Automated Design of Elevator Systems: Experimenting with Constraint-Based Approaches

Stefano Demarchi, Marco Menapace, Armando Tacchella

Università degli Studi di Genova
stefano.demarchi@edu.unige.it, marco.menapace@edu.unige.it,
armando.tacchella@unige.it

Abstract. System configuration and design is a well-established topic in AI. While many successful applications exists, there are still areas of manufacturing where AI techniques find little or no application. We focus on one such area, namely building and installation of elevator systems, for which we are developing an automated design and configuration tool. The questions that we address in this paper are: (*i*) What are the best ways to encode some subtasks of elevator design into constraint-based representations? (*ii*) What are the best tools available to solve the encodings? We contribute an empirical analysis to address these questions in our domain of interest, as well as the complete set of benchmarks to foster further research.

1 Introduction

Context. The problem of automating product configuration and design has a long history in diverse application fields. In the late 1970s, possibly the first automated configuration program was developed at Digital Equipment Corporation to build computer systems meeting custom requirements [16]. Later on, approaches based on constraint programming [17, 18] and extensions tackling a variable number of constraints [21, 25] were presented. A relatively recent survey [26] mentions various product configuration tools and methods that have been reported in the literature. In spite of many success stories, product configuration and design remains still a manual endeavor in many, if not most, manufacturing areas. We argue that building and installation of elevator systems is one such area. Excluding two recent contributions of ours [2, 11], to the best of our knowledge the only paper that tackles configuration of elevator systems dates back to [14].

Motivations. We are developing the tool LiftCreate to design elevator systems starting from a database of commercial components and some basic data, e.g., hoistway size, number of floors, and payload. LiftCreate guarantees feasibility of the design according to 2014/33/EU regulation within the framework of EN 81-20/81-50 norms. LiftCreate is built around special purpose heuristics to handle most design subtasks, including choice and fitting of components and optimization of the design. Since LiftCreate is deployed as a web application, hard-coding of search and optimization subtasks allowed us to achieve the levels of efficiency and predictability required, but at the expense of flexibility as additions or modifications are now very time-consuming. This is why in previous works we started experimenting with Optimization Modulo Theories (OMT) solvers [23].In our first attempts we focused on OMT techniques only, and we did not investigate different ways to encode elevator configuration as a constraint satisfaction problem.

Research questions. Our first and foremost question is how to choose among different alternatives to encode design subtasks into sets of constraints. In particular, we consider alternative ways to (*i*) encode the choice of components, (*ii*) represent relevant quantities, (*iii*) encode functions and (*iv*) handle multiple cost functions. The second question relates to the constraint-based tools and techniques of choice. As shown in the literature, see, e.g., [3, 7, 10], there is some debate over using constraint-based technologies other than OMT solvers to tackle encodings involving a mixture of arithmetic constraints and cost functions. For this reason, here we compare the performances of two encodings, namely in the SMT-LIB [5] language and in the MiniZinc [1] language. We feed our SMT-LIB encoding to two OMT solvers, i.e., OptiMathSat [24] and z3 [19], and our MiniZinc encoding to a pool of constraint solvers, namely Chuffed [9], OR-Tools [20], ECL^{i}PS^e [22], CPLEX [13] and Gurobi [12]. The ultimate goal is to understand how different encoding choices impact on different solvers in our context, but also to evaluate the best combination for LiftCreate.

Contribution. Our experiments involve a number of design scenarios characterized by different hoistway sizes and two different design targets: *baseline* and *full*. In the former we select, place and fit only two components, namely the car frame — the structure that supports the elevator car — and the doors. The full encoding includes also sizing of the hydraulic cylinder pushing the car frame, and minimization of the forces on the car frame rails. We provide empirical evidence that all the selected solvers can deal with the baseline integer-based encodings; on the other hand, the full encoding involves arithmetic over reals and thus only OMT solvers consistently provide results within the time limit of five CPU minutes. Most solvers show comparable performances on the baseline encoding, but only Chuffed seems to have an edge over the other solvers in this group, with only one exception on a scenario for which no feasible configuration exists; z3 is generally faster than OptiMathSat, but on integer-based encodings z3 must yield to Chuffed, albeit not substantially. Finally, we show that special purpose heuristics can still be faster than most combinations of solvers and encodings in the constraint-based approach; however, considering the gain in flexibility, some combinations of solvers and encodings are competitive enough to invest further development effort in them. Given our results, we believe that also our benchmarks make for a valuable contribution and we made them available at

`https://github.com/sdemarch/AIxIA\_2021\_Benchmarks`

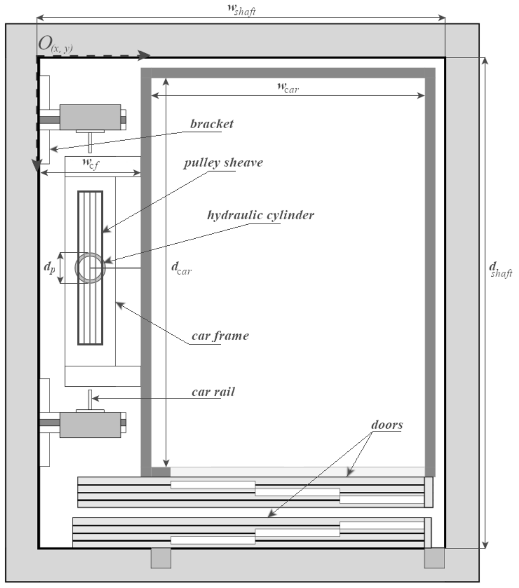

Fig. 1. Plan view of a configured RHE. The shaft is the gray box surrounding the other components, the car frame is on the left side of the drawing and the doors are at the bottom.

2 Design of Elevator Systems

In this paper we focus on Roped Hydraulic Elevators (RHEs) in which the lifting power is provided by hydraulics. In Figure 1 we show the cross-section of a RHE with its main components to be described briefly in the following. The shaft (hoistway) is the outermost grey-filled rectangle: although it is not part of the elevator system per se, the shaft depth (d_{shaft}) and width (w_{shaft}) constrain the space available for installing the elevator. The main component of the elevator is the car for people transportation, represented as a rectangle with dark-grey borders (walls). The car is characterized by a depth (d_{car}) and a width (w_{car}). On the left side of Figure 1, we can see the *car frame*, i.e., the structure that supports the car and connects it to the *hydraulic cylinder* providing the lifting power. The hydraulic cylinder is placed inside the car frame structure and it is depicted as two concentric circles below the *pulley sheave*. In the following, we consider the width of the car frame, denoted as w_{cf}, and the hydraulic cylinder barrel diameter, denoted as d_p. The car frame is fixed to the shaft walls via T-shaped *brackets* to support the *car rails* on which the car frame core gear slides. At the bottom of Figure 1, we can see the doors, namely the *car door* and the *landing door* — three-paneled sliding doors in this case. The car door is only one and it is attached to the car, while there is one matching landing door for each floor. For each component we identify a *base point*, i.e., the insertion point of the component in the drawing, denoted as (x_{comp}, y_{comp}). Each component base point is found at the topmost left angle of the rectangle which envelopes the component structure, and we identify by convention the coordinate system origin with the shaft base point, shown in Figure 1 as $\mathcal{O}_{x,y}$. All the named quantities are mapped with *decision variables* and *parameters*, and represent lengths measured in millimeters. As for the domain of such quantities, parameters are constrained to take values from the database instance where components are saved. In principle, each parameter can take any value from a set of discrete quantities depending on the choice of the component, and these numbers are treated as constants in the final encoding. On the other hand, decision variables are always positive and limited from above by the dimensions of the shaft.

We focus on the selection, placement and sizing of three components: the car frame, the car doors and the hydraulic cylinder. The choice of the doors and the car frame impacts on their placement, since different components have different sizes and parameters, and placement must be determined considering shaft size, encumbrances, and tolerances. In turn, the placement of the doors and the car frame has an impact on the forces exerted on the car rails, as well as the choice and verification of the hydraulic cylinder, determined by the compatibility with the car frame and the overall suspended weight. Our approach is to split the encoding of elevator design problems in two parts: a set of (hard) constraints to encode geometrical and dynamical aspects required to obtain *feasible* configurations, and a set of soft constraints, modeled as cost functions, to drive the search towards feasible configurations that are also *desirable* according to best practices in elevator design such as minimizing costs by choosing the smallest components that make the design feasible, while, at the same time, utilizing all the space available.

Considering geometrical and dynamical constraints first, we can observe that choosing and placing the car frame must take into account two main issues. First, given the shape of the brackets, it is not possible to model the car frame as a simple rectangle in order to fit it with the other components. Therefore the placement of the car frame is computed by subtracting residuals from the total shaft depth. Second, the placement of the car frame must take into account its maximum overhang, i.e., the car cannot "lean" too much outside the car frame core gear. The constraints to place the car-landing door pair should guarantee that both structures fit the shaft, that the actual opening fits the car and that the landing door frame does not exceed the shaft size. All these constraints are expressed through linear inequalities. In order to avoid collision of moving parts, there are two-dimensional non-overlapping constraints between the car door and both the car frame and the brackets, expressed with disjunctive constraints. Concerning the hydraulic cylinder, there are two main constraints to be respected. The first one is relative to the compatibility between the car frame and the cylinder, expressed through a car frame parameter which indicates the maximum diameter of a cylinder that can be inserted. The second constraint requires that the force on the piston, due to the overall suspended weight, does not exceed the critical force, i.e., the force which can make steel collapse, which also depends on the choice of car frame. The details related to geometrical and dynamical constraints corresponding to the informal description above are reported in [11]

To shape the cost functions, we consider four design objectives. The first is that the car frame should be aligned as much as possible to the center of the car on the y axis — i.e., considering Figure 1, the car frame should appear vertically centered with respect to the car. The second objective regards doors positioning: in case of symmetric doors, i.e., the opening midpoint coincides with the door frame midpoint, the opening of the car door should be aligned as much as possible to the center of the car on the x axis; in case of asymmetric doors, then the opening should be as close as possible to the opposite side of the car frame in order to minimize the chance of interference. Notice that in Figure 1 we have a non-symmetric door, which is aligned to the right car wall. As third objective we have that the car frame and the hydraulic cylinder should not be over-sized, and the opening of the car door should not be undersized. Finally, the fourth objective is that the force exerted on the car rails should be minimized.

3 Encoding Elevator Systems Design

Component selection The car frame, the cylinder and the doors are selected from a database of components. In order to automate the design of an elevator, we must consider that choosing different components yields different parameter values for each one. The relationship between the selection of a component and the assignment of the corresponding parameter values can be encoded via Boolean implications of the form

$$Id_x = i \Rightarrow x.p = v \tag{1}$$

where Id_x encodes the identifier of choice for component x (a decision variable), i is a specific identifier value, $x.p$ is some parameter of the component x and v is the value of $x.p$ given that the component x with identifier i was chosen — see, e.g., [4]. To encode constraints of the form (1) a combination of Boolean reasoning with integer arithmetic is sufficient. However, considering the way data sets are usually encoded in MiniZinc with arrays [15], we consider an alternative encoding where we associate an array to each component parameter. For example, if a component x has two parameters p_1 and p_2, we build two arrays P_1 and P_2 that will store the values of p_1 and p_2 for each instance of the component. The index of the arrays becomes a decision variable chosen by the solver to enforce the correct values of the parameters.

Look-up tables Some parameters, e.g., the maximum number of passengers that the car may accommodate, are a function of others, e.g., the car surface. However, instead of expressing such constraints directly — which might involve the use of non-linear or transcendental functions — the correspondence between free parameters and derived ones is encoded with *look-up tables.* Table 1 exemplifies such a table assuming that the car surface A is contained within three ranges. The car payload is computed in a similar way, but, since the surface ranges are different, we need another set of constraints structured in the same way.

Surface (A)	**Passengers** (P)
$a1 < A \leq a2$	P_0
$a2 < A \leq a3$	$P_0 + 1$
$A > a3$	$P_0 + 2$

Table 1. Look-up table to encode the number of passengers P with starting value P_0 as a function of the car surface A.

These requirements can be easily modeled with implications in the same way as component selection: the surface A is a decision variable that implies the number of passengers or the payload. However, both SMT-LIB and MiniZinc allow users to define custom *functions*. In practice, functions are series of *if-then-else* statements about, e.g., the car surface, where each function returns, e.g., the corresponding number of passengers or the payload.

Integers vs. Reals Most parameters involved in the design process are expressed in millimeters which suggests integer-based encodings. However, some parameters like the forces exerted on the car rails involve arithmetic over reals. This makes the corresponding constraint satisfaction problems members of the mixed-integer arithmetic family. In such encodings, the main disadvantage is that a large number of integer quantities may increase considerably the solution time. We try to improve on this by relaxing some of the integer quantities to reals. In particular, we consider component parameters since parameters are not *decided* but their value is only assigned based on the choice of a component. This means that the domain of the parameters is a finite set and we can relax the arithmetic encoding without producing invalid results. In this representation the only operation that could add decimal digits is division, but since in our encoding there are only a few such operations, boundary checking can be implemented easily. These considerations do not hold for some decision variables including, but not limited to, the index used to select components. Also, CP solvers like Chuffed are not affected by this choice due to the fact that they do not support floating-point arithmetic.

Single and Multi-objective optimization Here we describe alternative constructions of the cost functions, mentioning the details of the parameters involved when necessary. In particular, we encode the design objectives associating a cost to each one of them. The cost associated to car frame misalignment on the y axis is expressed by the absolute value of the distance between the car frame and the car axes. We define the car frame vertical axis $axisY_{cf}$ as the car frame vertical base point y_{cf} plus half of the distance between the car rails dcr

$$axisY_{cf} = y_{cf} + \frac{dcr}{2}. \tag{2}$$

The vertical car axis $axisY_{car}$ is defined as the car vertical base point y_{car} plus half of the car depth d_{car}:

$$axisY_{car} = y_{car} + \frac{d_{car}}{2}. \tag{3}$$

The difference between the terms (2) and (3) gives us the first contribution to the cost function c_{cf}:

$$c_{cf} = |axisY_{cf} - axisY_{car}| \tag{4}$$

The second objective we consider is related to doors. In this case we define the horizontal car axis $axisX_{car}$ as the horizontal car base point x_{car} plus half of the car width w_{car}:

$$axisX_{car} = x_{car} + \frac{w_{car}}{2} \tag{5}$$

The horizontal door axis $axisX_{door}$ is defined as the horizontal door base coordinate x_{cd} plus the length of its left axis la_{cd}:

$$axisX_{door} = x_{cd} + la_{cd} \tag{6}$$

In the case of symmetric doors, good design practices suggest that $axisX_{door}$ and $axisX_{car}$ should be aligned. In the case of non-symmetric doors, it is preferable to have the door *opening* as close as possible to the side of the car which is opposite to the car frame. In a configuration like the one in Figure 1 we can define the base coordinate of such side as:

$$x_{wall} = x_{car} + w_{car} \tag{7}$$

To take into account the different arrangement of doors, we introduce a binary variable, δ_t, which is assigned to 1 if the current door is a non-symmetric door and to 0 otherwise. We can then summarize the contribution to the cost function as:

$$\begin{aligned} c_{door} = (\ &(1-\delta_t)|axisX_{car} - axisX_{door}| + \\ &\delta_t(x_{wall} - (x_{cd} + la_{cd} + \tfrac{opening}{2}))\ ) \end{aligned} \tag{8}$$

The first contribution of (8) is zero when $\delta_t = 1$, i.e, for non-symmetric doors we try to minimize the distance from the side of the elevator opposite to the car frame, whereas when $\delta_t = 0$ we try to align the door and the car axes. The third objective is related to the selection of the componentes, and gives the guidelines for sizing the car frame, the doors and the cylinder. The maximization of the door *opening* leads to accessible elevators which are always considered a plus, whenever feasible; the minimization of the car frame depth *dcr* and the barrel diameter d_p suggests components which are not over-sized, thus helping to keep costs at bay. These criteria can be translated into one additional contribution to the overall cost function defined as:

$$c_{size} = (dcr + d_p - opening) \tag{9}$$

Finally, the last term to minimize is the sum of F^x_{cr} and F^y_{cr}, i.e., the x and y components of the force exerted on the car rails F_{cr}:

$$c_{cr} = F^x_{cr} + F^y_{cr} \tag{10}$$

The computation of F_{cr} is non-trivial and requires additional equations and parameters that we briefly describe. The components of F_{cr} are obtained as

$$\begin{array}{l} F_{cr}^{x} = k \cdot g \cdot \frac{Q_x(Q+75)+P_x \cdot car_W + cdP_x \cdot cd_W + cf_W \cdot CF_x}{2 \cdot h} \\ F_{cr}^{y} = k \cdot g \cdot \frac{Q_y(Q+75)+P_y \cdot car_W + cdP_y \cdot cd_W}{2 \cdot h} \end{array}$$

where the parameters have the following meaning:

- k is a parameter depending on the kind of safety brakes installed;
- g is the standard acceleration due to gravity;
- Q is the car payload;
- P_x and P_y are the midpoint coordinates of the car;
- Q_x and Q_y are obtained through the equations

$$\begin{array}{l} Q_x = max\{P_x + \frac{w_{car}}{8}, P_x - \frac{w_{car}}{8}\} \\ Q_y = max\{P_y + \frac{d_{car}}{8}, P_y - \frac{d_{car}}{8}\}; \end{array}$$

- car_W is the car weight;
- cdP_x, cdP_y are the coordinates of the center of gravity of the car door;
- cd_w is the car door weight and cf_W is the car frame weight;
- CF_x is a coefficient computed as

$$CF_x = 1.5 \cdot \frac{w_{cf}}{2}$$

  where w_{cf} is the distance from the car frame base point to to the left car wall;
- h is the distance between guide shoes, i.e., the supports which slide on the car rails.

In previous works of ours we consider the weighted sum of the costs c_{cf}, c_{door} and c_{size} to obtain the overall cost function, but the contribution c_{cr} may conflict with the previous ones because the farthest is the door from the car frame, the greater is the force exerted on the car rails. Nevertheless, since the car rails can be chosen once the other components are fitted, this objective can be considered with a lower priority. If we follow a single-objective approach, we can weight the cost c_{cr} significantly less than the other three. The overall cost function $\mathcal{C}$ becomes

$$\mathcal{C} = \alpha_1 c_{cf} + \alpha_2 c_{door} + \alpha_3 c_{size} + \alpha_4 c_{cr} \tag{11}$$

with $\alpha_4 \ll \alpha_i$ for $i \neq 4$. Alternatively, we can exploit priorities among different cost functions by resorting to multi-objective optimization using, e.g., the lexicographic method whereby preferences are imposed by ordering the objective functions according to their significance — see [8] for details. Here we consider two cost functions:

$$\begin{array}{l} \mathcal{C}_1 = \alpha_1 c_{cf} + \alpha_2 c_{door} + \alpha_3 c_{size} \\ \mathcal{C}_2 = c_{cr} \end{array} \tag{12}$$

where the objective function $\mathcal{C}_1$ is minimized first.

4 Experimental Results

To understand the impact of different choices for encoding our problem, we consider a pool of solvers from the SMT and CP communities. We test our SMT-LIB [6] encoding with two OMT solvers, namely z3 [19] by Microsoft Research and OptiMathSat [24] by FBK, and our MiniZinc [15] encoding with five different solvers. We use the lazy clause generation based solver Chuffed [9], the MiniZinc challenge winner Google OR-Tools [20], the CLP solver ECL^{i}PS^e [22] and the two MIP solvers CPLEX [13] and Gurobi [12]. With all these solvers we can observe how different approaches in solving combinatorial optimization problems behave with our encoding choices. [1] We consider the default configuration of every solver, even if we are aware that tuning each solver for the specific problem might yield better results. However, we do not wish to introduce bias in our experiments due to the fact that we may know a solver or a technique better than others and thus obtain effective configurations on specific solvers only.

All the results are obtained considering setups with shafts of varying sizes: we have a set of eight shafts with fixed width of 1300 millimeters and another set with 1500 millimeters width. In both cases, the depth goes from 800 to 1500 using a 100 mm step. Each experiment is subject to a timeout of 5 minutes of CPU time, and the times are expressed in milliseconds.[2] Since we evaluate diverse solvers, we consider run time as the only yardstick for comparison. Other measures, e.g., size of explored search space, number of sub-problems generated, might make sense only for specific solvers. Furthermore, our ultimate goal is to find an approach viable for practical applications where execution time is the most crucial aspect. We consider two different sets of experiments: a baseline encoding dealing with the configuration of the car frame and the door pair only, and a full encoding dealing also with the selection and sizing of the hydraulic cylinder as well as the minimization of forces on the car rails. In particular, in the baseline encoding we consider only the cost components related to car frame and doors, whereas the full encoding takes into account all the cost components. Overall, the baseline encoding features 29 parameters and 10 decision variables, whereas the full encoding features 42 parameters and 17 decision variables. The number of constraints varies from a minimum of 30 for the baseline encoding considering arrays and functions to 401 for the full encoding with implications to represent parameters and look-up tables. Both the baseline and the full encodings are generated considering a database of 25 car frames, 236 doors and 47 hydraulic cylinders.

In Table 2 we show the results obtained on the baseline encoding by all the solvers we consider. For each solver we report the best time obtained on two variations: one in which the selection of components is based on arrays and another

[1] We run Chuffed v0.10.3, OR-Tools v7.8, ECL^{i}PS^e v7.0, CPLEX v12.7, Gurobi v9.0.1, z3 v4.8.7 and OptiMathSat v1.7.0.1. z3 and OptiMathSat do not generate proofs of their results in the (default) configuration that we tested.

[2] All tests run on a PC equipped with an Intel® Core$^{\mathrm{TM}}$ i7-6500U dual core CPU @ 2.50GHz, featuring 8GB of RAM and running Ubuntu Linux 16.04 LTS 64 bit.

Shaft	**OR-Tools**	**Chuffed**	**ECL^{i}PS^e**	**CPLEX**	**Gurobi**	**z3**	**OptiMathSat**
1300×800	662	200	924	856	886	**100**	254
1300×900	645	**198**	703	1020	1802	432	30680
1300×1000	674	**192**	1401	918	933	416	58066
1300×1100	659	**179**	1734	940	971	582	154739
1300×1200	655	**191**	1796	1056	1237	417	82698
1300×1300	661	**188**	1771	1090	1725	495	100822
1300×1400	637	**188**	1366	918	887*	435	79323
1300×1500	672	**206**	875	1118	925*	517	98355
1500×800	644	199	678	1023	824	**116**	247
1500×900	664	**179**	691	902	881	787	101458
1500×1000	673	**195**	1379	987	887*	619	70082
1500×1100	639	**206**	1942	971	903	682	105071
1500×1200	660	**264**	2024	1060	934	501	83719
1500×1300	636	**224**	2412	987	1018	417	121801
1500×1400	645	**192**	1509	871	919	470	97753
1500×1500	653	**216**	845	856	935	463	142557

Table 2. Comparison of solvers on the baseline encoding: the first column reports the setup and the other columns report the time (ms) taken to solve each setup by the solvers — best times appear in boldface.

featuring Boolean implications. Both variations are integer-based because not all the solvers support arithmetic over reals, so we do not consider relaxations here; also, since the car surface computation involves a division, we omit the deduction of the car payload and passengers which are required for a complete design. All the solvers leveraging MiniZinc encodings fare the best runtime when the component parameters are encoded with arrays: CP solvers like Chuffed seem to make effective use of element constraints and MIP solvers appear to handle the translation of array constraints better than Boolean implications. On the other hand, OMT solvers run faster on the version based on Boolean implications, as the addition of arrays involves dealing with more theories at once and this inevitably hurts performances.

As we can observe in Table 2, Chuffed is the one yielding the best runtimes, except for two setups where z3 is the fastest solver. Noticeably, these setups do not admit a feasible configuration given the shaft size and the components available. z3 and OR-Tools are second best, their runtimes being always less than one second; MIP solvers CPLEX and Gurobi seem slightly less effective than the leading pack. In some cases, marked with an asterisk in Table 2, Gurobi returned "*UNSAT or UNKNOWN*" as an answer even if a solution exists and the MiniZinc file is the same for all solvers, so we conjecture that numerical stability might be an issue in these cases, but we are investigating other potential causes. ECL^{i}PS^e results are mixed, i.e., some setups are solved faster than OR-Tools or z3, others take more than two seconds to solve. OptiMathSat is surprisingly slow on these encodings: if we exclude scenarios for which no feasible configuration exists, then OptiMathSat best result is 30 seconds to solve the 1300×900 setup.

	z3						OptiMathSat						
Shaft	**I**		**I + R**		**I + R + F**		**I**		**I + R**		**I + R + F**		**Heuristic**
	SO	**MO**	**SO**	**MO**	**SO**	**MO**	**SO**	**MO**	**SO**	**MO**	**SO**	**MO**	
1300×800	157	149	131	134	143	221	109	119	**100**	131	116	110	583
1300×900	8878	229522	1205	**844**	1978	2321	—	—	74960	53535	68215	52068	1784
1300×1000	36120	133325	2818	3362	4433	**2704**	156761	48328	136109	107356	132166	112379	921
1300×1100	36448	60589	3514	1967	2365	**1554**	192198	127753	160883	176491	199352	113889	2177
1300×1200	42328	5530	6876	3460	**2637**	4155	—	208852	276380	181817	193987	160401	6865
1300×1300	94325	8982	22279	**2521**	5294	5304	244973	129848	225067	165818	292777	197777	15278
1300×1400	30452	133087	7374	**1779**	11096	3707	259953	244078	—	256791	—	242488	11190
1300×1500	177355	25697	18810	4061	234235	**1998**	258119	213104	259693	172842	274986	222485	24380
1500×800	176	141	121	114	140	129	100	**85**	100	**85**	100	101	926
1500×900	25964	56674	1619	**1212**	3382	1876	141359	—	206751	95370	167671	100623	5215
1500×1000	91242	235192	2888	1803	5121	**1725**	—	118777	223241	93759	173623	187153	2952
1500×1100	—	18023	4977	7517	**3925**	4446	219596	187570	205041	179414	183862	156035	4875
1500×1200	139993	68562	7001	**1242**	7431	1571	251651	148664	231829	70431	254111	189705	6232
1500×1300	291712	—	26724	4895	20263	**4325**	—	225509	—	232735	—	255728	33785
1500×1400	—	6264	35073	3139	169215	**2675**	—	184555	271054	107886	—	180857	21910
1500×1500	—	17824	37722	2703	121472	**2528**	257242	222360	—	167762	—	251033	8699

Table 3. Comparison of z3 and OptiMathSat on the full encoding: the first column reports each setup; the other columns, grouped by solver, report runtimes (ms) of different versions: integer-based "**I**", relaxed "**I + R**" and relaxed with functions "**I + R + F**", respectively. Subcolumns "**SO**" and "**MO**" refer to single objective and multiobjective optimization, respectively — best times among z3 and OptiMathSat appear in boldface. The last column reports LiftCreate heuristic engine runtimes.

When considering the full encoding, we limit our comparison to z3 and OptiMathSat, since they are the only ones that appear to handle encodings which contain a substantial part of arithmetic over reals involved in cylinder selection, sizing and computation of forces on the car rails. Among the MiniZinc-based tools, ECL^{i}PS^e is meant to support arithmetic over reals, but even the baseline encoding with relaxations resulted in a timeout for every setup other than the ones for which no feasible configuration exists. We experimented also with OR-Tools on an encoding obtained considering fixed-point arithmetic over 64 bit integers, but to no avail. We did not try the fixed-point encoding on other CP tools as they do not support 64 bit precision which is the least one required to avoid overflowing the calculations. In Table 3 we collect the results of the comparison between z3 and OptiMathSat, adding the runtime of the heuristic search performed by LiftCreate for reference. We focus on the implication-based encoding given the results with the baseline encoding. In the table, columns labeled "**I**" report runtimes on the integer-based versions, columns labeled "**I+R**" report runtimes on relaxed versions, and columns labeled "**I+R+F**" report runtimes on versions where look-up tables are represented as nested *if-then-else* functions rather than straight implications. The columns "**SO**" and "**MO**" report the results of single-objective and multi-objective optimization, respectively. In the single-objective case, considering equation (11), we set the free parameters α_1, α_2 and α_3 to 0.3 and α_4 to 0.1 in order to encode different priorities. In the multi-objective encoding, we set all weights to one. The choice of the weights reflects that the first three components of the cost function have the same priorities. Different weights could be chosed according to the user's preferences, and we know — from other experiments that we do not show here to save space — that different choices do not impact on performances.

Considering the results in Table 3, we see that the integer-based version of the full encoding is the least appealing option: while z3 performs slightly better than OptiMathSat on this version, other solutions yield faster runtimes. In particular, relaxing the encoding has a substantial impact both on z3 and OptiMathSat: solving time decreases by orders of magnitude in some cases with respect to the integer-based encoding. Finally, considering the addition of native SMT-LIB functions we see that the results are mixed, i.e., it is not so clear that choosing them improves the solving time. Noticeably, while OptiMathSat remains slower than z3, it never exceeds the time limit on this encoding. As for single vs. multi-objective encoding, we can see that the multi-objective approach performs better than the single-objective one. In spite of some exceptions, multi-objective optimization — specifically, with z3, relaxed encodings and native SMT-LIB functions — seems to be the winning option overall. When it comes to comparing the heuristic engine of LiftCreate with the best results of the constraint-based approach, we should take into account that the former deals with the *complete* design cycle and not just with some subtasks. Given this initial bias, that in some cases the heuristic engine outperforms most constraint-based solutions, but it is overall slower than the best ones, it is fair to say that OMT solvers with relaxed encodings and multi-objective optimization provide a feasible replacement to heuristic search in the design subtasks that we considered here.

5 Conclusions

In this paper we have considered the problem of solving some subtasks in automated design and configuration of elevator systems with a purely constraint-based approach. Elevator configuration is a seemingly neglected problem in the constraint programming literature, but we believe that it offers many interesting starting points for useful research that can spill over to other domains. We have shown that some combinations of encoding techniques like relaxations and multi-objective optimization together with specific solvers seem more promising than others, and that the best combination is competitive with special-purpose heuristics. Noticeably, this result has been obtained without tweaking the default configuration of the solvers to match the specific problem. While better results could be obtained for specific configurations we believe that an added value of a solver lies also in its ability to tackle problems effectively without undergoing extensive customization.

Bibliography

[1] MiniZinc constraint modeling language (2019), `https://www.minizinc.org/`

[2] Annunziata, L., Menapace, M., Tacchella, A.: Computer intensive vs. heuristic methods in automated design of elevator systems. In: European Conference on Modelling and Simulation, ECMS 2017, Budapest, Hungary, May 23-26, 2017, Proceedings. pp. 543–549 (2017)

[3] Ansótegui, C., Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving weighted csps with meta-constraints by reformulation into satisfiability modulo theories. Constraints An Int. J. **18**(2), 236–268 (2013)

[4] Bacchus, F.: Gac via unit propagation. In: International conference on principles and practice of constraint programming. pp. 133–147. Springer (2007)

[5] Barret, Clark and Fontaine, Pascal and Tinelli, Cesare: The smt-lib standard - version 2.6 (2017), `http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf`

[6] Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. In: Gupta, A., Kroening, D. (eds.) Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK) (2010)

[7] Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving constraint satisfaction problems with SAT modulo theories. Constraints An Int. J. **17**(3), 273–303 (2012)

[8] Chang, K.H.: Chapter 19 - multiobjective optimization and advanced topics. In: Chang, K.H. (ed.) e-Design, pp. 1105 – 1173. Academic Press, Boston (2015)

[9] Chu, G.: Improving combinatorial optimization. In: Twenty-Third International Joint Conference on Artificial Intelligence (2013)

[10] Contaldo, F., Trentin, P., Sebastiani, R.: From minizinc to optimization modulo theories, and back (extended version) (2019), available at `http://arxiv.org/abs/1912.01476v2`

[11] Demarchi, S., Menapace, M., Tacchella, A.: Automating elevator design with satisfiability modulo theories. In: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI). pp. 26–33. IEEE (2019)

[12] Gu, Z., Rothberg, E., Bixby, R.: Gurobi optimization (2019), `http://www.gurobi.com/`

[13] IBM: Ibm ilog cplex optimization studio (2017) cplex users manual, version 12.7 (2017)

[14] Marcus, S., Stout, J., McDermott, J.: Vt: An expert elevator designer that uses knowledge-based backtracking. AI magazine **8**(4), 41–41 (1987)

[15] Marriott, K., Stuckey, P.J., Koninck, L., Samulowitz, H.: A minizinc tutorial (2014)

[16] McDermott, J.: R1: The formative years. AI magazine **2**(2), 21–21 (1981)

[17] Mittal, S., Falkenhainer, B.: Dynamic constraint satisfaction. In: Proceedings eighth national conference on artificial intelligence. pp. 25–32 (1990)

[18] Mittal, S., Frayman, F.: Towards a generic model of configuraton tasks. In: IJCAI. vol. 89, pp. 1395–1401 (1989)
[19] de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
[20] Perron, L., Furnon, V.: Or-tools (2020), `https://developers.google.com/optimization/`
[21] Quva, M.: A framework for constraint-programming based configuration (05 2011)
[22] Schimpf, J., Shen, K.: Eclipse from lp to clp. Theory and Practice of Logic Programming **12**(1-2), 127156 (2012). https://doi.org/10.1017/S1471068411000469
[23] Sebastiani, R., Trentin, P.: On optimization modulo theories, maxSMT and sorting networks. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 231–248. Springer (2017)
[24] Sebastiani, R., Trentin, P.: Optimathsat: A tool for optimization modulo theories. Journal of Automated Reasoning (Dec 2018). https://doi.org/10.1007/s10817-018-09508-6, `https://doi.org/10.1007/s10817-018-09508-6`
[25] Stumptner, M., Friedrich, G.E., Haselbok, A.: Generative constraint-based configuration of large technical systems. Artificial Intelligence for Engineering Design, Analysis and Manufacturing **12**(4), 307320 (1998). https://doi.org/10.1017/S0890060498124046
[26] Zhang, L.L.: Product configuration: a review of the state-of-the-art and future research. International Journal of Production Research **52**(21), 6381–6398 (2014)