obtained by replacing the binary decomposition model with a finite domain model. We then describe an optimization for this model which allows us to replace the use of costly conjunctions with simple restrict operations during synchronization between decomposed constraints. In addition we describe how to improve the performance of the autocomplete operation, by using projection on relevant variables to significantly reduce the size of the data structures involved.

We empirically verify the critical significance of these improvements using our own implementation of a string variable based configurator on real-world example data.

1 Introduction

Interactive configuration is a type of decision support used when dealing with Constraint Satisfaction Problem (CSP) in which a user is assisted in the task of configuration by interacting with a piece of software known as a *configurator*. The user repeatedly chooses an unassigned variable and assigns it a value until all variables are assigned. The task of the configurator is to present only the choices for each of the unassigned that can lead to a complete solution to the problem. The set of valid choices for an unassigned variable x is called the *valid domain* of x [2].

As an example consider the problem of assigning values to the variables x_1, x_2 and x_3 where $x_1 \in \{1, \dots, 5\}$ and $x_2, x_3 \in \{1, \dots, 10\}$ with the requirement that $x_1 = 1 \vee x_1 = 2 \vee x_2 = 2$ and $x_2 = x_3$. Initially the user can choose to assign a value from $\{1, \dots, 5\}$ to x_1 or assign a value from $\{1, \dots, 10\}$ to x_2 or x_3 . Suppose the user assigns 3 to x_3 . In this case the valid domain of x_2 is $\{3\}$ and the valid domain of x_1 is $\{1, 2\}$. Obviously the valid domain of each unassigned variable has to be updated every time a value is assigned. Since the user will have to wait from the time he makes a choice to the time the configurator is ready to present the valid domains it is critical that the valid domains can be computed fast enough to ensure that the user experiences it as real-time interaction, which in practice means that the configurator has to update the valid domains within approximately 250 milliseconds [3]. In theory it is an NP-hard problem to calculate even the initial valid domains since this corresponds to achieving GAC for the entire constraint problem. However, if we have precompiled a *binary decision diagram* (BDD) [4] that represents the solutions to the set of constraints in the configuration problem, then we are able to keep the computation time polynomial in the size of the BDD. The BDD constructed in this manner can be exponentially large, but in practice BDDs have proved themselves to be tractable for many configuration problems occurring in practice [2].

Standard configurators based on this approach use a set of binary variables in the BDD to represent finite domain variables in the configuration problem. In this paper we will consider the case of variables that take unbounded strings as their values which implies that a standard finite domain configurator cannot be used to achieve interactive configuration.

As a small example of the functionality we want to provide, suppose that a user has to fill in a form containing a number of input fields under some set of constraints on the content of the input fields. Consider a CSP with the string variables phone, country, zip and district along with the following constraints:

I The prefix of phone is “+45” \iff country = “Denmark”

II country = “Denmark” \implies zip has four digits

III zip = “2300” \wedge country = “Denmark” \iff district = “Copenhagen S”

Suppose that the user enters district = “Copenhagen S”. This restricts the valid domain of zip to the singleton set {“2300”} and the valid domain of country to {“Denmark”} by (III). The valid domain of phone is decreased to the set of strings which has “+45” as a prefix by (I).

Suppose instead that the user has entered phone = “+4523493844”. This decreases the valid domain of country to {“Denmark”} by (I), and the valid domain of zip to strings consisting of 4 digits. This restriction can be performed as soon as the user have entered “+45”, since every completion of phone achieved by appending additional letters at the end of phone still will have “+45” as a prefix.

2 Related Work

As mentioned in the introduction, interactive configuration of variables with finite integer domains has already been described in [2]. Other previous results take a similar approach but compiles into other representations than BDDs, such as for example a *Deterministic Finite Automaton* (DFA) [5].

The idea of using regular expressions as constraints on variables has been considered from two different perspectives by [6] and [7]. In [6] regular expressions are applied as a global constraint to the variables in the constraint network where each variable is considered as a letter and the alphabet corresponds to the domains of the variables. Hence [6] considers only regular expression constraints on a single string of fixed size unlike the unbounded strings considering in this paper and [1]. In [7] the domains of string variables are specified by regular string constraints, but unlike the result in this paper and [1] their approach does not allow mixing regular constraints with logical constraints.

3 Preliminaries

Given a constraint c , we will use c interchangeably as the constraint itself and the set of solutions it allows. Additionally given a set $C = \{c_1, \dots, c_k\}$ of constraints, we also use C to denote the constraint representing $\bigwedge_{1 \leq j \leq k} c_j$.

Definition 1 (Regular CSP) A regular CSP is a triple $CSP = (X, \Sigma, C)$. Where $X = \{x_1, \dots, x_n\}$ is a set of variables, each on the domain Σ^* , where Σ denotes an alphabet. Finally $C = \{c_1, \dots, c_{|C|}\}$ is a set of constraint of the form

$$c ::= c \vee c \mid \neg c \mid match(x_i, \alpha)$$

where α is a regular expression over Σ and $1 \leq i \leq n$.

The expression $match(x, \alpha)$ evaluates to true iff $x \in L(\alpha)$, where $L(\alpha)$ is the language defined by the regular expression α . We use $f \wedge g$, $f \implies g$ and $f \iff g$ as shortcuts for $\neg(\neg f \vee \neg g)$, $\neg f \vee g$ and $(f \implies g) \wedge (g \implies f)$ respectively.

Regular expressions are written using the syntax

$$\alpha ::= \alpha\alpha \mid \alpha|\alpha \mid \alpha * \mid w$$

listed in increasing order of strength of binding, where $w \in \Sigma$.

We denote by $\rho = \{w_1, \dots, w_n\}$, where $\rho_i = w_i$, a complete assignment of the values $w_1, \dots, w_n \in \Sigma^*$ to the variables x_1, \dots, x_n . An assignment ρ is a solution to the regular CSP (X, Σ, C) if the ρ satisfies all the constraints in C . An assignment ρ' is an extension of ρ iff $\rho'_i = \rho_i \cdot \bar{w}_i$ for $\bar{w}_i \in \Sigma^*$. Note that ρ is therefore considered an extension of itself.

Definition 2 (Extendable assignment) For a regular CSP $= (X, \Sigma, C)$ an assignment ρ is extendable iff there exists an extension ρ' of ρ for which ρ' is a solution to the regular CSP.

The core problem discussed in [1] is that of tracking whether or not an assignment is extendable under append operations to X . Based on this information decision support can be provided in the form of autocompletion and suggestions.

Definition 3 (DFA) We define a DFA as $M = (Q, \Sigma, \delta, s, A)$, where Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow Q$ is a transition function, Σ is some alphabet, $s \in Q$ is the starting state and $A : Q \rightarrow \mathcal{B}$ is a function such that a state q is accepting iff $A(q) = true$.

We use $\hat{\delta}(q, w)$ as a shorthand for $\delta(\dots \delta(\delta(q, w_1), w_2), \dots, w_l)$, where (w_1, \dots, w_l) are the letters of a word in Σ^* . A word $w \in \Sigma^*$ is accepted by M iff $A(\hat{\delta}(s, w)) = true$. We use $\hat{\delta}(w)$ as a shortcut for $\hat{\delta}(s, w)$. Further we say that a state in a DFA q is reachable from a state p , denoted $p \rightsquigarrow q$ iff there exists a word $w \in \Sigma^*$ such that $\hat{\delta}(p, w) = q$.

4 Decomposing the regular CSP

The general idea presented in [1] is to separate the logical constraints defined by the logical operators in the regular CSP from the semantics of the match-expressions. As a simple example consider the regular CSP

$$match(x_1, \alpha) \wedge match(x_2, \beta) \vee \neg match(x_2, \gamma)$$

where α, β and γ are three regular expressions. The boolean connectors \wedge, \vee and \neg define a SAT problem if the match-expressions are replaced by boolean variables. In this case

$$b_1 \wedge b_2 \vee \neg b_3$$

We call this problem the *logical constraint*.

4.1 Logical constraint

For every match-expression $match(x_i, \alpha_j^i)$ we introduce a boolean variable y_j^i , where $y_j^i = true$ corresponds to $\rho_i \in L(\alpha_j^i)$ and $y_j^i = false$ corresponds to $\rho_i \notin L(\alpha_j^i)$ and let $Y_i = \{y_j^k \mid k = i\}$. Every assignment ρ to the variables in X corresponds to an assignment

$$\psi_i = (\rho_i \in L(\alpha_1^i), \dots, \rho_i \in L(\alpha_{m_i}^i))$$

for each set of variables Y_i where m_i is the number of match-expressions on the variable x_i .

If we replace every match-expression in the constraints in C with their corresponding y -variable we get a SAT problem which we denote C_L . While an assignment ρ solving C implies assignments to Y that satisfies C_L , the reverse is not true. To see this consider the regular constraint $match(x_1, "a") \wedge match(x_1, "b")$. Though $\psi_i = (true, true)$ satisfies the logical constraint there is no assignment to the variable x_1 that corresponds to such an assignment. We therefore need some further constraints on the Y -variables to model this restriction.

4.2 The reachability constraint

In order for the regular CSP to be satisfied by a solution to the logical constraint it has to be allowed by the semantics of the match expressions. That is, the assignments to the logical variables corresponding to match expressions on the same variable x_i , has to be simultaneously achievable by a single word assigned to x_i .

We define the *acceptance value* of a word assigned to x_i as

$$a_i(w) = (match(w, \alpha_1^i), \dots, match(w, \alpha_{m_i}^i)).$$

Informally, the acceptance $a_i(w)$ is the tuple of evaluations of each match expression on x_i of the word w .

Furthermore, for a string variable x_i and a word w we define the *reachability constraint* $R_i(w)$ over the variables Y_i :

$$R_i(w) = \{a_i(w') \mid w' = w \cdot \bar{w}\}$$

That is, the reachability constraint $R_i(w)$ restricts the variables in Y_i to those assignments that are consistent with

some word assigned to x_i that can be obtained by appending to w .

Given an assignment ρ to X we define \bar{C} as:

$$\bar{C} = C_L \wedge \bigwedge_{1 \leq i \leq n} R_i(\rho_i)$$

This SAT problem has the property that ρ is extendable iff \bar{C} is satisfiable.

In the encoding of match-expression we will always add a special symbol, say $\langle EOL \rangle$, at the end of each regular string in the match-expressions, representing that the typed input is *completed*, by the user typing enter, leaving the input field or similar. For simplicity we will not consider this.

4.3 Core configuration algorithms

In the following we sketch the core algorithms that is used in both the previous approach of [1] and in this paper. Below ρ is the current assignment to X and initially $\rho = (\epsilon, \dots, \epsilon)$. Therefore we initially compute \bar{C} as

$$\bar{C} \leftarrow C_L \wedge \bigwedge_{1 \leq i \leq n} R_i(\epsilon)$$

The first algorithm is $APPEND(x_i, w)$ which updates \bar{C} to reflect that the word w has been appended by the user to the variable x_i . Before the append is performed it is checked whether ρ will still be extendable after w is appended to ρ_i .

$APPEND(x_i, w)$

- 1 **if** $\bar{C} \wedge R_i(\rho_i w)$ is unsatisfiable
- 2 **then** error "invalid append"
- 3 $\rho_i \leftarrow \rho_i w$
- 4 $\bar{C} \leftarrow \bar{C} \wedge R_i(\rho_i)$
- 5 **for each** $(x_j \in X)$
- 6 **do** $AUTOCOMPLETE(x_j)$

An update using $APPEND$ might enable us to *autocomplete* one or more variables. A word w assigned to x_i can be autocompleted to $w \cdot \bar{w}$ iff $w \cdot \bar{w}$ is a prefix of all valid assignments to x_i . The algorithm $AUTOCOMPLETE$ shown below performs this update if possible.

$AUTOCOMPLETE(x_i)$

- 1 **if** $\bar{C} \wedge a_i(\rho_i w)$ is unsatisfiable and
 there exists exactly one $w \in \Sigma$ for which
 $\bar{C} \wedge R_i(\rho_i w)$ is satisfiable
- 2 **then** $\rho_i \leftarrow \rho_i w$
- 3 $AUTOCOMPLETE(x_i)$

The final algorithm is $SUGGEST$. It is initially called with the parameters (x_i, ϵ) and returns the valid domain of x_i (in practice an upper limit is put on the number of strings returned).

```

SUGGEST( $x_i, w$ )
1   $S \leftarrow \emptyset$ 
2  if  $\bar{C} \wedge \{a(\rho_i \cdot w)\}$  is satisfiable
3    then  $S \leftarrow S \cup \{\rho_i \cdot w\}$ 
4  for each  $w' \in \Sigma$  for which
       $\bar{C} \wedge R_i(\rho_i \cdot w \cdot w')$  is satisfiable
5    do SUGGEST( $x_i, w \cdot w'$ )
6  return  $S$ 

```

4.4 Implementation

We will now briefly sketch how the logical operations and the computation of the reachability constraints in the above algorithms were implemented in [1]. Initially the logical constraint is compiled into a single MDD[8] using the standard MDD logical operators. Following this a *deterministic finite automata* F_j^i is build for each α_j^i . Let $F^i = \{F_j^k \mid k = i\}$.

In order to combine the constraints on each string variable x_i we compute the the *minimized product DFA* of the DFAs in F^i as defined below.

Definition 4 ((Minimized) Product DFA – (M)PDFA)

A product DFA of a set of DFAs $\{(Q_1, \Sigma, \delta_1, s_1, A_1), \dots, (Q_k, \Sigma, \delta_k, s_k, A_k)\}$ is the automata $M'_\times = (\times_{1 \leq j \leq k} Q_j, \Sigma, \delta, \times_{1 \leq j \leq k} s_j, \times_{1 \leq j \leq k} A_j)$ where $\delta((q_1, \dots, q_k), a) = (\delta_1(q_1, a), \dots, \delta_k(q_k, a))$. We use $M_\times = (Q_\times, \Sigma_\times, \delta_\times, s_\times, a_\times)$ to denote the minimized product DFA obtained by minimizing M'_\times .

The minimized product DFA can be constructed in linear time in the size of the output using a simultaneous DFS in all k DFAs, constructing a state in the MDFA for each distinct combination of states in the DFAs that is encountered in the DFS [1]. For convinience, given a state $q \in M'_\times$ where $\hat{\delta}_\times(s_\times, w) = q$ we let $R_i(q) = R_i(w)$.

4.4.1 Computation of reachability constraints

Given the minimized product DFA $M'_\times = (Q_\times, \Sigma_\times, \delta_\times, s_\times, a_\times)$ for F^i we can evaluate $a_i(w) = a_i(\hat{\delta}_\times(s_\times, w))$ easily, but as indicated in the algorithms shown above it is also necessary that we can compute $R_i(w)$.

The algorithm that computes the reachability constraints labels each state with $B(q) = \{a_i(q)\}$. In the next step the strongly connected components $\mathcal{S} = \{S_1, \dots, S_k\}$ in M'_\times are found and each S_j is associated a label $B(S_j) = \{a_i(q') \mid q' \in S_j\}$. The strongly connected components are then considered in reverse topological order. For each component S_j with neighbors S_{k_1}, \dots, S_{k_c} , $B(S_j)$ is updated to $\bigcup_{1 \leq i \leq c} B(S_{k_i})$. This allows $R_i(q)$ to be computed as $B(S_q)$. The labels and final reachability constraints are represented using MDDs.

Since both reachability constraints and the logical constraint are now represented as MDDs, we can easily support the operations required by APPEND, AUTOCOMPLETE and SUGGEST.

5 Improved representation of acceptance values

The approach described in Section 4 has two critical performance bottlenecks. The first is Line 2 of APPEND, which performs a conjunction between the MDD representing the current logical model \bar{C} and a reachability constraint. The second is Line 1 in AUTOCOMPLETE which is called n times for every call to APPEND and involves several conjunctions between the logical model and a reachability constraint in order to determine satisfiability.

Conjoining two MDDs G_1 and G_2 requires time $O(|G_1| \cdot |G_2|)$ in the worst case. As pointed out in [1] this issue makes it impossible to guarantee a response time linear in the size of the compiled representation. This guarantee is an important feature in the standard interactive configuration scheme.

In this section we will consider two ways to tackle the bottleneck:

- In Section 5.1 we will reduce the problem by encoding the acceptance values in a more efficient way. This will reduce the size of the MDD encoding of both the reachability constraints and the MDD encoding of \bar{C} . Hence the time required to perform the conjunctions will be decreased significantly.
- In Section 5.2 we will change the representation of the reachability constraints in such a way that we can replace each conjunction of the MDDs in the algorithms, described in Section 4, by a restrict-operation. Restrictions can be done in linear time in the size of the MDD and hence we can obtain the linear response time guarantee that could not be provided in [1].

In the approach described in Section 4, the acceptance values are modelled directly with boolean variables. This approach has the advantage that it makes it very simple to construct the MDD as each match-expression corresponds to exactly one boolean variable. Both improvements that we will consider in the next two subsections will involve changing this simple representation to a more complex but also more efficient representation.

For the sake of simplicity we will in the reminder of Section 5 neglect the fact that (almost) every state in a PDFA has a transition to a state corresponding to that all DFAs are rejecting with a self-loop on all $w \in \Sigma$.

5.1 Integer encoding

In many cases only a small fraction of the 2^{m_i} possible acceptance values for a variable x_i can occur because the regular constraint tightly limits the simultaneous allowed assignments to Y . We can use this fact to create a much more compact MDD to represent the logical constraints of the regular CSP. To this end we introduce for each string variable x_i an integer valued variable \bar{y}_i and for each PDFA we assign a unique integer id $I^i(a_\times^i(q_i))$ to each occurring acceptance value $a_\times^i(q_i)$. Further, we denote by \mathcal{I}_j^i the set of ids of acceptance values satisfying the j th match expression on x_i . That is:

$$\mathcal{I}_j^i = \{I^i(a_\times^i(q_i)) \mid q_i \in Q_\times^i \wedge \text{the } j\text{th entry in } a_\times^i(q_i) \text{ is true}\}$$

Suppose that α_j^i is the regular expression of the j occurrence of a match expression on x_i . We can now express the corresponding logical constraints by replacing each occurrence of $\text{match}(x_i, \alpha_j^i)$ (which in Section 4 was replaced by y_j^i) with the disjunction $\bigvee_{a \in \mathcal{I}_j^i} (\bar{y}_i = a)$. That is, we specify that \bar{y}_i must be assigned to one of the integer ids that correspond to a state in the PDFA satisfying the match expression. The use of this encoding implies new reachability constraints $\bar{R}_i(q)$ restricting \bar{y}_i to the set of ids reachable from q in M_\times^i .

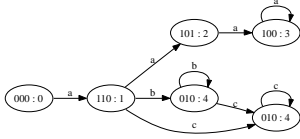


Figure 1. A PDFA on the constraint from Example 1. Each state q in the figure is labeled with $a : b$ where $a = a_\times^1(q)$ (showing *true* as 1, and *false* as 0) and $b = I^1(a_\times^1(q))$.

Example 1 Consider the constraint

$$\text{match}(x_1, aa^*) \wedge (\text{match}(x_1, ab * c^*) \vee \text{match}(x_1, aa)) \quad (1)$$

and the corresponding PDFA in Figure 1. In this example 5 of the 2^3 possible acceptance values occurs in the PDFA. Note that since the acceptance value 010 appears twice the integer id 4 appears twice as well.

In the encoding from Section 4 the representation of \bar{C} would be $y_1^1 \wedge (y_2^1 \vee y_3^1)$. In the integer encoding we have numbered the five acceptance values with $0, \dots, 4$, such that

$$\begin{aligned} \bar{y}_1 = 0 &\iff (\neg y_1^1 \wedge \neg y_2^1 \wedge \neg y_3^1) \\ \bar{y}_1 = 1 &\iff (y_1^1 \wedge y_2^1 \wedge \neg y_3^1) \\ \bar{y}_1 = 2 &\iff (y_1^1 \wedge \neg y_2^1 \wedge y_3^1) \\ \bar{y}_1 = 3 &\iff (y_1^1 \wedge \neg y_2^1 \wedge \neg y_3^1) \\ \bar{y}_1 = 4 &\iff (\neg y_1^1 \wedge y_2^1 \wedge \neg y_3^1) \end{aligned}$$

which entails that

$$\begin{aligned} y_1^1 &\iff \bar{y}_1 = 1 \vee \bar{y}_1 = 2 \vee \bar{y}_1 = 3 \\ y_2^1 &\iff \bar{y}_1 = 2 \\ y_3^1 &\iff \bar{y}_1 = 1 \vee \bar{y}_1 = 4 \end{aligned}$$

Hence the integer constraint will be

$$(\bar{y}_1 = 1 \vee \bar{y}_1 = 2 \vee \bar{y}_1 = 3) \wedge (\bar{y}_1 = 2 \vee (\bar{y}_1 = 1 \vee \bar{y}_1 = 4))$$

which reduces to $\bar{y}_1 = 1 \vee \bar{y}_1 = 2$.

5.2 Updates using restrict

In order to achieve that restrict operations are sufficient for updates we need to adapt the MDD on \bar{C} to the structure of the PDFA as described below.

Definition 5 (Connection Graph) A graph $G_\times(V_\times, E_\times)$ is a connection graph on a PDFA M_\times^i iff there exists a bijective mapping $\Gamma : Q_\times \rightarrow V_\times$ and for any $p, q \in Q_\times$ it holds that $(\Gamma(p), \Gamma(q)) \in E_\times \iff \exists w \in \Sigma : \delta_\times(p, w) = q$.

We define reachability constraints for the nodes in V_\times as $R_i(\Gamma(q)) = R_i(q)$ and the acceptance value of the nodes as $a_i(\Gamma(q)) = a_i(q)$.

Note that the edges in the connection graph are unlabeled and that all transitions between p to q are represented by the single unlabeled edge $(\Gamma_v(p), \Gamma(q))$.

We define *contraction* of an edge (u, v) as redirecting all start-points and end-points of edges from u to v , that is, replacing all (u, u') by (v, u') and all (u', u) by (u', v) in E , and then removing u from V .

Definition 6 (Macro-DAG) A Macro-DAG on a PDFA M_\times^i is a connection graph (V_\times, E_\times) on M_\times^i where all edges $(u, v) \in E_\times$ where $R_i(u) = R_i(v)$ are contracted. The height h of the DAG is the length of the longest path in the DAG. Nodes with an out-degree of zero are called terminals and are labeled with h . All other nodes are called internal nodes and are labeled with the length of the longest path from the root to the node. The edges in the Macro-DAG is labeled in such a way that any pair of edges with the same start-point have different labels.

Note that all internal nodes have labels that are less than h . A Macro-DAG is essentially an MDD where terminals with different reachability constraints are kept separate.

Every $match(x_i, \alpha_j^i)$ is represented by an MDD. This MDD is obtained by removing all terminals t that corresponds to violating $match(x_i, \alpha_j^i)$ from the Macro-DAG (V_x^i, E_x^i) , removing all nodes from which a terminal is not reachable and merging isomorphic nodes. The variable labeling of each MDD node v is $\hat{y}_{offset+l(v)}$ where $l(v)$ is the label on v and offset is the first variable that is not used in the encoding of the variables x_1, \dots, x_{i-1}

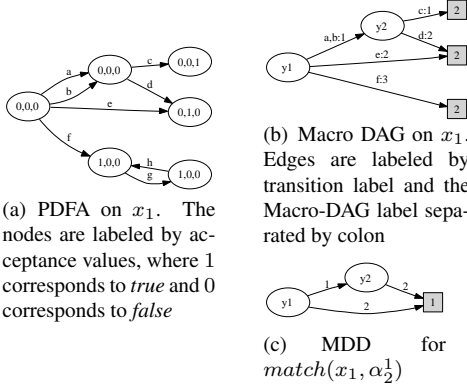


Figure 2. Data structures representing the CSP consisting of the constraint $match(x_1, (a|b)c) \vee match(x_1, ((a|b)d)|e) \vee match(x_1, f(gh)*)$

Figure 2 shows how a Macro-DAG is obtained from a PDFA and how the MDD of a match-expression is obtained from a Macro-DAG. Note that the two edges labeled a and b in Figure 2(a) are represented by a single edge in the Macro-DAG. The two nodes with acceptance value 1, 0, 0 in Figure 2(a) are contracted into one node in the Macro-DAG.

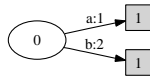
The only terminal in the Macro-DAG that does not violate the the second match-expression is the middle one, hence only this one terminal is present in Figure 2(c).

The next example clarifies how the MDD is constructed for each of the three representations in Section 4, 5.1 and 5.2 respectively.

Example 2 Consider

$$\begin{aligned} f_1 &= match(x_1, (a|b)c) && \wedge && match(x_2, a) \\ f_2 &= match(x_1, ((a|b)d)|e) && \wedge && match(x_2, a) \\ f_3 &= match(x_1, f(gh)*) && \wedge && match(x_2, b) \end{aligned}$$

and regular CSP on the constraint $f_1 \vee f_2 \vee f_3$. The Macro-DAG on x_1 is the one shown in Figure 2(b), and the Macro-DAG on x_2 is:



In Figure 3 we have shown how the MDD on \bar{C} will be constructed given the three different encodings. For each encoding four MDDs encoding f_1, f_2, f_3 and $f_1 \vee f_2 \vee f_3$ respectively are shown. We have assumed that the match-expressions are numbered in order of occurrence in the constraint. In the integer encoding the acceptance values of x_1 are numbered 001 = 1, 010 = 2 and 100 = 3. For x_2 the numbering is 10 = 1 and 01 = 2.

Suppose that $x_1 = x_2 = \epsilon$ and that an a is appended to x_1 . In the encoding in (a – d) an MDD corresponding to $\neg y_1 \wedge \neg y_2 \wedge y_3 \vee \neg y_1 \wedge y_2 \wedge \neg y_3$ is conjoined with the MDD from Figure 3(d). In the encoding in (e – h) an MDD corresponding to $\bar{y}_1 = 1 \vee \bar{y}_1 = 2$ will be conjoined with the MDD from Figure 3(h). In the encoding in (i – l) the value of \hat{y}_1 will be restricted to 1 in the MDD from Figure 3(h).

6 Improving performance of autocomplete

The approach in [1] suggests performing the satisfiability check in Line 1 of AUTOCOMPLETE by actually computing the conjunction using the MDD apply operator. This will be very expensive both asymptotically and in practice. A very significant improvement can be achieved by using projections.

Definition 7 (Projection) We denote by the projection of \bar{C} on x_i , written as $\bar{C} \downarrow x_i$ as the constraint \bar{C} where all but the variables that corresponds to x_i are removed from \bar{C} by existential quantification.

The general idea is to compute $(\bar{C} \downarrow x_i) \wedge R_i(\rho_i w)$ instead of $\bar{C} \wedge R_i(\rho_i w)$ in Line 1 of AUTOCOMPLETE. We note that we can compute a projection on all the variables in X by a single scan through the MDD hence in linear time in the size of the MDD. This can be done simply by copying each variable layer and reduce it.

Note that making the projection on \bar{C} is related to computing the Valid Domains on \bar{C} [9].

7 Experiments

We have implemented a configurator for regular CSPs that is able to choose between the three different representations of acceptance values described in Section 4.4.1. The implementation was done in C# using the BudDDy package [10] and the C# port supplied as a part of the C# implementation of CLab [11]. MDDs have been encoded as BDDs by using log-encoding [9], hence the size mentioned in the experiments are counting BDD nodes.

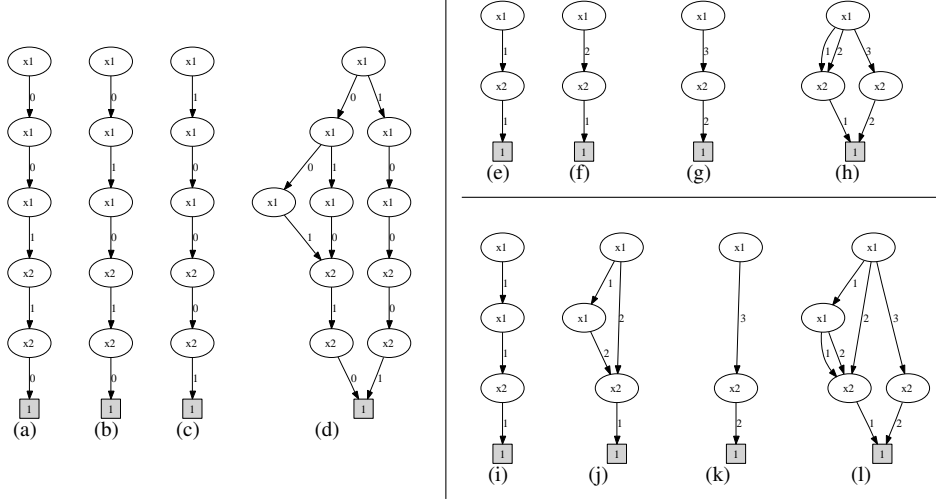


Figure 3. The construction of the MDD based the encoding from Section 4 (a-d), Section 5.1 (e-h) and Section 5.2 (i-l)

7.1 Database scenario

For the first part of our experiments we consider a scenario where we want to assist the user in filling out a form that is restricted based on the content of several relational database tables. We start by describing how we model this problem as a regular CSP.

Let $S = \{x_1, \dots, x_k\} \subseteq X$ be a set of string variables and let T be a table with k attributes containing m tuples $(t_1^1, \dots, t_k^1), \dots, (t_1^m, \dots, t_k^m)$.

We want to model the constraint c such that for any assignment of values a_1, \dots, a_k to the string variables x_1, \dots, x_k the constraint c is satisfied iff (a_1, \dots, a_k) corresponds to a tuple in T , that is $\bigvee_{1 \leq i \leq m} \bigwedge_{1 \leq j \leq k} a_j = t_j^i$. We can express this as the *table* constraint on the variables in S as $\bigvee_{1 \leq i \leq m} \bigwedge_{1 \leq j \leq k} \text{match}(x_j, t_j^i)$. using the fact that $x = w$ and $\text{match}(x, w)$ are equivalent iff $w \in \Sigma^l$ for some l , that is, w is a word. When constructed in this fashion, the MDFAs are likely to result in a tree structure. Further if we assume that no match-expression occurs more than once in the constraints every state in the MDFA will correspond to acceptance of at most one match-expression.

Note that the conjunction of the table constraints of two relations R and T is equivalent to the table constraint of the *join* that is $R \bowtie T$.

7.2 Results

We loaded a database join with $91 \times 91 \times 77 = 637637$ tuples based on a join from the Northwind database instance [12]. We joined two copies of the *Customers* table (one with renamed attributes) and the *Products* table. The tables have no attributes in common. We show the results of testing

interaction sequences with our configurator on the resulting string configuration instance in Figure 4(a) and 4(b).

7.3 Car configuration scenario

In addition to the above database data, we also loaded the reault instance from CLib [13] (the largest configuration instance available). This instance represents realworld data related to the the configuration of a specific car model. Compiled as a BDD it has a node count of approximately 450000 nodes. Since this instance is a finite domain constraint problem, we turned it into a string variable problem by mapping each finite domain value into a string drawn from a database. For this scenario we only tested with the integer encoding technique from Section 5.1, and not for the encoding used in [1] or the integer encoding from Section 5.2. We generated a set of interaction sequences for this problem. The max interaction time was in this case 350 ms.

8 Conclusion

We have presented several improvements to the approach in [1]. We have implemented a configurator that can use the approach from [1] as well as the improved approaches presented in this paper. By using the implemented configurator we have empirically showed that the representation of the constraints suggested in [1] fail to give a real-time response time (250 ms) on a real-world instance, that our improved representation easily can achieve.

8.1 Future work

On the implementation side, one additional optimization is possible. Earlier in the paper we have been consider-

	[1]	Sec. 5.1	Sec. 5.2
BDD nodes in RCs	210341	27182	6773
BDD nodes in AVs	147173	12709	22556
BDD nodes in \bar{C}	143038	13356	23877
States in MDFAs	16998	16998	16998

(a) This figure shows the sizes of the data structures involved in modeling the database instance described in Section 7.2. As expected the integer encoding from Section 5.1 greatly reduces the size of the BDDs representing the logical constraints and reachability constraint (RC) compared to the representation used in [1]. The representation from Section 5.2, yields a somewhat larger logical BDD, but smaller BDDs for the reachability constraints. The former is due to the additional variables used to encode acceptance values, while the latter is caused by the improved prefix sharing.

	bitvector		integer		Operation count
	no proj	proj	Sec. 5.1	Sec. 5.2	
Total APPEND	1710	1690	140	260	47
Total BDD Projection	0	15610	2080	2460	435
Total AUTOCOMplete	93454	40	20	140	435
Max Response time	5721	520	40	10	

(b) The results shown in this table is obtained by loading the database described in Section 7.2 and making a small scenario by handpicking some append operations that require a lot of synchronization. The figure shows the total time used in the entire scenario by APPEND, AUTOCOMplete and BDD projections respectively. The number of times each of the three operations are performed in the scenario are listed in the right-hand column. The critical performance parameter in the figure is the response time. The approach of [1] provides a completely unacceptable response time of nearly 6 seconds. Even when projection are used, the maximal response time is problematic. Both integer based representations provide acceptable response times. In comparing the integer representations, we note that even though the integer representation of Section 5.2 is slower on *average* than the one from Section 5.1 it is better in the *worst case*. This is due to the fact that the former representation is very fast for the first few assignments, which is usually the most time consuming, since the size of \bar{C} is large and a lot of pruning will be performed

Figure 4.

ing modifications of the decision diagram by AND and RESTRICT operations. In practice it will be more efficient to avoid modifying the decision diagram and instead simply store the list of values that has not been restricted. In this scenario the projections can still be computed efficiently.

On the theoretical side we plan to consider more expressive representations than regular and DFAs for representing the string domains as well as allowing more complex constraints on string variables than just match expression.

References

- [1] Hansen, E.R., Andersen, H.R.: Interactive configuration with regular string constraints. In: Association for the Advancement of Artificial Intelligence. (2007)
- [2] Hadzic, T., Subbarayan, S., Jensen, R.M., Andersen, H.R., Hulgaard, H., Møller, J.: Fast backtrack-free product configuration using a precompiled solution space representation. In: Proceedings of the International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems, DTU-tryk (2004) 131–138
- [3] Raskin, J.: The Humane Interface. Addison Wesley (2000)
- [4] Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers **35**(8) (1986) 677–691
- [5] Amilhastre, J., Fargier, H., Marquis, P.: Consistency restoration and explanations in dynamic CSPs-Application to configuration. Artificial Intelligence **135**(1-2) (fvrier 2002) 199–234 bb modif 28/02/02.
- [6] Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004), Springer (2004) 482–495
- [7] Golden, K., Pang, W.: Constraint reasoning over strings. In: CP. (2003) 377–391
- [8] Kam, T., Villa, T., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: Multi-valued decision diagrams: Theory and applications. International Journal on Multiple-Valued Logic **4** (1998) 9–62
- [9] Tarik Hadzic, Rune Møller Jensen, H.R.A.: Calculating valid domains for bdd-based interactive configuration. (2006) Available in 2007 at: <http://www.itu.dk/people/tarik/cvd/cvd.pdf>.
- [10] Lind-Nielsen, J.: BuDDy - A Binary Decision Diagram Package. <http://sourceforge.net/projects/buddy> (online)
- [11] Torbjørn Meistad, Y.R., Lindsve, G.T.: A configuration support tool based on constraint programming and binary decision diagrams. Available online at: <http://www.itu.dk/people/rmj/data/systems/ClabSharp10.zip> (2007)
- [12] Microsoft: Northwind sample databases. Available online at: <http://msdn.microsoft.com/en-us/library/ms143221.aspx> (2004)
- [13] CLib: Configuration benchmarks library. Available online at: <http://www.itu.dk/research/cla/externals/clib/> (2007)