A Hybrid System for Warehouse Layout Planning Based on Answer Set Programming and Conditional Expert Knowledge

Andre Thevapalan[1,3], Marco Wilhelm[1], Gabriele Kern-Isberner[1], Pascal Kaiser[2], and Moritz Roidl[2]

[1] Dept. of Computer Science, TU Dortmund University, Germany
[2] Dept. of Mechanical Engineering, TU Dortmund University, Germany
[3] andre.thevapalan@tu-dortmund.de

Abstract. Planning warehouse layouts is a complex logistical task the success of which heavily depends on the expertise of the warehouse designer. A common strategy in warehouse layout planning is to begin by allocating the main functional areas of the warehouse, also known as block layout planning, before placing individual facilities. In this paper, we propose a hybrid reasoning system to assist the warehouse designer during the block layout planning process. In a first step, answer set programming (ASP) is used to compute all feasible allocations of the functional areas which are in compliance with technical domain knowledge and instance-specific data. Afterwards, these allocations are rated based on plausible logistical expert knowledge which is formalized in a conditional knowledge base. This procedure passes into an iterative process involving adaptations of the ASP program until "plausibly good" block layouts are found or the user interacts with the system to provide further guidance. The benefits of our approach are manifold. In particular, we aim to accelerate the warehouse layout planning process and increase the reproducibility as well as explainability of the final layouts.

Keywords: (preferred) answer set programming · conditionals · c-representations · hybrid reasoning · warehouse layout planning.

1 Introduction

Planning layouts of warehouses is a crucial task in logistics because the warehousing costs are determined by the warehouse layout to a large extent already [24]. The planning process is characterized by many trade-offs between conflicting objectives with the overall goal of minimizing the operational costs [18] which makes it a highly complex task that typically results in a large number of feasible layouts of varying quality. The finally realized layout heavily depends on the warehouse designer's expertise [34]. A common strategy to address the complexity of warehouse layout planning is to start with a rough layout of the warehouse before single facilities are placed [7]. In this first phase of the planning process the main functional areas of the warehouse like the receiving, picking, storage,

and departure areas are arranged. It results in a *block layout* that is usually based on maximizing the operational efficiency determined by the expected material flow between the functional areas (also known as the *facility layout problem*, cf. [7]). A block represents the region of a functional area then. This first phase of the warehouse layout planning task is followed by the detailed planning of the functional areas which involves the placement of facilities, storage spaces, paths and so on.

In this paper, we focus on the block layout planning task and propose an interactive hybrid reasoning system to assist the warehouse designer during the planning process. Our system combines two well-established approaches from the field of knowledge representation and reasoning, namely *answer set programming (ASP)* [13, 12] and *conditional reasoning* [8, 1], and provides the user with plausible arrangements of the functional areas within the warehouse. These block layouts can then be refined and adapted to individual requests of the warehouse designer. The benefits of our system include a speed-up of the planning process through the automated solving of combinatorial allocation problems, thus a reduction of the planning costs, as well as an increased reproducibility and explainability of the final layouts because the logistical background knowledge is made explicit in form of ASP rules and conditionals. This may help to increase the acceptance of the layouts by the customer. Furthermore, our system possibly leads to layout suggestions that can be easily overlooked by a human planner due to the complexity of the problem.

In our hybrid reasoning system, answer set programming is used to describe technical requirements to warehouse layouts and to calculate all feasible block layouts which are in compliance with these requirements. ASP is a form of declarative programming which is based on logical rules. Its main feature is the implementation of *default negation* [5] which allows the user to formulate default statements such as "$H \leftarrow A, \texttt{not}\ B$." expressing that H follows from A unless (the exceptional case) B holds. With the power of default negation and its highly optimized solvers (cf. `clingo`, [10]), ASP is able to solve complex combinatorial search problems. Similar to ASP rules, *conditionals* express defeasible statements but call for a richer semantics which involves a notion of plausibility. A *ranking function* κ accepts a conditional $(B|A)$, meaning "if A holds, then usually B follows," if the verification of the conditional is more plausible than its falsification, in symbols if $\kappa(A \wedge B) < \kappa(A \wedge \neg B)$. Hence, conditionals are especially suited to express preferences. Here, we use conditional reasoning to calculate a preference ordering on the feasible block layouts computed with ASP in order to rate them with respect to their quality. With conditional knowledge bases, we make the logistical commonsense knowledge of warehouse designers explicit.

Up to now, for the facility layout problem no holistic computer-aided approaches are available [3] so that the planning of block layouts is usually done by hand to a large extent. Current related research is mostly focused on computing layouts using genetic algorithms [22, 28, 32] which have the disadvantage of being a black-box approach, i.e., details on how a layout was formed can generally not be determined. Recent work also utilizes fuzzy logic in connection

with user experience [2, 20] which, from a semantical viewpoint, is significantly different from our hybrid reasoning approach. In their survey paper [27], the authors mention knowledge-based systems developed to solve the facility layout problem. Those systems are, however, built upon very limited forward chaining inference techniques. While the authors in [26] use answer set programming in the logistics domain to allow efficient cooperation between logistics robots, to the best of our knowledge, the idea to use ASP in combination with sophisticated conditional reasoning techniques to solve the facility layout problem has yet to be explored. For an overview of research on facility layout problems, we refer the reader to [23, 28]. Note that our approach on combining ASP and conditional reasoning is not limited to the facility layout problem but is useful whenever combinatorial problems can be expressed in form of logical, possibly defeasible statements.

The rest of the paper is organized as follows. In Sec. 2 we settle the logical foundations of our hybrid reasoning system, which involves a brief introduction to answer set programming and conditional reasoning. In Sec. 3 we recall our approach on prioritizing answer sets based on conditional expert knowledge from [33] and discuss related work. Afterwards, we specify the block layout planning problem in Sec. 4 and present our hybrid reasoning system in Sec. 5. In Sec. 6 we evaluate our approach based on a typical real world problem, before we conclude with an outlook in Sec. 7.

2 Logical Foundations

As a background language we consider a *relational language* $\mathcal{L}(\Sigma)$ [15] defined over a *signature* $\Sigma = (\mathcal{C}_\Sigma, \mathcal{P}_\Sigma)$ where $\mathcal{C}_\Sigma$ and $\mathcal{P}_\Sigma$ are finite sets of *constants* and *predicates* respectively. *Formulas* in $\mathcal{L}(\Sigma)$ are either *atoms*, i.e., predicates of arity n together with an n-tuple of constants or variables, or they are recursively defined as *negations* $\neg A$, *conjunctions* $A \wedge B$, or *disjunctions* $A \vee B$ where $A, B \in \mathcal{L}(\Sigma)$. A *literal* is an atom or its negation. An expression (atom, literal, formula, etc.) is *ground* if it does not mention variables. A ground formula is called a *sentence*. The semantics of sentences is given by *interpretations* mapping them to 0 or 1 as usual. We abbreviate $\neg A$ with $\overline{A}$, $A \wedge B$ with AB, and denote arbitrary *tautologies* $A \vee \overline{A}$ with $\top$ and arbitrary *contradictions* $A\overline{A}$ with $\bot$.

Answer Set Programming *Answer set programming (ASP)* [13, 12] is a logic-based declarative programming language with the *default negation* "not" [5] as its main feature. An *ASP program*, or *program* for short, is a finite set of *rules*

$$r\colon \quad H \leftarrow A_1, \ldots, A_k, \mathsf{not}\ B_1, \ldots, \mathsf{not}\ B_l., \tag{1}$$

where $H, A_1, \ldots, A_k, B_1, \ldots, B_l$ are literals from $\mathcal{L}(\Sigma)$. Rules without *body literals*, i.e., rules of the form (1) with $k = l = 0$, are called *facts* and are abbreviated as "H.". Rules without a *head literal* H are *constraints*. If a rule mentions a variable, then it is understood as a schema and has to be *grounded* before solving the program. The *grounding* of a rule r means the syntactical replacement of

each variable in r by a constant from $\mathcal{C}_\Sigma$. A ground rule of the form (1) can be read as: "If the *positive body literals* $A_1, \ldots, A_k$ hold and it can be assumed that the *default-negated body literals* $B_1, \ldots, B_l$ do not hold, then the *rule head* H follows." The assumption that B_j does not hold does not mean that its complement $\overline{B_j}$ has to be true; it is also appropriate that the truth value of B_j is *unknown* which differentiates the default negation "not" from the classical negation "$\neg$". The *grounding* $\mathcal{P}^g$ of a program $\mathcal{P}$ is obtained by replacing all rules that mention variables by all of their possible groundings. The formal semantics of ASP programs is defined based on their *Gelfond-Lifschitz-reduct* [14]: Let $\mathcal{P}$ be a ground program, and let $\mathcal{S}$ be a *state* which is a consistent set of ground literals from $\mathcal{L}(\Sigma)$, i.e., there is no ground atom $A \in \mathcal{L}(\Sigma)$ with $\{A, \overline{A}\} \subseteq \mathcal{S}$. Then, the *Gelfond-Lifschitz-reduct* of $\mathcal{P}$ wrt. $\mathcal{S}$ is the program[4]

$$\mathcal{P}^{\mathcal{S}} = \{H \leftarrow A_1, \ldots, A_k. \mid H \leftarrow A_1, \ldots, A_k, \mathsf{not}\ B_1, \ldots, \mathsf{not}\ B_l. \in \mathcal{P} \quad \text{and} \quad \forall j \in [l]\colon B_j \notin \mathcal{S}\}.$$

The reduct $\mathcal{P}^{\mathcal{S}}$ is a classical logic program free of default negation. Such programs provide a unique minimal model which we denote with $\mathsf{Cl}(\mathcal{P}^{\mathcal{S}})$ and which can be obtained by recursively applying the rules in $\mathcal{P}^{\mathcal{S}}$, beginning with the facts. In other words, $\mathsf{Cl}(\mathcal{P}^{\mathcal{S}})$ is the inclusion minimal set of ground literals so that for all rules $r\colon H \leftarrow A_1, \ldots, A_k.$ in $\mathcal{P}^{\mathcal{S}}$, if $A_1, \ldots, A_k \in \mathsf{Cl}(\mathcal{P}^{\mathcal{S}})$, then $H \in \mathsf{Cl}(\mathcal{P}^{\mathcal{S}})$. Based on this, $\mathcal{S}$ is called an *answer set* (= model) of $\mathcal{P}$ if $\mathsf{Cl}(\mathcal{P}^{\mathcal{S}}) = \mathcal{S}$. Unlike classical logic programs, ground ASP programs may have several models, or also none. With $\mathcal{AS}(\mathcal{P})$ we denote the set of all models (answer sets) of $\mathcal{P}$. The answer sets of a non-ground ASP program $\mathcal{P}$ are the answer sets of its grounding, i.e., $\mathcal{AS}(\mathcal{P}) = \mathcal{AS}(\mathcal{P}^g)$. A program $\mathcal{P}$ is *consistent* iff $\mathcal{AS}(\mathcal{P}) \neq \emptyset$.

Conditional Reasoning A *conditional* $(B|A)$ where A and B are sentences from $\mathcal{L}(\Sigma)$ is a syntactic representation of the defeasible statement "if A holds, then *usually* B follows," where the formal semantics of "usually" has to be made precise. So-called *open conditionals* $(B|A)$ where A or B contains free variables are understood as schemata here. Before evaluating them they have to be grounded by a proper instantiation through constants like rules in ASP. Finite sets of conditionals are called *knowledge base*. Throughout the paper, we enumerate the conditionals in knowledge bases Δ and refer to the i-th conditional in Δ with $\delta_i = (B_i|A_i)$. In addition, we allocate $n = |\Delta|$. The formal semantics of conditionals is based on *possible worlds*. Here, possible worlds are the interpretations from $\mathcal{L}(\Sigma)$ represented as complete conjunctions of ground literals. Each ground atom from $\mathcal{L}(\Sigma)$ occurs in a possible world ω once, either positive or negated. The set of all possible worlds is denoted by $\Omega(\Sigma)$. A *ranking function* [29] is a mapping $\kappa\colon \Omega(\Sigma) \to \mathbb{N}_0^\infty$ which assigns to every possible world a degree of implausibility while satisfying the normalization condition $\kappa^{-1}(0) \neq \emptyset$. The higher its *rank* $\kappa(\omega)$, the less plausible a possible world ω is. Ranking functions are extended to sentences via $\kappa(A) = \min_{\omega \in \Omega(\Sigma)\colon \omega \models A} \kappa(\omega)$

[4] We abbreviate $[l] = \{1, \ldots, l\}$ with the special case $[0] = \emptyset$.

and *accept* a conditional $(B|A)$ iff $\kappa(AB) < \kappa(A\overline{B})$, i.e., iff the *verification* AB of the conditional is more plausible than its *falsification* $A\overline{B}$. A ranking function κ is a *ranking model* of a knowledge base Δ iff κ accepts all conditionals in Δ. If a knowledge base Δ has a ranking model, then it is called *consistent*. An elaborate, principle-based class of ranking models is constituted by the so-called *c-representations* [17]. A ranking function κ is a *c-representation* of Δ if κ is a ranking model of Δ and there is $\boldsymbol{\eta} \in \mathbb{N}_0^n$ such that

$$\kappa(\omega) = \sum\nolimits_{\delta_i \in \mathsf{fal}_\Delta(\omega)} \eta_i, \quad \omega \in \Omega(\Sigma),$$

where $\mathsf{fal}_\Delta(\omega) = \{\delta_i \in \Delta \mid \omega \models A_i\overline{B_i}\}$ is the set of conditionals from Δ which are falsified in ω. We write $\kappa = \kappa_\Delta^{\boldsymbol{\eta}}$ in this case. The *value* η_i can be understood as a penalty point for falsifying the i-th conditional from Δ. If a knowledge base is consistent, then it also has a c-representation.

3 Preferred ASP via Conditional Expert Knowledge

Applying answer set programming to real world problems can lead to a vast number of answer sets which are not necessarily equally meaningful. This is particularly the case when "soft constraints" play a role, the purpose of which is not to constrain the answer sets, but to imply preferences among them. Soft constraints cannot be formalized in pure ASP, though. In our hybrid reasoning system, we make use of our approach first presented in [33] in order to rate answer sets. We briefly recall this approach and discuss related work afterwards.

Combining ASP and Conditional Reasoning In line with [33], let $\mathcal{P}$ be an ASP program, and let Δ be a consistent knowledge base which formalizes plausible expert knowledge about the domain considered in $\mathcal{P}$. The signatures of $\mathcal{P}$ and Δ may differ but are intended to have a significant overlap. Further, let κ be a ranking model of Δ. For each answer set $\mathcal{A}$ of $\mathcal{P}$ we compute its κ-rank

$$\kappa(\mathcal{A}) := \kappa(\bigwedge\nolimits_{l \in \mathcal{A}} l)$$

and say that $\mathcal{A}$ is *preferred* to $\mathcal{A}' \in \mathcal{AS}(\mathcal{P})$ if $\kappa(\mathcal{A}) < \kappa(\mathcal{A}')$, and that $\mathcal{A}$ and $\mathcal{A}'$ are *equally preferred* if $\kappa(\mathcal{A}) = \kappa(\mathcal{A}')$. Therewith, we establish a preference preorder on $\mathcal{AS}(\mathcal{P})$, which reflects the plausibility of the answer sets of $\mathcal{P}$ with respect to the expert knowledge in Δ. The *most preferred answer sets* $\mathcal{A}$ of $\mathcal{P}$ are those with $\kappa(\mathcal{A}) \leq \kappa(\mathcal{A}')$ for all $\mathcal{A}' \in \mathcal{AS}(\mathcal{P})$, i.e.,

$$\mathsf{mpref}_\kappa(\mathcal{P}) := \{\mathcal{A} \in \mathcal{AS}(\mathcal{P}) \mid \forall \mathcal{A}' \in \mathcal{AS}(\mathcal{P}) \colon \kappa(\mathcal{A}) \leq \kappa(\mathcal{A}')\}. \tag{2}$$

This set is non-empty by construction. We give a simple yet generic example which illustrates the approach.

Example 1. Let the ASP program $\mathcal{P} = \{r_1, \ldots, r_4\}$ with

$$r_1\colon\ s.,\qquad r_2\colon\ c \leftarrow s.,\qquad r_3\colon\ p \leftarrow c, \mathsf{not}\ \bar{p}.,\qquad r_4\colon\ \bar{p} \leftarrow s, \mathsf{not}\ p.,$$

state that class c instances show property p unless proven wrong (r_3) while instances of the subclass s of c (r_2) normally show the opposite of property p (r_4). In addition, rule r_1 states that we reason about the subclass s. For instance, you can think about birds, which constitute a subclass of living beings and usually are able to fly, while living beings in general usually do not fly. The answer sets of $\mathcal{P}$ are $\mathcal{A}_1 = \{c, s, p\}$ and $\mathcal{A}_2 = \{c, s, \bar{p}\}$ which means that we infer from $\mathcal{P}$ that instances of s and, therewith, of c may or may not show property p. This inference is very cautious because intuitively we would assume that the more specific information that subclass s instances usually do not show property p outweighs the information that general instances of class c do show property p. In other words, one would assume that subclasses inherit properties from their superclasses only until more specific information about the subclass is known. However, the fact that s forms an exceptional subclass of c with respect to property p is not fully reflected by $\mathcal{P}$. The point here is that the rules r_3 and r_4 are applied without any precedent.

Following the approach from [33], we can formulate the knowledge about s being an exceptional subclass of c with respect to p explicitly in form of the knowledge base $\Delta = \{\delta_1 \colon (\bar{s}|cp)\}$ ("if an instance of class c shows property p, then it is usually not an instance of subclass s"). The c-representation $\kappa^{\boldsymbol{\eta}}_{\Delta}$ of Δ with the smallest possible penalty point $\eta_1 = 1$ assigns the ranks $\kappa^{\boldsymbol{\eta}}_{\Delta}(csp) = 1$ and $\kappa^{\boldsymbol{\eta}}_{\Delta}(\omega) = 0$ for $\omega \neq csp$. Hence, $\kappa^{\boldsymbol{\eta}}_{\Delta}(\mathcal{A}_2) = 0 < 1 = \kappa^{\boldsymbol{\eta}}_{\Delta}(\mathcal{A}_1)$ and $\mathcal{A}_2$ is preferred to $\mathcal{A}_1$ due to the general knowledge in Δ.

Most preferred answer sets $\mathcal{A}$ of $\mathcal{P}$ do not necessarily satisfy $\kappa(\mathcal{A}) = 0$. This means that soft constraints do not necessarily have to hold in most preferred answer sets. The idea of most preferred answer sets is rather to satisfy all rules in $\mathcal{P}$ and be in compliance with Δ as good as possible.

Related Work on Preferred ASP In the last decades, several further approaches on *prioritized* resp. *preferred ASP* have emerged [4, 6, 21, 31, 25, 35]. Most of them, like the leading approach from [4], require an ordering on the rules in the ASP program as an additional input in order to rate the answer sets.

Example 2. Following the approach from [4], the fact that in Example 1 s is an exceptional subclass of c with respect to p is implicitly realized by preferring the application of r_4 (the information about the subclass s) to r_3 (the information about the class c). Rules that are preferred have to be applied first, which, in this case, also leads to the unique preferred answer set $\mathcal{A}_2$.

Compared to [4], the approach from [33] has some advantages with respect to our application to the warehouse layout planning task: (1) While the approach from [4] returns a single most preferred answer set, the approach from [33] leads to a preference ordering on the whole set of answer sets $\mathcal{AS}(\mathcal{P})$. In particular, there may be several most preferred answer sets which can be presented to the user as plausible alternatives. (2) The calculation of the preference ordering on $\mathcal{AS}(\mathcal{P})$ according to [33] can automatically be inferred from the knowledge

	Functional area		Functional area
asps	Automatic small parts storage	*op*	Order picking area
con	Consolidation area	*ps*	Packing station
dep	Departure area	*rc*	Receiving area
hbs	High-bay storage	*sps*	Special parts storage

Table 1: Overview of the functional areas considered in this paper.

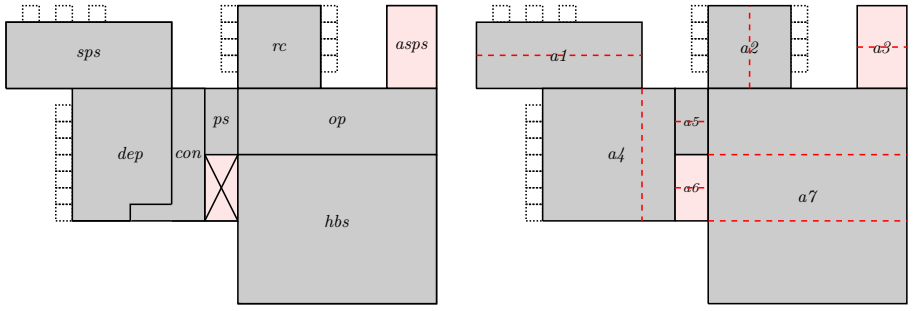
base Δ, while in [4] the ordering on the rules in $\mathcal{P}$ has to be established by hand. Especially if $\mathcal{P}$ is large or $\mathcal{P}$ has to be updated, setting up the ordering on the rules manually can become a difficult task. (3) The expert knowledge which gave reason to prefer specific ASP rules resp. answer sets can be made explicit in the knowledge base Δ, and the preferences can be derived from that. This improves the transparency of the prioritization and generates generic justifications for explanations. In addition, technical domain knowledge and expert knowledge can be clearly separated into $\mathcal{P}$ and Δ.

4 Block Layout Planning

We specify the task of *block layout planning* which we aim to solve with our hybrid reasoning system. Under *block layout planning*, we understand the task of arranging the *functional areas* of a warehouse which are relevant for the warehousing process within an empty floor plan. Hereby the warehouse is specified by its surrounding walls as well as further structural components such as gates and areas which are not directly relevant for the warehousing process. The set of functional areas may depend on the warehouse type but is committed in this paper to those areas that are listed in Tab. 1 and which comprise typical functional areas of various warehouse types. The allocation of the functional areas is subject to more or less strict restrictions. Strict resp. *technical* restrictions have to be satisfied in any case to obtain a feasible layout. Typical technical restrictions are:

T1 The functional areas must fit into the warehouse.
T2 The size of a functional area may not fall below a lower bound. This minimal size of the functional area is the result of the required throughput of goods in the warehouse and can be computed in advance (cf. [16] for computation methods).
T3 Functional areas have to be connected.
T4 Some functional areas like the receiving area, the departure area, and the special parts storage require a direct access to gates.
T5 Reserved areas of the warehouse have to be kept empty.
T6 Individual wishes of the customer have to be considered.

Some further requirements impact the operational efficiency of the warehouse or other planning goals but are rather quality criteria for preferred warehouse



(a) Prototypical block layout. (b) Possible refinements of initial zoning.

Fig. 1: Warehouse asse from Ex. 3.

layouts than hard constraints. These "plausible constraints" are typically part of the warehouse designer's expert knowledge and rarely formalized. In the first place, plausible constraints arise from the required material flow between the functional areas. Functional areas with a high material flow in-between should be placed next to each other. For example, it is plausible to place the consolidation area where packed goods are assembled to an order next to the departure area. Here, we consider the following plausible expert knowledge:

P1 Functional areas with high material flow in-between are usually next to each other. This particularly holds for the material flow chain "$asps \rightarrow op \rightarrow ps \rightarrow con \rightarrow dep$" which is relevant for the taking out of the warehouse.

P2 Functional areas usually have gates only if they need them and gates are usually solely next to functional areas that need gates.

Example 3. A floor plan of the warehouse asse ("another supplier storing everything"), which is actually a real world example of a warehouse, with a possible realization of a block layout is shown in Fig. 1a. The gray area is the area of the warehouse which shall be filled with the functional areas. The red area in the upper right is an attached building that is reserved for the automatic small parts storage. The other red area marked with a cross is not part of the warehousing area but provides facilities for the workers. Gates are indicated by dashed lines. The warehouse has a total size of $72,450\,m^2$ and the minimal sizes of the functional areas were computed by a logistics expert to:

$$\begin{aligned} \mathsf{size}(asps) &\geq 4,500\,m^2, & \mathsf{size}(con) &\geq 4,800\,m^2,\\ \mathsf{size}(dep) &\geq 9,500\,m^2, & \mathsf{size}(hbs) &\geq 19,000\,m^2,\\ \mathsf{size}(op) &\geq 9,000\,m^2, & \mathsf{size}(ps) &\geq 1,100\,m^2,\\ \mathsf{size}(rc) &\geq 6,000\,m^2, & \mathsf{size}(sps) &\geq 8,000\,m^2. \end{aligned}$$

5 Hybrid Reasoning System for Block Layout Planning

We present our hybrid reasoning system based on ASP and conditionals developed to assist warehouse designers during the block layout planning (cf. Sec. 4).

Algorithm 1 General functionality of our hybrid reasoning system

Input: ASP program $\mathcal{P}$ and ranking model κ of Δ formalizing the block layout planning task
Output: Set of block layouts L

$L = \emptyset$
while L is not accepted**:**
  compute feasible allocations $\mathcal{AS}(\mathcal{P})$
  filter preferred allocations $\mathsf{mpref}_\kappa(\mathcal{P})$
  $L =$ "rendered allocations encoded in $\mathsf{mpref}_\kappa(\mathcal{P})$"
  if L does not contain ambiguous zones**:** system accepts L
  else refine zoning and adapt $\mathcal{P}$
  if size of zone gets under user-defined threshold**:** ask user to accept L
return L

First, we give an overview of the system and then go into the details of the workflow (cf. Alg. 1).

Overview The ultimate goal of our system is to find a *zoning* of the warehouse for which most plausible assignments of the functional areas (Tab. 1) to the zones exist which satisfy all required constraints as best as possible. Given an empty floor plan and an initial division of this plan into *zones*, we calculate with ASP all feasible allocations of the functional areas to the zones. The functional areas may be allocated to several zones, and zones may house several functional areas. Afterwards, we rate these allocations based on a conditional knowledge base comprising logistical expert knowledge about block layout planning. If in all most preferred allocations, each zone hosts at most one functional area, then these most preferred allocations are presented to the user as plausible alternatives for block layouts of the warehouse. Otherwise, there is at least one zone with an ambiguous allocation of functional areas. In this case, we automatically refine the zoning and start the process anew such that the computations pass into an iterative process. Notice that in interaction with the user we allow to end the process prematurely if the size of zones gets under a user-defined threshold. By this we prevent that zones in a layout get too fine-granular which as a result can lead to a vast amount of very similar and thereby (for the user) redundant solutions. The user can also inspect the latest layouts and decide to continue the process or appropriately adapt the instance data and restart the process anew.

Our system requires the user to provide an initial zoning of the warehouse. This zoning does not have to be a suggestion for a possible block layout already but may be much coarser. With the initial zoning, the user can counter technical constraints such as **T5** and **T6** (Sec. 4) and additional requirements like fire protection specifications, for instance. A technical constraint to zonings is that the refinement process requires rectangular zones.

In the following, we describe the single components of our system in detail, starting with the allocation process implemented in ASP. We discuss the automatic evaluation of the allocations based on conditional expert knowledge, the

iterative zone refinement which comes along with some adaptations of the ASP program to reduce computational complexity, and finally the abortion condition, hence the termination of our approach.

Allocation of Functional Areas to Zones With ASP We use an ASP program $\mathcal{P}$ to compute feasible allocations of functional areas in a given warehouse. We distinguish between instance data given as facts and the problem encoding. With the instance data, the warehouse is modeled as a set of zones that have a size, possibly a gate, and that can be adjacent to each other. A functional area has a minimal space requirement and possibly needs access to a gate.

Example 4. Consider the instance data (cf. Tab. 1 for abbreviations)

$$
\begin{array}{lll}
zone(a1)., & size(a1, 2000)., & has\_gate(a1)., \\
zone(a2)., & size(a2, 10000)., & adjacent(a1, a2)., \\
fa(hbs)., & min\_size(hbs, 11000)., & \\
fa(rc)., & min\_size(rc, 1000)., & needs\_gate(rc)., \\
\end{array}
$$

where the two zones $a1$ and $a2$, with sizes of $2,000\,m^2$ and $10,000\,m^2$, respectively, are defined. Both zones are adjacent, and zone $a1$ includes access to gates. The instance states that the two functional areas *hbs* and *rc* with a space requirement of $11,000\,m^2$ and $1,000\,m^2$, respectively, must be allocated. In this, the functional area *rc* has to have access to gates. Note that these requirements are jointly satisfiable only if *hbs* is distributed over both zones. In principle, *rc* could be distributed over both zones, too, but this is not mandatory and intuitively less plausible.

In the problem encoding, the goal of the program is to compute possible allocations of functional areas to zones. We allow a functional area to be distributed across multiple adjacent zones and for a zone to contain (parts of) multiple functional areas. Adding constraints **T1**–**T6** to the program ensures that unfeasible layouts are filtered out. Each answer set then contains a viable position for each functional area in the warehouse, represented by one or more zones. In our approach, only subset-minimal answer sets are considered, ensuring that functional areas are not spread across more zones than necessary to satisfy **T2**. In the following excerpt of the program, you can find the rule that models the allocations (also called the *generating part*, rule (3)) and two constraints corresponding to **T4** and **T5** (*testing part*, rules (4) and (5)):

$$\{assign(F, Z)\} \leftarrow fa(F), zone(Z). \tag{3}$$

$$\leftarrow needs\_gate(F), assign(F, Z), \mathsf{not}\ has\_gate(Z). \tag{4}$$

$$\leftarrow assign(\_, Z), blocked(Z). \tag{5}$$

Rule (3) is a so-called *choice rule* which states that in each answer set for every functional area F and zone Z the atom $assign(F, Z)$ is either true or false.

Example 5. For the instance data from Example 4, our ASP program would output a unique answer set that includes the literals $assign(a1, rc)$, $assign(a1, hbs)$, and $assign(a2, hbs)$, meaning that rc is assigned to the zone $a1$ and hbs is assigned to $a1$ and $a2$.

Notice that $a1$ contains multiple functional areas and is therefore possibly not precise enough for the expert. Hence, in an iterative process, we split zones to obtain more accurate allocations. This zone refinement is discussed later on.

Evaluation of the Allocations Based on Conditional Expert Knowledge Once all feasible allocations of the functional areas to the zones are computed with ASP, they are rated based on plausible logistical expert knowledge which is implemented in a conditional knowledge base Δ. The input of this evaluation component is the set $\mathcal{AS}(\mathcal{P})$ describing the assignments of the functional areas to the zones, mentioning the zones which have gates, as well as giving the information which zones are adjacent. In addition, the evaluation component needs a list of the functional areas which need gates and a list of the tuples of functional areas with expected high material flow in-between. The output is the set of the most preferred allocations $\mathsf{mpref}_{\kappa^{\boldsymbol{\eta}}_{\Delta}}(\mathcal{P})$ then (cf. (2)) where $\kappa^{\boldsymbol{\eta}}_{\Delta}$ is a c-representation of Δ. The expert knowledge in Δ, here the plausible constraints **P1** and **P2** (cf. Sec. 4), is generic in the sense that it applies to the task of planning a block layout wrt. any warehouse instance. Besides the strict separation of instance data and general domain knowledge, the benefit of considering purely generic knowledge at this point is that we can prepone and reuse the usually computationally expensive computation of $\kappa^{\boldsymbol{\eta}}_{\Delta}$. The plausible constraints **P1** and **P2** are realized in Δ by the following conditional schemata:

- $(\exists i, j.\mathsf{adj}(i, j) \wedge \mathsf{ass}(X, i) \wedge \mathsf{ass}(Y, j) | \mathsf{hMF}(X, Y))$ — *"Functional areas with high material flow in-between are usually next to each other."*,
- $(\exists X.\mathsf{needsG}(X) \wedge \mathsf{ass}(X, i) | \mathsf{hasG}(i))$ — *"Zones with gates usually house a functional area which needs gates."*,
- $(\neg\mathsf{ass}(X, i) | \mathsf{hasG}(i) \wedge \neg\mathsf{needsG}(X))$ — *"Zones with gates usually do not house functional areas which do not need gates."*,

where the variables X, Y range over the set of functional areas and the variables i, j range over the zones. The additional conditional schemata in Δ,

- $(\exists_{\leq 1} i.\mathsf{ass}(X, i) | \top)$ — *"Functional areas are usually assigned to at most one zone."*,
- $(\exists_{\leq 1} X.\mathsf{ass}(X, i) | \top)$ — *"Zones usually house at most one functional area."*,

push the iteration process towards the preferred one-to-one correspondence between functional areas and zones. Provided that there are at least two zones,[5] the c-representation $\kappa^{\boldsymbol{\eta}}_{\Delta}$ with minimal impact vector $\boldsymbol{\eta}$ is given by $\boldsymbol{\eta} = (1, \ldots, 1)$. That is, every falsification of any of the conditionals in Δ is penalized with 1

[5] In case of only one zone, there is trivially only one feasible layout (if any) which assigns all functional areas to this zone.

by κ_{Δ}^{η}. In this case, $\mathsf{mpref}_{\kappa_{\Delta}^{\eta}}(\mathcal{P})$ is the set of the answer sets (= feasible allocations) which falsify the least number of conditionals from Δ. By tracing back which conditionals are falsified by an answer set and which are not, we can give explanations for the rating of the respective allocations.

Iterative Zone Refinement If the allocation of the functional areas to the zones leads to ambiguities in the most preferred allocations, the computation passes into an iterative process, i.e., the zoning is refined and passed to the ASP solver again. Hereby, we call a zone *ambiguous* if there is a most preferred allocation which assigns more than one functional area to this zone, and we call it *unambiguous* otherwise. In the following, we explain how we store and refine the zoning. A necessary precondition of our approach is that the zones are rectangular. Therewith, they can be stored as tuples of their four bounding sides (left, top, right, and bottom side). Each side is divided into sections (intervals), depending on their functionality (a section is either a wall, a gate, or a transition between zones). Technically, sides are stored as tuples again, where each entry of the side reflects a section and consists of the starting and the end point of the section as well as a description of its functionality. When the most preferred allocations of functional areas to the zones have been computed and zoning refinement is necessary, we proceed as follows: Every unambiguous zone remains unchanged. To ambiguous zones, we apply the following refinement steps:

- Divide the zone along the perpendiculars, relative to the two facing bounding sides that are longer than the others (or relative to any side of the zone if the zone is quadratic), which go through the starting or ending point of any section of the sides unless the perpendicular intersects the relative interior of a section (1) which has the functionality "gate", or (2) which has the functionality "transition between zones" and does not span the whole side.
- If the previous step does not lead to a refinement of the zone because one of the two exceptions applies (and the zone is not quadratic), then try the same with the pair of the shorter sides.
- If the previous two steps do not lead to a refinement, divide the zone along the perpendicular bisector of the two facing sides which are longer than the others (or along both perpendicular bisectors wrt. all sides of the zone if the zone is quadratic) unless one of the exceptions (1) or (2) applies.
- If the previous step does not lead to a refinement of the zone, then try the same with the pair of the shorter sides.

Only if for all zones these steps do not lead to a refinement of the zone, then apply them without exception (2) to all ambiguous zones. Only if this does not lead to a refinement, too, then apply the refinement steps to the ambiguous zones while omitting exception (1) as well.

Example 6. Fig. 1b shows a possible zoning of the asse warehouse (black borders) and all possible refinements (dashed red lines). Note that both $a1$ and $a2$ are divided across the shorter sides as the longer sides have gates. Furthermore, $a7$ can be divided into three sections because of the borders of $a5$ and $a6$.

A general strategy of our zone refinement is to keep the number of zones small. On the one hand, this is necessary to prevent an exponential blow up when computing feasible allocations of functional areas to zones with ASP which can quickly become intractable. On the other hand, the zones shall not be confused with a grid which we place over the floor plan. The basic idea is that the zones shall adapt to the block layout as far as possible. At the end, when the iteration terminates and the final block layouts are generated, neighboring zones with the same functional area can be merged to a single zone such that there is a one-to-one correspondence between functional areas and zones (recall that we require zones which house the same functional area to be adjacent).

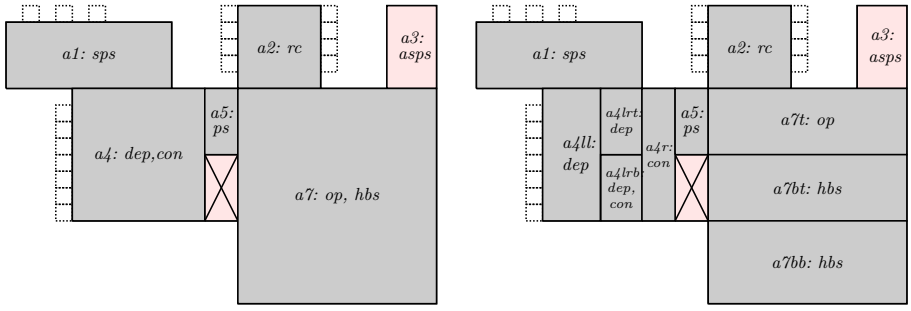
ASP Adaptations In our approach, the instantiated program $\mathcal{P}$ is adapted in each iteration step in two ways. First, the zone refinement is applied to $\mathcal{P}$ by replacing ambiguous zones by their refinements. The more zones the floor plan is divided into, the more feasible allocations of functional areas to zones can be expected. Hence, finer zonings result in greater computational costs, in particular in the ASP part of our system, and there is a need to counteract this computational blowup. Because of this, we additionally adapt the ASP program as follows. In each iteration step i, we memorize for each zone z the different functional areas $FA_{z,i}$ that were assigned to it over all preferred models. In the subsequent iteration $i+1$ only functional areas that belong to $FA_{z,i}$ can be assigned to zones that arose from z. The reduction of assignable functional areas per zone is encoded using ASP integrity constraints which, as described in [9], serve to filter out unwanted answer sets.

Example 7 (Ex. 5 contd.). Suppose that the refinement process splits zone $a1$ into $a1l$ and $a1r$. As only the areas *hbs* and *rc* were assigned to the original zone $a1$, after the refinement process, the following constraints are added to the testing part:

$$\leftarrow assign(a1l, Z), Z \mathrel{!}= hbs, Z \mathrel{!}= rc.$$
$$\leftarrow assign(a1r, Z), Z \mathrel{!}= hbs, Z \mathrel{!}= rc.$$

The constraints in Ex. 7 ensure that either *hbs* or *rc*, both, or none of these areas can be assigned to $a1l$ and $a1r$. Adding such constraints represent a heuristic extension to the system that allows to ignore assignments that were already recognized as non-preferable.

Abortion Condition and Termination The refinement process ends if either (1) at most one functional area is assigned to each zone or, (2) if the size of a zone drops under a certain threshold and the expert wishes to manually end the process because of that. The value of such a threshold can be stated by the user via the instance data. With (2) the expert is able to inspect the latest results and stop the process if they are already satisfied with the preliminary preferred layouts. In practice, the expert is able to manually refine remaining ambiguous zones in a layout more efficiently and avoid higher computation times



(a) After iteration 1. (b) After final iteration 4.

Fig. 2: Examples of most preferred allocations of functional areas for the warehouse asse from Ex. 3.

than necessary. With (1) and (2) it is guaranteed that the refinement process terminates, either because the preferred layouts only contain unambiguous zones, or because it is stopped manually by the expert.

6 Evaluation and Explanations

We illustrate our hybrid reasoning system using the warehouse asse as introduced in Ex. 3. The goal is to find the most suitable allocations of the functional areas in Tab. 1 to positions in the warehouse according to the modeled expert knowledge. For that we start with the initial zoning as shown in Fig. 1b. Note that zone $a6$ is blocked, i.e., no area can be allocated to it and it is therefore not considered for further refinements (cf. **T5**). Furthermore, we define that the refinement should stop if a zone gets smaller than 2% of the warehouse size and want to respect the user-defined assignment of the automatic small parts storage ($asps$) to $a3$ (cf. **T6**). As a consequence, neither will $asps$ be assigned to other zones than $a3$ nor will any other functional area be assigned to $a3$. As illustrated in Alg. 1, in each iteration step feasible and preferred allocations are computed and the ASP program $\mathcal{P}$ is adapted based on these results. The number of feasible (most preferred) allocations are: $14,977\,(3)$ after iteration 1, $116\,(1)$ after iteration 2, $3\,(1)$ after iteration 3, and $2\,(2)$ after iteration 4. Fig. 2 shows two allocations that are computed during the iteration process.

After the first iteration, we observe three preferred allocations which show two ambiguous zones, $a4$ and $a7$ (Fig. 2a shows one of the preferred allocations). Accordingly, these zones are refined which results in two new zones that replace $a4$ and three new zones that replace $a7$ (cf. Fig. 1b). Additionally, the program $\mathcal{P}$ is extended with constraints that encode that only dep and/or con can be assigned to the zones originating from $a4$. Similarly, any refinement of $a7$ can only house op and/or hbs. The allocation from Fig. 1b is preferred because it has rank 3 and there is no feasible allocation with a lower rank. The rank is 3 because three conditionals from Δ (cf. Sec. 5) are violated: $a4$ and $a7$ mention

more than one functional area, and $a4$ has gates while housing *con* which does not need gates. By comparing these violations with those of the other allocations we can principally explain *why* the allocation from Fig. 1b is preferred. While the refinement of $a7$ after the first iteration leads to unambiguous zones already (zones $a7t$, $a7bt$, and $a7bb$ in Fig. 2b), the two new zones which replace $a4$ have to be split further in the next iterations. After iteration 4, we obtain two allocations which both are preferred. Each of them still has an ambiguous zone, either $a4lrb$ (cf. Fig. 2b) or $a4lrt$. Splitting these zones would lead to zone sizes under the defined threshold of 2%. Consequently, we stop the refinement process and display the allocations to the user (Line 8 of Alg. 1). At this point it can be more efficient to let the expert apply final refinements manually. A possible outcome is realized in the block layout visualized in Fig. 1a.

7 Conclusion

We proposed a knowledge-based hybrid reasoning system developed to assist warehouse designers during the planning of warehouse layouts. Given an empty floor plan of a warehouse and specific requirements regarding the desired material flow, our system computes "plausibly good" allocations of the functional areas which are relevant for the warehousing process, i.e., it produces so-called block layouts, which can then be filled with facilities in a second planning phase. Our system makes use of two well-established approaches from the field of knowledge representation and reasoning, namely answer set programming (ASP) and conditional reasoning with ranking functions. With ASP all feasible allocations of the functional areas with respect to an initial zoning of the warehouse are computed. Afterwards, the plausibilities of these allocations are rated based on a conditional knowledge base comprising logistical expert knowledge. Therewith, either feasible block layouts are produced or the computations pass into an iterative process with a refined zoning of the warehouse and an adaptation of the ASP program. Beyond its logistical benefits, such as speeding up the planning process and increasing transparency, our system demonstrates the potential of combining ASP and semantic conditional reasoning.

In future work, we want to broaden the evaluation of our approach including further comparisons of the produced block layouts of our system against human-designed layouts in order to fine-tune the system. We also want to combine it with automatizations of the fine-planning of the functional areas (cf. [30]). From the computer science perspective, we want to more strongly interconnect ASP and conditional reasoning, e.g., by excluding implausible allocations of functional areas before they have to be computed with ASP. Finally we plan to expand the interactive component of the system to allow the user to guide the search process using additional expert knowledge and efficiently find the most suitable layouts.

Acknowledgements This work was supported by Grant KE 1413/14-1 of the German Research Foundation (DFG) awarded to Gabriele Kern-Isberner.

References

1. Adams, E.W.: The logic of conditionals. Inquiry: An Interdisciplinary Journal of Philosophy **8**(1-4), 166–197 (1965)
2. Altuntas, S., Selim, H., Dereli, T.: A fuzzy DEMATEL-based solution approach for facility layout problem: a case study. Int. J. Adv. Manuf. Technol. **73**(5-8), 749–771 (Jul 2014)
3. Beyer, T., Göhner, P., Yousefifar, R., Wehking, K.H.: Agent-based dimensioning to support the planning of intra-logistics systems. In: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA). pp. 1–4 (2016)
4. Brewka, G., Eiter, T.: Preferred answer sets for extended logic programs. Artif. Intell. **109**(1-2), 297–356 (1999)
5. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977. pp. 293–322. Advances in Data Base Theory, Plemum Press, New York (1977)
6. Delgrande, J., Schaub, T., Tompits, H.: Logic programs with compiled preferences. In: Horn, W. (ed.) ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000. pp. 464–468. IOS Press (2000)
7. Dukic, G., Opetuk, T.: Warehouse Layouts, pp. 55–69. Springer London, London (2012)
8. Finetti, B.d.: La logique de la probabilité pp. 31–39 (1936)
9. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012)
10. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. Theory Pract. Log. Program. **19**(1), 27–82 (2019)
11. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The potsdam answer set solving collection. AI Commun. **24**(2), 107–124 (2011)
12. Gelfond, M.: Answer sets. In: van Harmelen, F., Lifschitz, V., Porter, B.W. (eds.) Handbook of Knowledge Representation, Foundations of Artificial Intelligence, vol. 3, pp. 285–316. Elsevier (2008)
13. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes). pp. 1070–1080. MIT Press (1988)
14. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**(3/4), 365–386 (1991)
15. Genesereth, M., Kao, E.J.: Relational Logic, pp. 63–81. Springer International Publishing, Cham (2017)
16. Gudehus, T.: Logistik 2. Springer Berlin Heidelberg (2012)
17. Kern-Isberner, G.: A thorough axiomatization of a principle of conditional preservation in belief revision. Ann. Math. Artif. Intell. **40**(1-2), 127–164 (2004)
18. Kovács, G.: Warehouse design - determination of the optimal storage structure. Acta Technica Corviniensis - Bulletin of Engineering (2017)
19. Lifschitz, V.: Answer Set Programming. Springer (2019)

20. Lin, Q., Wang, D.: Facility layout planning with shell and fuzzy ahp method based on human reliability for operating theatre. Journal of Healthcare Engineering **2019**, 1–12 (2019)
21. Nieuwenborgh, D.V., Vermeir, D.: Preferred answer sets for ordered logic programs. Theory and Practice of Logic Programming **6**(1-2), 107–167 (2006)
22. Pournaderi, N., Ghezavati, V.R., Mozafari, M.: Developing a mathematical model for the dynamic facility layout problem considering material handling system and optimizing it using cloud theory-based simulated annealing algorithm. SN Appl. Sci. **1**(8) (Aug 2019)
23. Pérez-Gosende, P., Mula, J., Díaz-Madroñero, M.: Facility layout planning. an extended literature review. International Journal of Production Research **59**(12), 3777–3816 (2021)
24. Rouwenhorst, B., Reuter, B., Stockrahm, V., van Houtum, G., Mantel, R., Zijm, W.: Warehouse design and control: Framework and literature review. European Journal of Operational Research **122**(3), 515–533 (2000)
25. Sakama, C., Inoue, K.: Prioritized logic programming and its application to commonsense reasoning. Artificial Intelligence **123**(1-2), 185–222 (2000)
26. Schäpers, B., Niemueller, T., Lakemeyer, G., Gebser, M., Schaub, T.: Asp-based time-bounded planning for logistics robots. Proceedings of the International Conference on Automated Planning and Scheduling **28**(1), 509–517 (Jun 2018)
27. Shouman, M., Nawara, G., Reyad, A., El-Darandaly, K.: Facility layout problem (flp) and intelligent techniques: a survey. In: 7th International Conference on Production Engineering, Design and Control, Alexandria, Egypt, February (2001)
28. Singh, S.P., Sharma, R.R.K.: A review of different approaches to the facility layout problems. Int. J. Adv. Manuf. Technol. **30**(5-6), 425–433 (Sep 2006)
29. Spohn, W.: The Laws of Belief - Ranking Theory and Its Philosophical Applications. Oxford University Press (2014)
30. Thevapalan, A., Wilhelm, M., Kern-Isberner, G., Kaiser, P., Roidl, M.: An interactive modelling environment for designing warehouse layouts based on ASP. In: Franklin, M., Chun, S.A. (eds.) Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023. AAAI Press (2023)
31. Wang, K., Zhou, L., Lin, F.: Alternating fixpoint theory for logic programs with priority. In: Lloyd, J., Dahl, V., Furbach, U., Kerber, M., Lau, K.K., Palamidessi, C., Pereira, L.M., Sagiv, Y., Stuckey, P. (eds.) Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings. LNCS, vol. 1861, pp. 164–178. Springer (2000)
32. Wei, X., Yuan, S., Ye, Y.: Optimizing facility layout planning for reconfigurable manufacturing system based on chaos genetic algorithm. Production & Manufacturing Research **7**(1), 109–124 (2019)
33. Wilhelm, M., Thevapalan, A., Kern-Isberner, G.: Prioritizing answer sets based on conditional expert knowledge. In: Franklin, M., Chun, S.A. (eds.) Proceedings of the Thirty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2023, Clearwater Beach, FL, USA, May 14-17, 2023. AAAI Press (2023)
34. Wunderle, A., Sommer, T.: Erfahrung und augenmaß zählen. Hebezeuge Fördermittel (2014)
35. Zhang, Z.: Introspecting preferences in answer set programming. In: Palù, A.D., Tarau, P., Saeedloei, N., Fodor, P. (eds.) Technical Communications of the 34th International Conference on Logic Programming, ICLP 2018, July 14-17, 2018, Oxford, United Kingdom. OASIcs, vol. 64, pp. 3:1–3:13. Dagstuhl (2018)