Interactive Configuration Based on Linear Programming

Tarik Hadzic
Henrik Reif Andersen

IT University Technical Report Series

TR-2005-67

ISSN 1600–6100

June 2005

ISSN 1600–6100

ISBN 87-7949-097-2

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: +45 72 18 50 00
Telefax: +45 72 18 50 01
Web www.itu.dk

Interactive Configuration Based on Linear Programming

Tarik Hadzic, Henrik Reif Andersen

Department of Innovation,
IT University of Copenhagen, Denmark
tarik@itu.dk,hra@itu.dk

Abstract. Interactive configuration denotes a process of a user interactively specifying a product (or a service) using a supporting program called a *configurator*. Choices for each available product component are usually modelled as variables over finite domains, and the knowledge about the valid product specifications is encoded as propositional constraints over these variables. Interactive configuration over finite domains is NP-hard. Most solution approaches therefore either give up on some interactive requirements or move the NP-hard part to an offline phase by first compiling the set of valid assignments to efficient structures (such as reduced ordered BDDs) and then performing polynomial interactions online.

In this paper we consider the case when all the constraints are linear inequalities and when the variable domains are the set of real numbers. Using results from the field of linear programming (LP) we show that in this case the interactive configuration can be performed in polynomial time. We moreover show how the simplex algorithm (in worst-case exponential but performing very well in practice), can be efficiently adapted to support interactive configuration. We also identify and implement some new, LP-specific configuration functionalities, and illustrate how the concept of interactive configuration can be used in classical LP problems, especially to provide support for interactively selecting values for variables.

1 Introduction and Related Work

Configuration problems emerged as a research topic in the 1980s as the result of a manufacturing shift from mass-production to mass-customization. *Interactive configuration* is an important application area where a user interactively tailors a product (a car, a PC, a device driver,...) to his specific needs using a supporting program called the *configurator*. Choices for each available component are usually modelled as variables over finite domains, and the knowledge about the valid product specifications is encoded as propositional constraints over these variables.

Interactive configuration functionalities (such as giving feedback to user about available choices) should satisfy a number of well defined user-friendly requirements under fast response time limitations. The interactive configuration problem is NP-hard. Constraint satisfaction problem (CSP) approaches [7, 6] therefore give up on some of the user-friendliness requirements, while symbolic approaches have to divide the computational effort to an offline and an online phase. First they compile valid assignments to efficient data structures, such as reduced ordered BDDs [8, 20]. If the compiled representation is small enough, then the already available efficient algorithms deliver basic configuration functionalities satisfying all the requirements. A limitation for symbolic approaches is the inability to efficiently model arithmetic constraints over infinite domains.

Presently, we investigate systems modelled only by linear arithmetic constraints. This might be used for later efficient modelling of more involved hybrid systems. The field of linear programming (LP) has extensively studied the optimization involving only linear arithmetic constraints, and has provided us with efficient algorithms and strong theoretical results. Using these results, we show that the configuration functionalities can be delivered in polynomial time. We also demonstrate how the idea of interactive configuration could be utilized in classical LP optimization problems.

Related work in symbolic model checking of real-time and hybrid systems [14, 4] addresses the problem of combining both propositional and arithmetic constraints without considering the requirements imposed by interactive configuration.

Hybrid approaches for handling both discrete and linear constraints using CSP and LP techniques are becoming increasingly popular [13, 9]. Most of these approaches are used for problems in the area of combinatorial optimization.

Sensitivity analysis is often used in linear programming to improve the performance of solving several closely related LP problems, or to answer how much the coefficients in the objective function can be changed without violating the optimal basis [22]. Therefore, it provides valuable feedback to the user about the entire range of possible solutions based on the optimal solution. This resembles the interactive configuration which provides (richer and stronger) feedback for any partial solution, at the price of greater computational cost.

The remainder of the paper is organized as follows. In Sect. 2, we formally define interactive configuration. Basic linear programming results are described in Sect. 3. In Sect. 4, we illustrate the use of interactive configuration in LP problems. In Section 5, we show how to implement the basic configuration functionalities. Finally, we demonstrate LP-specific configuration functionalities in Sect. 6 and draw conclusions in Sect. 7.

2 Interactive Configuration

Configuration formalisms are tightly related to *product configuration*, which is the primary application area since the 1980s. Our formal definition introduces variables, domains for the variables defining the combinatorial space of possible assignments and constraints defining which combinations are valid assignments. Each variable represents a product component, the variable domain refers to the options available for its component and constraints specify the rules that the product must satisfy.

Definition 1. *A* configuration problem $\mathcal{C}$ *is a triple* $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ *where* $\mathcal{X}$ *is a set of variables* $\{x_1, x_2, \ldots, x_n\}$*,* $\mathcal{D}$ *a set of their finite domains* $\{D_1, D_2, \ldots, D_n\}$ *and* $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$ *a set of propositional formulas over atomic propositions* $x_i = v$ *where* $v \in D_i$*, specifying conditions that the variable assignments have to satisfy.*

Formulas $\mathcal{F}$ are given by the following syntax:

$$f ::= \; x_i = v \mid f \wedge g \mid \neg f \tag{1}$$

(where values $v \in D_i$)

For a configuration problem $\mathcal{C}$, we denote the solution space $\mathcal{S}(\mathcal{C}) \subseteq D_1 \times D_2 \times \ldots \times D_n$ as the set of all valid assignments, i.e. the set of all assignments to the variables $\mathcal{X}$ that satisfy the rules $\mathcal{F}$. Many interesting questions about configuration problems are hard to answer. Just determining whether the solution space is empty is NP-complete, since the Boolean satisfiability problem can easily be reduced to it in polynomial time.

Example 2. Consider specifying a T-shirt by choosing the color (black, white, red, or blue), the size (small, medium, or large) and the print (”Men In Black” - MIB or ”Save The Whales” - STW). There are two rules that we have to observe: if we choose the MIB print then the color black has to be chosen as well, and if we choose the small size then the STW print (including a big picture of a whale) cannot be selected as the large picture of a whale does not fit on the small shirt. The configuration problem (X, D, F) of the T-shirt example consists of variables $X = \{x_1, x_2, x_3\}$ representing color, size and print. Variable domains are $D_1 = \{black, white, red, blue\}$, $D_2 = \{small, medium, large\}$, and $D_3 = \{MIB, STW\}$. The two rules translate to $F = \{f_1, f_2\}$, where $f_1 = (x_3 = MIB) \Rightarrow (x_1 = black)$ and $f_2 = (x_3 = STW) \Rightarrow (x_2 \neq small)$. There are $|D_1||D_2||D_3| = 24$ possible assignments. Eleven of these assignments are valid configurations and they form the solution space shown in Fig. 1.

$(black, small, MIB)$	$(black, large, STW)$	$(red, large, STW)$
$(black, medium, MIB)$	$(white, medium, STW)$	$(blue, medium, STW)$
$(black, medium, STW)$	$(white, large, STW)$	$(blue, large, STW)$
$(black, large, MIB)$	$(red, medium, STW)$	

Fig. 1. Solution space for the T-shirt example

Interactive configuration refers to the process of a user interactively assigning values to variables under given constraints by using a supporting program called a configurator. Each step in the user-configurator interaction includes a user selecting a value from a domain, and the configurator calculating valid domains for the other unassigned variables. Formally, let ρ denote a sequence of user assignments: $\rho = [x_{i_1} = v_{i_1}, \ldots, x_{i_k} = v_{i_k}]$ where all $i_1, \ldots, i_k$ are different and $v_{i_j} \in D_{i_j}$. Then, for each unassigned variable x_j the configurator calculates its valid domain $V_j^{\rho} \subseteq D_j$ with respect to assignments ρ. We use $V_j = V_j^{[\,]}$ when ρ is empty.

Unlike *batch configuration* [18, 19] where a system automatically finds one solution that respects user preferences, in interactive configuration the user explores the entire solution space by getting immediate feedback about the consequences of his assignments in the form of calculated valid domains. This is the core configurator functionality, and we refer to it as *Domain Calculation* (DC). The DC functionality has to satisfy the following requirements:

- *Completeness*: For each unassigned variable, *any* value that can be extended to a valid total assignment should be included in the calculated domain (i.e., we can specify any valid solution).
- *Validity*: Calculated domains should contain *only* those values that can be extended to a valid total assignment (i.e., we cannot make a selection that will eventually force us to backtrack).
- *Responsiveness*: The configurator's response time should be fast enough to provide a truly interactive user experience.

Validity and completeness ensure that the user cannot pick a value that is not a part of a valid solution, and furthermore, a user is able to pick all values that are part of at least one valid solution. These two requirements are hard to meet and often they are not satisfied in existing configurators, either exposing the user to backtracking or making some valid choices unavailable. When we add demand for short response-time the DC functionality becomes even harder to implement.

Example 3. For the T-shirt problem, the assignment $x_2 = small$ will, by the second rule, imply $x_3 \neq STW$ and since there is only one possibility left for variable x_3, it follows that $x_3 = MIB$. The first rule then implies $x_1 = black$. Unexpectedly, we have completely specified a T-shirt by just one assignment.

Other important interactive functionalities have been identified [17]. *Restoration* refers to the functionality of a user undoing the choice for some already assigned variable with the configurator recalculating valid domains. For example, should the user decide to remove assignment $x_{i_r} = v_{i_r}$ from ρ, the configurator would have to recalculate valid domains $V_j^{\rho\backslash\{x_{i_r}=v_{i_r}\}}$ with respect to the new sequence of assignments $\rho\backslash\{x_{i_r} = v_{i_r}\} = [x_{i_1} = v_{i_1}, \ldots x_{i_{r-1}} = v_{i_{r-1}}, x_{i_{r+1}} = v_{i_{r+1}}, \ldots, x_{i_k} = v_{i_k}]$.

Assisted conflict resolution allows a user to force an invalid choice for a variable $x_j = v_j$ ($v_j \in D_j\backslash V_j^\rho$). In response, he gets a minimal list of choices that need to be changed in order to restore consistency. This could be a list of assignments $\rho_r \subseteq \rho$ that has to be removed before the valid domains $V_j^{\rho\backslash\rho_r}$ are recalculated, and possibly a suggestion ρ'_r for new assignments to these variables.

In this paper, we consider the problem of providing domain calculation, restoration and assisted conflict resolution functionalities, while satisfying completeness, validity and responsiveness requirements for linear programming problems.

3 Linear Programming

We consider the *canonical form* of the linear programming problem:

Definition 4. *Given a set of n non-negative real variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, a linear objective function: $z = c_1x_1+c_2x_2+\ldots+c_nx_n$ and a set of m linear constraints $\mathcal{C} = \{C_1, \ldots, C_m\}$ of the form*

$$C_i : a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \leq b_i$$

($a_{ij}, b_i, c_j \in \mathbb{R}, i = 1, \ldots, m, j = 1, \ldots, n$)*, the problem of finding the assignment for $x = (x_1, \ldots x_n)$ that maximizes (minimizes) the objective function is called the* linear programming (LP) problem*.*

There are other, equivalent forms of the LP problem. In *general form* constraints are of the form $C_i : a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \approx b_i$ where $\approx\, \in \{=, \leq, \geq\}$, and variables x_i don't have to be nonnegative. However, our (canonical) definition of a linear program is not limiting, since every problem in general form can be translated to a canonical one. Namely, a constraint of the form $a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \geq b_i$ is equivalent to $(-a_{i1})x_1 + (-a_{i2})x_2 + \ldots + (-a_{in})x_n \leq -b_i$, and any equality constraint $\sum a_{ij}x_j = b_i$ can be replaced by $\sum a_{ij}x_j \leq b_i$ and $\sum a_{ij}x_j \geq b_i$. Any unrestricted variable x_i can be represented as the difference $x_i = x_i^+ - x_i^-$ of two nonnegative variables x_i^+, x_i^-. In addition, the problem of minimizing of a function z is equivalent to the problem of maximizing of function $-z$.

For convenience we introduce the following notation: vector $c \in \mathbb{R}^n$ represents the coefficients in the objective function $c = (c_1, \ldots, c_n)$, vector $b \in \mathbb{R}^m$ represents the right-hand coefficients $b = (b_1, \ldots, b_m)$ in the inequalities $\mathcal{C}$. A vector $x \in \mathbb{R}^n$ is the decision vector $(x_1, \ldots x_n)$ and a matrix $A \in \mathbb{R}^{m\times n}$ stores the variable coefficients from $\mathcal{C}$. (A coefficient $a_{ij} \in A$ is multiplied with the variable x_j in the inequality C_i.) The problem of maximizing the objective function under the given constraints can now be written as:

$$\max c^T x, \text{subject to } Ax \leq b, x \geq 0 \tag{2}$$

(where c^T is the transpose of the $n \times 1$ matrix c).

For a set of constraints $\mathcal{C}$ (which we sometimes refer to as the *model* of the LP problem) and any linear function $d = d_0 + d_1x_1 + \ldots + d_nx_n$ ($d_i \in \mathbb{R}$) we write LP(max, d, $\mathcal{C}$) and LP(min, d, $\mathcal{C}$) to denote the results of maximizing/minimizing of the function d. Therefore, the optimization of the objective function $c^T x$ can be written as:

LP(max, $c^T x$, $\mathcal{C}$) for the result of: max $c^T x$, subject to $Ax \leq b, x \geq 0$
LP(min, $c^T x$, $\mathcal{C}$) for the result of: min $c^T x$, subject to $Ax \leq b, x \geq 0$

Given a sequence of user assignments $\rho = [x_{i_1} = v_{i_1}, \ldots, x_{i_k} = v_{i_k}]$, we write $\mathcal{C}^\rho$ for the set of constraints $\{C_1^\rho, \ldots C_m^\rho\}$, where each constraint $C_j : a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jn}x_n \leq b_j$ has been transformed to C_j^ρ by assigning the variables from ρ and moving the resulting constants $a_{ji_1}v_{i_1}, \ldots, a_{ji_k}v_{i_k}$ to the right side of the inequality. Coefficients relating to variables in $\mathcal{X} \setminus \{x_{i_1}, \ldots, x_{i_k}\}$ remain unchanged while the right-hand constant b_j^ρ becomes $b_j^\rho = b_j - a_{ji_1}v_{i_1} - \ldots - a_{ji_k}v_{i_k}$.

We also write $A^\rho \in \mathbb{R}^{m\times(n-k)}$, $x^\rho \in \mathbb{R}^{m-k}$, and $b^\rho \in \mathbb{R}^m$ for the corresponding matrices and vectors, i.e., x^ρ is the vector of the unassigned variables while b^ρ is the vector of the modified right-hand constants b_j^ρ. We write $c^\rho \in \mathbb{R}^{(n-k)}$ for the vector of coefficients standing with remaining unassigned variables. However, a new constant $c_0^\rho = c_{ji_1}v_{i_1} + \ldots + c_{ji_k}v_{i_k}$ has emerged and the new objective function is: $c_0^\rho + (c^\rho)^T x^\rho$

LP problems have an interesting geometric interpretation. Linear inequalities $\mathcal{C}$ describe a convex polytope which bounds a *feasible region* $S(\mathcal{C})$ (corresponding to configuration solution space, also denoted as $S(\mathcal{C})$). If the polytope is *nonempty* (i.e. there is at least one solution satisfying all the constraints) and if the polytope is *bounded* (i.e. the objective function cannot have an arbitrarily large maximum) then any optimal value is obtained at a *vertex* of the polytope [16].

We distinguish between the two representations of a polytope - a *halfspace representation* (a polytope is the intersection of a finite number of halfspaces) and a *vertex representation* (a polytope is the convex combination of a finite number of vertices). The LP model $\mathcal{C}$ is the halfspace representation.

The convexity property of the polytopes guarantees that the valid domains V_j are always of the interval form $[l_j, u_j]$, i.e. that all values between the minimum and maximum value for a variable x_j belong to the set of valid values for this variable (Fig. 2).

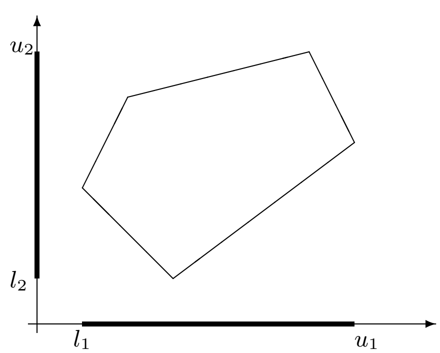

Fig. 2. Valid domains V_j are always of the interval form $[l_j, u_j]$ since the polytopes are convex geometrical figures. In the plane, the polytope is actually a polygon.

The valid domains can be found as follows:

$$V_j = [l_j, u_j], \text{where } l_j = LP(min, x_j, \mathcal{C}), u_j = LP(max, x_j, \mathcal{C}), \tag{3}$$

In addition, when the sequence of user assignments ρ is given, we denote the valid domains under these assignments as $V_j^\rho = [l_j^\rho, u_j^\rho]$, i.e.

$$V_j^\rho = [l_j^\rho, u_j^\rho], \text{where } l_j^\rho = LP(min, x_j, \mathcal{C}^\rho), u_j^\rho = LP(max, x_j, \mathcal{C}^\rho), \quad (4)$$

The *simplex algorithm* (Dantzig, 1947) solves the LP problem by *pivoting*, i.e. walking down the edges of a polytope. So far, all variants of the simplex algorithm are in the worst case exponential. However, empirical observations indicate the remarkable fact that the "simplex method typically requires at most $2m$ to $3m$ pivots to attain optimality" [21]. Some LP-solving packages [15] report $m + n$ empirically observed complexity.

The *ellipsoid algorithm* [12] was the first LP solving algorithm with worst-case *polynomial complexity.* Although in practice performing worse than simplex, it is of great theoretical importance, proving that the LP problem is of polynomial complexity. The *interior-point method* [11] is an algorithm that obtains both a worst-case polynomial bound and performs well in practice (comparably to the simplex algorithm).

4 Interactive Configuration in LP-problems

In this chapter we will illustrate how the concept of interactive configuration could be used in classical LP problems.

The Diet Problem

The diet problem was first considered in the US Army for constructing the cheapest yearly diet plan for soldiers on the field, that would minimize the cost while satisfying all nutritional needs. It was one of the first problems used to test the simplex algorithm [5].

Example 5. Consider a choice of n foods and m nutrients. The quantity of i-th nutrient in a unit of the j-th food is denoted by a_{ij}. The unit price of j-th food is c_j $(i = 1, \ldots, m, j = 1, \ldots, n)$. Let x_j $(j = 1, \ldots, n)$ denote the yearly consumption of each food and the max_i, min_i $(i = 1, \ldots, m)$ the maximum and minimum yearly requirements of the i-th nutrition. Then the LP formulation of the diet problem can be expressed as:

$$\begin{array}{ll}
\text{minimize} & \sum c_j x_j \\
\text{subject to:} & \sum a_{ij} x_j \geq min_i \\
 & \sum a_{ij} x_j \leq max_i \\
 & x_j \geq 0 \\
 & (i = 1, \ldots, m, j = 1, \ldots, n)
\end{array}$$

The Interactive Diet Problem

This form of the diet problem is inadequate for delivering personalized diet plans. It provides a single, fixed diet plan, which is unlikely to be used in our daily lives. For example, we might want to specify that we require (at least) 1kg of chocolate per year, and get the feedback on what are valid choices for other food quantities, that would not violate restrictions on maximum sugar consumption. We might also want to limit the maximum price we want to pay, and check the effects on the maximum amount of chocolate we can now include. A system delivering *interactive configuration functionalities* could help in making a cost efficient and pleasant diet plan.

In a simple scenario, the system first precalculates valid domains $[l_j, u_j]$ for each variable x_j. Then the sequence of interactions starts with a user at each step assigning a value to a specific variable $x_j = v_j, v_j \in [l_j, u_j]$. In response, the system calculates the minimum/maximum quantities for other foods $[l_i, u_i](i \neq j)$, i.e. calculates valid domains respecting the completeness and validity requirement. This user-configurator interaction stops when the user has assigned values to all variables. Alternatively, the user may decide to stop the assignment process before, leaving certain variables unassigned and letting the configurator find the optimum assignment to the remaining undecided variables. In Fig. 3 we show how this process might look from the user perspective.

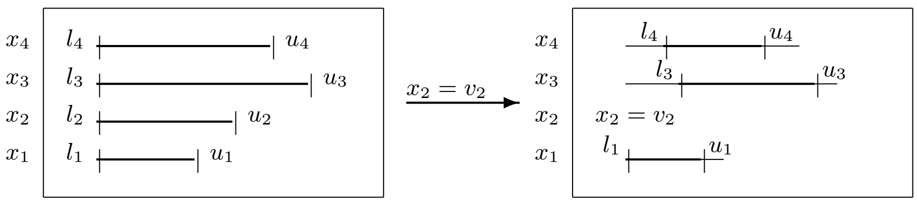

Fig. 3. Interactive LP configuration. After the user has assigned $x_2 = v_2$ the domains $[l_j, u_j]$ are recalculated. They will always become smaller, or at least no larger than before.

The diet example illustrates how the concept of interactive configuration can be used to increase the usability of existing systems in solving LP problems. Instead of each time solving a separate LP problem tailored for a specific user, the system should make it easy for a user to interactively explore the entire solution space, thus effectively accommodating different users and their preferences.

5 LP Implementation of Configuration Functionalities

5.1 Naive Algorithm for DC

Since the convexity property of the polytopes guarantees that the valid domains V_j always have the form $[l_j, u_j]$, it is sufficient to find a maximum value u_j and a minimum value l_j in order to calculate V_j. This leads to the algorithm for calculating valid domains for the undecided variables under the given model $\mathcal{C}$ and a sequence of user assignments ρ, presented in Fig. 4.

CVD($\mathcal{C}, \rho$)
1: FOR EACH $x_j \notin dom(\rho)$
2: $l_j^\rho = LP(min, x_j, \mathcal{C}^\rho)$
3: $u_j^\rho = LP(min, x_j, \mathcal{C}^\rho)$
4: IF $l_j^\rho = u_j^\rho$ THEN $\rho := \rho \dagger [x_j = l_j^\rho]$

Fig. 4. Calculating Valid Domains. The notation $\rho \dagger [x_i = v_i]$ denotes appending the element $[x_i = v_i]$ to the list ρ.

The entire interaction is modelled by the algorithm presented in Fig. 5.

Theorem 6. *The DC algorithm has worst-case polynomial complexity.*

Proof. In each step of the user-configurator interaction DC solves $2 \cdot (n - k)$ ($k = |\rho|, n = |\mathcal{X}|$) LP problems (line 8), two for each unassigned variable. Since LP-problems are polynomially hard, so is the entire interaction as it solves at most a linear number of polynomially hard problems.

DC($\mathcal{C}$)
1: FOR EACH $x_j \in \mathcal{X}$
2: $\quad l_j = LP(min, x_j, \mathcal{C})$
3: $\quad u_j = LP(max, x_j, \mathcal{C})$
4: sufficiently_specified := FALSE, $\rho := [\ ]$
5: WHILE sufficiently_specified = FALSE
6: $\quad$ USER_CHOICE (x_i, v_i) where $(x_i \notin dom(\rho), v_i \in V_i^{\rho})$
7: $\quad \rho' := \rho \dagger [x_i = v_i]$
8: $\quad$ CVD$(\mathcal{C}, \rho')$
9: $\quad \rho := \rho'$
10: $\quad$ IF $dom(\rho') = \mathcal{X}$ OR user wants to end THEN
11: $\qquad$ sufficiently_specified := TRUE

Fig. 5. The Domain Calculation Algorithm. USER_CHOICE(x_i, v_i) in line 6, denotes the user choosing an assignment $x_i = v_i$.

However, if we chose to use the simplex implementation of the LP solver, then although we do not get any polynomial guarantees for the DC functionality, we can expect a very fast running time in practice. Each call to DC has an expected empirical performance (using the MINOS package, [15]) of $2(n-k)\cdot(m+n-k)$ pivoting steps, (k - number of already assigned variables, $n = |\mathcal{X}|$, $m = |\mathcal{C}|$) which is of low-degree polynomial complexity.

5.2 An Improved Simplex Algorithm for DC

LP problems guarantee that all local optimums are global optimums. Therefore, the simplex algorithm is sure to have reached the global optimum as soon as no improvements to the objective function are possible. The basic idea behind the improved algorithm to be presented is in the greedy nature of simplex. If we are maximizing a variable x_j simplex is in each step exploring a vertex of the polytope with an increased x_j coordinate. The intuition is to start the simplex search from the vertex with the highest x_j coordinate. For this, we can take advantage of the fact that we are solving many highly related LP problems. We maintain the list $\mathcal{V}$ that for every variable x_j stores 2 vertices $\mathcal{V}(x_j, min), \mathcal{V}(x_j, max)$ with a minimal/maximal j coordinate encountered so far. We refer to these vertices as *extreme points*. Since in every LP call the simplex visits a number of intermediate nodes, we update the structure $\mathcal{V}$ every time an intermediate node has a maximal/minimal coordinate for some variable x_j. Every subsequent simplex call to maximize/minimize x_j starts from the appropriate extreme point $\mathcal{V}(x_j, max)$ / $\mathcal{V}(x_j, min)$. This guarantees that no intermediate node will be visited twice.

We write $LP_{\boldsymbol{v}}(min, x_j, \mathcal{C})$, $LP_{\boldsymbol{v}}(max, x_j, \mathcal{C})$, for the results of the simplex algorithms starting at vertex $\boldsymbol{v}$. The improved algorithm differs from the DC algorithm (page 8) by the way the l_j, u_j are calculated (line 8):

ImprovedCVD$(\mathcal{C}, \rho, \mathcal{V})$
FOR EACH $x_j \notin dom(\rho)$
$\quad \boldsymbol{v} = \mathcal{V}(x_j, min)$, $l_j = LP_{\boldsymbol{v}}(min, x_j, \mathcal{C})$ (update $\mathcal{V}$)
$\quad \boldsymbol{v} = \mathcal{V}(x_j, max)$, $u_j = LP_{\boldsymbol{v}}(max, x_j, \mathcal{C})$ (update $\mathcal{V}$)

Fig. 6. The improved version of the CVD algorithm. A list of "good" vertices $\mathcal{V}$ is used to select the starting vertex in the simplex iteration.

We consider this only as the first step towards more efficient DC algorithms. One could think of other, more involved ways to quickly reach the "good" starting vertex $\boldsymbol{v}$, with a maximal/minimal coordinate. However, the bad theoretical guarantee for the simplex makes it hard to give stronger theoretical guarantees for such an algorithm.

5.3 Restoration and Assisted Conflict Resolution

A user's decision to undo the assignment to a variable is called *restoration.* We have already described a way to implement this functionality in section 2. The algorithm of Fig. 7 follows that description:

`Restoration:`
1: `USER_CHOICE` (x_{i_r}) $(x_{i_r} \in dom(\rho))$
2: $\rho' = \rho \backslash [x_{i_r} = v_{i_r}]$
3: `FOR EACH` $x_j \notin dom(\rho')$
4: $\quad l_j = LP(min, x_j, \mathcal{C}^{\rho'})$, $u_j = LP(max, x_j, \mathcal{C}^{\rho'})$

Fig. 7. Restoration algorithm for unassigning user choices. $\rho \backslash [x_{i_r} = v_{i_r}]$ denotes removing the element $x_{i_r} = v_{i_r}$ from the list ρ.

It is easy to see that this is a worst-case polynomial algorithm, since we can use the worst-case polynomial version for LP-solvers in line 4.

If the user enforces a conflicting assignment, the system should offer a list of choices that need to be changed in order to restore consistency while keeping the last assignment. This functionality is called *assisted conflict resolution.* As suggested in section 2, one way to implement assisted conflict resolution is to generate a list of conflicting assignments $\rho_r \subseteq \rho$ that needs to be removed.

A simple way to implement it is to move the conflicting assignment $x_k = v_k$ to the beginning of the user assignment list, and apply the rest of the original assignments in ρ until a conflict is reached. We remove conflicting assignments and continue until we reach the last assignment $x_k = v_k$. At the end, after calculating valid domains with a reduced user assignment list $V_j^{\rho_{new}}$, we will restore consistency (Fig. 8).

`Assisted Conflict Resolution:`
1: `USER_CHOICE` (x_c, v_c) $(v_c \notin V_c^{\rho})$
2: $\rho_{new} := [x_c = v_c]$
3: `FOR` j = i_1 `to` i_k $([x_j = v_j] \in \rho)$
4: $\quad l_j = LP(min, x_j, \mathcal{C}^{\rho_{new}})$, $u_j = LP(max, x_j, \mathcal{C}^{\rho_{new}})$
5: $\quad$ `IF` $v_j \in [l_j, u_j]$ `THEN` $\rho_{new} := \rho_{new} \dagger [x_j = v_j]$

Fig. 8. Assisted conflict resolution algorithm.

The total number of LP calls (in line 4) is $2k$, where $k = |\rho|$. Obviously, this algorithm has a worst-case polynomial bound. The list of choices that has to be removed $\rho \backslash \rho_{new}$ is *minimal* in the sense that the list of reduced assignments ρ_{new} cannot be extended with any assignment from ρ without enforcing a conflict. Of course, this does not have to be the *minimum* among all possible divisions $\rho \backslash \rho_{new}$.

Theorem 7. *Given the sequence of user assignments ρ, and the user assigned variable $x_j = v_j$, the Restoration algorithm unassigns the variable and correctly recalculates the valid domains. Given the sequence of user assignments ρ, and the* *conflicting user assignment $x_c = v_c$ ($v_c \notin V_c^\rho$) the Assisted Conflict Resolution algorithm correctly calculates the list of assignments that has to be removed in order to restore consistency. Restoration and Assisted Conflict Resolution have polynomial worst-case complexity.*

6 LP Specific Configuration Functionalities

When all the constraints are linear inequalities, the interactive configuration becomes an easy (polynomial) problem. Therefore, we are able to provide more functionalities than in the combinatorial case. The convexity property enables an efficient manipulation of the solution space based on maximum/minimum values for a specific variable while the geometric interpretation facilitates new ways for a user to perceive and explore the solution space.

6.1 Domain Restriction

Providing *interactive configuration functionalities* to the diet problem can be additionally improved by allowing a user not just to assign a value to a variable x_j, but also to restrict the existing domain $V_j = [l_j, u_j]$ to $[l'_j, u'_j]$, where $l_j \le l'_j \le u'_j \le u_j$.

The other domains V_i $(i \neq j)$ could be calculated by adding constraints $l'_j \le x_j \le u'_j$ to the existing model $\mathcal{C}$ and recalculating the valid domains.

```
Naive Domain Restriction
1: USER_CHOICE (x_j, l'_j, u'_j), (l'_j, u'_j ∈ [l_j, u_j], l'_j ≤ u'_j)
2: C' = C ⋃ (l'_j ≤ x_j ≤ u'_j)
3: FOR EACH x_i ≠ x_j
4:         l_i = LP(min, x_i, C') , u_i = LP(max, x_i, C')
```

Fig. 9.

However, we could do better by noting that in the recalculated domains V_i $(i \neq j)$ the extreme points $\mathcal{V}(x_i, min)$, $\mathcal{V}(x_i, max)$ that do not violate $l'_j \le x_j \le u'_j$, remain extreme points. Therefore, by using the same structure $\mathcal{V}$ introduced in the improved version of the DC algorithm (page 8), we can identify which extreme points have been violated (i.e. which are not in the feasible region any more), and perform an LP calculation only for those points.

Even more, the new extreme points are *always* found in the defining hyperplanes: $x_j = l'_j$ and $x_j = u'_j$. Therefore, for the set of all the extreme points violating inequality $l'_j \le x_j$ (denoted as $\mathcal{V}_{<l'_j}$) we perform optimization in the hyperplane $x_j = l'_j$. Similarly we perform optimization in the hyperplane $x_j = u'_j$ for $\mathcal{V}_{>u'_j}$. This leads to the algorithm in Fig. 10.

Manipulation With the Objective Function

A user could ask what values can the variables have if he requires that the cost of the final solution is within some fixed limits $[L, U]$. This question can be answered by adding a constraint $L \le c_1x_1 + \ldots + c_nx_n \le U$, to the model and manipulating it like any other domain restriction constraint. Actually, from the user's point of view, a new variable $z = c_1x_1 + \ldots + c_nx_n$ can be introduced, with its valid domain $V_z = [L, U]$.

Given the set of assignments ρ in the DC algorithm, V_z can be calculated by 2 LP calls: $U = LP(max, c_0^\rho + (c^\rho)^T x^\rho, \mathcal{C}^\rho)$ and $L = LP(min, c_0^\rho + (c^\rho)^T x^\rho, \mathcal{C}^\rho)$.

```
Improved Domain Restriction
USER_CHOICE (x_j, l'_j, u'_j), (l'_j, u'_j ∈ [l_j, u_j], l'_j ≤ u'_j)
ρ := [x_j = l'_j]
FOR EACH v ∈ V_{<l'_j}
      IF v = V(x_i, min) THEN l'_i = LP(min, x_i, C^ρ) (update V)
      ELSE (v = V(x_i, max)) u'_i = LP(max, x_i, C^ρ) (update V)
ρ := [x_j = u'_j]
FOR EACH v ∈ V_{>u'_j}
      IF v = V(x_i, min) THEN l'_i = LP(min, x_i, C^ρ) (update V)
      ELSE (v = V(x_i, max)) u'_i = LP(max, x_i, C^ρ) (update V)
```

Fig. 10.

In addition, when restricting the domain $V_z = [L, U]$ to $[L', U'] \subseteq [L, U]$, we can calculate domains $[l_i, u_i]$ by updating the model $\mathcal{C}' = \mathcal{C} \bigcup \{L' \leq c_1x_1 + \ldots + c_nx_n \leq U'\}$ and calculating $l_i = LP(min, x_i, \mathcal{C}')$, $u_i = LP(max, x_i, \mathcal{C}')$.

In the diet example, this means we can now interactively reduce the maximum price we are willing to pay for the food supplies, and explore how it effects the available choices.

6.2 Two-Dimensional Configuration

We can additionally take advantage of the geometric interpretation of the solution space. Namely, the valid intervals V_j can be seen as a 1-dimensional projection of the polytope $S(\mathcal{C})$ to the x_j axis. We want to extend this projection to a two-dimensional (x_i, x_j) plane. The projected polygon is convex and the vertices of the polygon are the projections of the polytope vertices. So, the resulting figure has nice geometric properties and this could help a user get a better insight in the relationship between the two parameters, and better explore the solution space. Even more, this is the special case of the general polytope projection (from $\mathcal{X}$ to the subset $\{x_{i_1}, \ldots, x_{i_k}\}$), which is a well explored problem with well established solving methods [1, 10].

To compute the projection, we need either the edge equations (halfspace representation) or the vertex coordinates (vertex representation). Although the general projection methods (block elimination, vertex based approaches) are usually fine tuned for only one kind of representation, we can choose any of them since in our case (k=2) both representations have the same complexity (i.e. the number of vertices is equal to the number of edges). In particular, in [10] the authors present an algorithm with linear complexity in the number of facets of the projection (for a constant size of a polytope).

7 Conclusion

We have shown how to use techniques from linear programming in interactive configuration of solution spaces described by the set of linear inequalities. The polynomial time algorithms for linear programming provide polynomial time algorithms for interactive configuration. Moreover, we have shown how to provide a polynomial time interactive dialog with the classical LP problems in order to provide a new way of finding a solution that is not only optimal but also meets some needs of a user that are not expressed in the linear inequalities.

References

1. Amenta, N., and Ziegler, M. G.: Shadows and slices of polytopes. Proceedings of the twelfth annual symposium on computational geometry (1996), 10-19

2. Bachant, J., and Soloway, E.: The Engineering of XCON. Communications of the ACM (1989) 32:311–317
3. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers (1986) 35:677–691
4. Chan, W., Anderson, R., Beame, P., and Notkin, D.: Combining Constraint Solving and Symbolic Model Checking for a Class of Systems with Non-linear Constraints. Computer Aided Verification Conference Proceedings (1997) LNCS 1254, 316-327
5. Dantzig, G. B.: Linear Programming and Extensions. Princeton University Press (1963)
6. Dechter, R., and Pearl, J.: Tree Clustering for Constraint Networks. Artificial Intelligence (1989) 38:353–366
7. Freuder, E. C., Carchrae, T., and Beck, J.C.: Satisfaction Guaranteed. IJCA Workshop on Configuration (2003), Eighteenth International Joint Conference on Artificial Intelligence
8. Hadzic, T., Subbarayan, S., Jensen, R. M., Andersen, H. R., Møller, J., and Hulgaard, H.: Fast backtrack-free product configuration using a precompiled solution space representation. Proceedings of the International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems (2004), DTU-tryk 131138
9. Hooker, J. N.: A Hybrid Method for Planning and Scheduling. CP 2004 Conference Proceedings (2004), LNCS 3258, 305–316
10. Jones, C. N., Kerrigan, E. C., and Maciejowski, J. M.: Equality Set Projection: A new algorithm for the projection of polytopes in halfspace representation. University of Cambridge, Technical Report (2004) CUED/F-INFENG/TR.463
11. Karmarkar N.: A new polynomial-time algorithm for linear programming. Combinatorica (1984) 4:373-396
12. Khachiyan, L. G.: A polynomial algorithm in linear programming. Doklady Akademiia Nauk SSR 244 (1979), 1093-1096
13. Leahu, L., and Gomes C. P.: Quality of LP-Based Approximations for Highly Combinatorial Problems. CP 2004 Conference Proceedings (2004), LNCS 3258, 377–392
14. Møller B. J.: Symbolic Model Checking of Real-Time Systems using Difference Decision Diagrams. PhD thesis (2002), IT-C DS: D-2002-001
15. Neculai, A.: On the complexity of MINOS package for linear programming. Center for Advanced Modeling and Optimization (2003)
16. Papadimitriou, C. H., and Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice Hall (1998)
17. Pargamin, B.: Vehicle Sales Configuration: the Cluster Tree Approach. ECAI 2002 Configuration Workshop (2002) 35–40
18. Sabin D., and Weigel R.: Product Configuration Frameworks - A Survey. IEEE Intelligent Systems (1998) 13:42–49
19. Sinz C., Kaiser A., and Kuchlin, W.: Formal methods for the validation of automotive product configuration data. Artificial Intelligence for Engineering Design (2002) 27:75–97
20. Subbarayan, S., Jensen, R. M., Hadzic, T., Andersen, H. R., Hulgaard, H., and Møller, J.: Comparing Two Implementations of a Complete and Backtrack-Free Interactive Configurator. CP 2004 Workshop on CSP Techniques with Immediate Application (2004)
21. Todd, J. M.: The many facets of linear programming. Mathematical Programming (2002) 91:417–436
22. Vanderbei, R. J.: Linear Programming: Foundations and Extensions. 2nd ed., (2001) Springer, International Series in Operations Research and Management Science, Vol. 37