Interactive Layout Generation with a Diagrammatic Constraint Language[1]

Christoph Schlieder* and Cornelius Hagen**

*University of Osnabrück, Institute for Semantic Information Processing
cschlied@cl-ki.uni-osnabrueck.de
**University of Freiburg, Center for Cognitive Science
hagen@cognition.iig.uni-freiburg.de

Abstract. The paper analyzes a diagrammatic reasoning problem that consists in finding a graphical layout which simultaneously satisfies a set of constraints expressed in a formal language and a set of unformalized mental constraints, e.g. esthetic preferences. For this type of problem, the performance of a layout assistance system does not only depend on its use of computational resources (algorithmic complexity) but also on the mental effort required to understand the system's output and to plan the next interaction (cognitive complexity). We give a formal analysis of the instantiation space of a weakly constrained rectangle layout task and propose a measure for the cognitive complexity. It is discussed how the user's control of the presentation order of the different constraint instantiations affects the cognitive complexity.

1 Partially Unformalized Constraint Systems

One of the fields in which constraint-based approaches to spatial configuration have been particularly successful is the generation of document layouts. Assistance systems as the ones described by [17] and [7] allow the user to specify layout properties with a relational constraint language. The system then generates the layout by means of a constraint solver. However, not all layout problems permit this neat form of declarativity which completely hides the search aspect of constraint satisfaction from the user. Often, the user has to interact with the system in order to guide the search process. One rather trivial reason for combining machine reasoning with mental reasoning is given when the search heuristics produce suboptimal results. A human observer who visually inspects a machine-generated layout can sometimes provide precious hints about how to improve the quality of the solution. Of course, this type of *mental postprocessing* constitutes only a last resort. Therefore, in applications such as yellow page layout where design rules are easily formalized, much research effort is spent on eliminating the need for mental postprocessing.

A completely different situation arises in creative layout tasks. These tasks are gov-

1. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) in the priority program on spatial cognition (grant Str 301/5-2).

Ch. Freksa et al. (Eds.): Spatial Cognition II, LNAI 1849, pp. 198-211, 2000.

erned by constraints that, at least until now, have resisted complete formalization. Typically, the layouter is able to specify some of the constraints that a solution must satisfy, but these constraints do only partially capture his notion of a "good" solution. Design principles which blend functional considerations with esthetic preferences are known to be very difficult to verbalize, let alone to formalize in a constraint language [3]. However, the mental processing of the unformalized design principles is generally very efficient: a human layouter recognizes without difficulty a solution satisfying such constraints; usually he is also able to rate the quality of the solution.

This type of layout task can be characterized as *partially unformalized constraint problem (PUCP)*: Find a layout that satisfies simultaneously (1) a set F of spatial constraints specified in a constraint language L, and (2) a set U of mental constraints which cannot be expressed in L. The assistance system then helps the user to explore the solution space of the PUCP. Its main function consists in computing instantiations for the constraints in F and in providing suitable navigation operations for moving between instantiations. The user navigates through the instantiation space checking the constraints in U until a satisfying layout is found. In other words, the system acts as a medium which guarantees that during the navigation the user will not take layouts into consideration which violate any of the formalized constraints (Fig. 1).

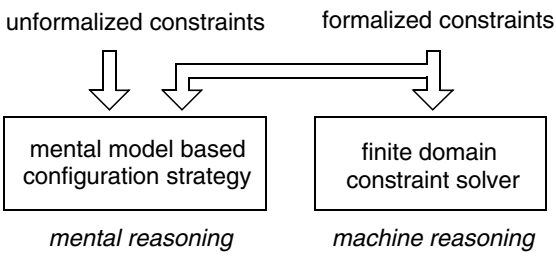

Fig. 1 Partially unformalized constraint problem

Because of the essential role that mental reasoning plays in solving a PUCP, a performance analysis of the assistance system cannot ignore the time that the user spends on evaluating a layout and planning the next interaction as it could be done in a mental postprocessing scenario. To mark this difference, we speak of interaction based on *mental coprocessing*.

Mental coprocessing is a special type of interactive constraint processing. It differs from other interaction schemes studied in the area of spatial constraint solvers in two ways:

(1) The user cannot provide the system with an instantiation. Mental and formal constraints combine to form a problem that is too complex to be solved mentally. This is in clear difference to the scenario described by Kurlander and Feiner [11] where constraints are inferred from example solutions specified by the user.

(2) The formal constraint system is weakly constrained. In other words, many instantiations exist and the problem consists in finding a way to present the large space

of alternative solutions to the user. Interactive constraint solvers typically rely on the assumption that the task is strongly constrained and a unique or no instantiation exits. A classical example is Borning's constraint solver ThingLab [2].

As a consequence, the problem for mental coprocessing consists in finding a good way to present a large instantiation space to the user. This problem has been addressed first not in spatial, but in temporal reasoning. Hoebel, Lorenzen, and Martin [8] visualizes the solutions found by their temporal planner, i.e. alternative plans satisfying a set of (formal) temporal constraints. Their approach amounts to simultaneously present all instantiations. Clearly, this is feasible only for very small spaces (< 100 instantiations).

The purpose of this paper is to study the case where a simultaneous presentation of all solutions is not feasible because of the size of the instantiation space. Our analysis refers to a concrete layout problem (section 2) which is based on a simple diagrammatic constraint language that is described in section 3. We argue that the cognitive complexity of a PUCP critically depends on the degree to which the user is allowed to control the order in which constraint instantiations are produced (section 4). A criterion for an optimal degree of cognitive control is formulated (section 5) and applied to the layout problem (section 6).

2 Constrained-Based Document Layout

Constraint languages for document layout allow to express relations between rectangular graphical objects such as textboxes and images. Fig. 2 shows some of the relational constraints used in the seminal work of Weitzman and Wittenburg [17]. These spatial relations describe the ordering of the boxes (e.g. left-of) or their alignment (e.g. top-aligned) without specifying metrical information. We consider a constraint language that encompasses all possible relations of this kind. The basic idea consists in projecting the configuration of rectangles onto the horizontal and the vertical axis thereby obtaining two configurations of intervals.

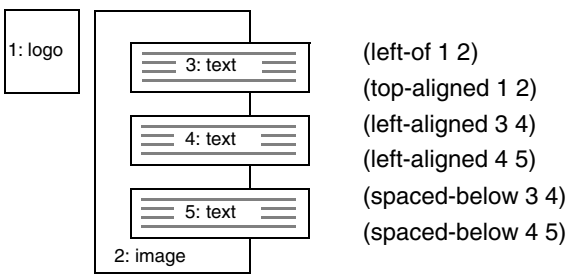

Fig. 2 Constraints for a document layout

Constraints between intervals are then expressed by means of a system of relations

that is widely used in AI research on temporal and spatial reasoning: the *interval relations* introduced by Allen [1]. The system $I = \{<, m, o, s, d, f, =, fi, di, si, oi, mi, >\}$ consists of the 13 relations between two intervals $X = [x_1, x_2]$ and $Y = [y_1, y_2]$ that can be distinguished in terms of the ordering of x_1, x_2, y_1, and y_2. Disjunctive information is represented by sets of interval relations. $X\ \{o,s,d\}\ Y$, for instance, is a shorter notation for $X\ o\ Y \vee X\ s\ Y \vee X\ d\ Y$. Every element of $\mathrm{INT} = 2^I$ can thus be interpreted as a *generalized interval relation*, i.e. a disjunction of interval relations. Inversion and composition are defined componentwise for relations from INT. Some relations deserve special attention: the universal relation **1** corresponding to I and the contradictory relation **0** corresponding to the empty set. The 13 interval relations are represented by singleton subsets that are called the *basic relations* of INT.

We use generalized interval relations for stating constraints between intervals. An *interval constraint system* for a set of intervals $\{X_1,...,X_n\}$ is an $n \times n$ matrix $C = (r_{ij})$ with entries $r_{ij} \in \mathrm{INT}$. The entry r_{ij} specifies the generalized interval relation that must hold between interval X_i and X_j. If the entry is **1** it does not constrain the position of the intervals — any relation could hold. In standard terminology, the r_{ij} constitute the constraint variables that are instantiated with values from the domain I. An assignment of values to the constraint variables of $C = (r_{ij})$ is given by a matrix $B = (b_{ij})$, whose entries are interval relations $b_{ij} \in I$ satisfying $b_{ij} \in r_{ij}$. We call B a consistent assignment or *instantiation* of C iff $\forall i \forall j \forall k\ b_{ik} \in (b_{ij} \bullet b_{jk})$. In other words, consistency means that the assignment is compatible with the composition table. Since the constraint variables range over a finite domain, namely the 13 interval relations, the satisfiability problem is decidable. Although for general interval constraint systems deciding satisfiability is known to be NP-complete, the problem becomes polynomial as soon as the relations of the constraint system are restricted to certain subsets of INT [13]. In these cases a simple $O(n^3)$ constraint propagation algorithm (path consistency) can be used to decide satisfiability, or — more interesting — to compute a constraint instantiation. In the general case the propagation algorithm is used as forward checking heuristic during backtracking [12].

3 A Simple Diagrammatic Constraint Language

As was already mentioned, constraints between two rectangles A and B are represented by constraints between the two pairs of intervals that result from projecting A and B onto the horizontal and vertical axis: A_x, B_x and A_y, B_y respectively. Formally, the *rectangle relation* $A\ r_x{:}r_y\ B$ with $r_x, r_y \in I$ holds iff $A_x r_x B_x$ and $A_y r_y B_y$ (cf. Fig. 3). The system of rectangle relations consists of $13 \times 13 = 169$ relations $R = \{<:<, <:m, <:o, ..., oi:>, mi:>, >:>\}$. In order to express disjunctive information, rectangle relations are generalized analogously to the interval relations. However, to guarantee tractability, we do not consider arbitrary relations from 2^R but only those that can be written as a product of generalized interval relations[2]. The product of $r_1, r_2 \in \mathrm{INT}$ is $r_1 \times r_2 := \{b_1{:}b_2 \mid b_1 \in r_1 \wedge b_2 \in r_2\}$. In this sense $\{f{:}d, f{:}o, m{:}d, m{:}o\} = \{f,m\} \times \{d,o\}$ is a *generalized rectangle relation* whereas $\{f{:}d, f{:}o, m{:}o\}$ is not. We denote the set of these

relations by REC. Because generalization is restricted, relations from REC can be written in product form, e.g. A {f,m}:{d,o} B for A {f:d, f:o, m:d, m:o} B (cf. Fig. 3).

The composition of generalized rectangle relations reduces to the composition of generalized interval relations. For relations $r_{11}{:}r_{12}$ and $r_{21}{:}r_{22}$ from REC $(r_{11}{:}r_{12}) \bullet (r_{21}{:}r_{22}) = (r_{11} \bullet r_{12}){:}(r_{21} \bullet r_{22})$. Now that composition is defined, constraint systems can be introduced in complete analogy to interval constraint systems: A *rectangle constraint system* for a set of rectangles $\{X_1,...,X_n\}$ is an $n \times n$ matrix $C = (r_{ij})$ with entries $r_{ij} \in$ REC. An assignment of values to the constraint variables of $C = (r_{ij})$ is given by a matrix $B = (b_{ij})$ whose entries are rectangle relations $b_{ij} \in R$ satisfying $b_{ij} \in r_{ij}$. The assignment B is an *instantiation* of C iff $\forall i \forall j \forall k \; b_{ik} \in (b_{ij} \bullet b_{jk})$. Note that because all relations are from REC, a rectangle constraint system can be solved by solving two independent interval relation systems.

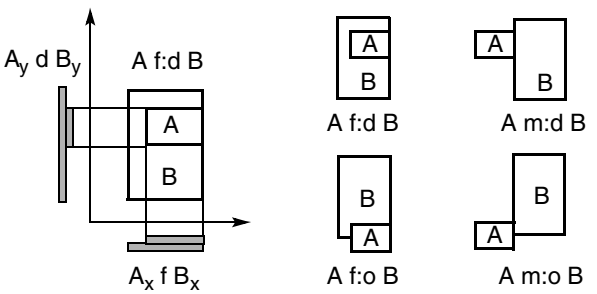

Fig. 3 Projection of layout boxes

Rectangle constraint systems differ from the finite domain constraint systems traditionally used for visual interaction. Constraint variables represent qualitative relations, not coordinates from a finite range of integers such as in [2] or symbolic constraints as in [7]. The type of qualitative abstraction underlying rectangle constraint systems can be described formally in the framework of relational algebras[3] [12]. Constraint-based reasoning in relational algebras has been studied thoroughly by AI research on *qualitative spatial reasoning* [4]. Most of the following analysis generalizes to reasoning with arbitrary spatial relational algebras.

4 Navigation in Instantiation Space

We consider a PUCP for which the set F of formal constraints is given by a rectangle constraint system. Only the user can check whether an instantiation of F also satisfies

2. Actually, more is needed for tractability. In each projection, the generalized interval relations must be members of a tractable subalgebra of 2^R (see [13]).
3. The technical difference between symbolic constraints such as those used in [7] and qualitative constraints is that the latter express information about relational composition and use path-consistency (not the weaker arc-consistency) as propagation method.

the additional set U of unformalized constraints. If $\Im$ denotes the set of instantiations of F, then the mental operation of checking the constraints in U realizes a mapping $m: \Im \to \{\text{true}, \text{false}\}$ where $m(I) = \text{true}$ if the instantiation $I \in \Im$ satisfies U, and $m(I) = \text{false}$ otherwise. In the following we regard this mental process as computationally intransparent. Although the details of the interaction between mental and machine reasoning can be designed very differently, all assistance systems for PUCP share the same *basic interaction cycle*. It consists of a loop which combines mental operations (step 1) and computations performed by the system (step 2):

repeat
  (1) Select a navigation operation $\omega \in \Omega$.
  (2) Find an alternative instantiation I_ω of F.
until $m(I_\omega)$ is true

Searching through the instantiations of F is realized by iterative execution of step 2. The user can influence search order through the choice of a navigation operation. In the document layout scenario, he could for example point at a layout box that the system should reposition, or, more specifically, drag the layout box to the place he prefers. In both cases, one option ω out of a set Ω of possible options is selected. However, the user has more control over the search process in the latter than in the former case. Quantitatively, this difference is reflected by a bigger cardinality of Ω: the number of layout boxes is multiplied by the number of positions in the layout grid. How much search control should the user be given? Obviously, more control is an advantage as long as it does not increase the cognitive complexity of the task. If the user executes c basic interaction cycles spending an average time T_ω on selecting the navigation operation then the *cognitive complexity*, i.e. the total time spent on mental reasoning, is given by

$$T_{\text{mental}} = c \cdot T_\omega$$

This means that there is a trade-off between spending time on evaluating different layouts generated by the system (c interaction cycles) and spending time on a single layout deciding how to modify it (T_ω navigation time). As a tool for describing the trade-off we introduce the *navigation graph*. The graph encodes what instantiation $I_\omega \in \Im$ is generated when the navigation operation $\omega \in \Omega$ is selected. Generally, this depends on the instantiation $J \in \Im$ to which the navigation operation is applied. In other words, each $\omega \in \Omega$ realizes a mapping $\omega: \Im \to \Im$. The navigation graph $N = (V, E)$ is specified by $V = \Im$ and $E = \{(I, J) \in \Im \times \Im \mid I = \omega(J), \omega \in \Omega\}$. Note that navigation graphs can become very large: $\Im$ can grow exponentially with n, the number of layout objects. Just think of the interval constraint system $C = (r_{ij})$ with entries $r_{ij} = \{<, >\}$ for $i \neq j$ and $r_{ii} = \{=\}$. Its navigation graph has $n!$ vertices. The number of edges of a navigation graph can vary considerably. Fig. 4 illustrates this with three regular graphs for $|\Im| = 2^n$: the cycle C_{2^n}, the hypercube H_{2^n}, and the complete graph K_{2^n}.

C_{2^n} and K_{2^n} describe the two extremes where the user exerts no or complete search control respectively. In the first case, the assistance system generates instantiations in a

fixed order. As the user is forced to follow the tour C_{2^n}, he spends constant time for the navigation decisions but it may take him as many as $O(2^n)$ interaction cycles to reach his goal (diameter of C_{2^n}).

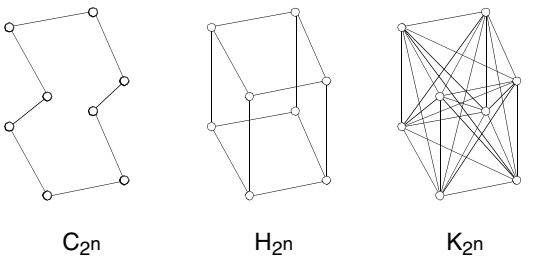

Fig. 4 Different types of navigation graphs

With K_{2^n} the user has complete search control. Any instantiation can be reached in a single interaction cycle. However, he must spend considerable time to decide which of the $O(2^n)$ navigation alternatives (degree of K_{2^n}) to choose. A compromise between the two extremes is realized by H_{2^n} which implements partial search control. There are $O(n)$ different navigation operations between which the user has to decide and with $O(n)$ interactions he reaches any instantiation.

navigation graph	*diameter*	*degree*	*search control*
C_{2^n}	$O(2^n)$	$O(1)$	none
H_{2^n}	$O(n)$	$O(n)$	partial
K_{2^n}	$O(1)$	$O(2^n)$	complete

Fig. 5 Navigation graph and search control

5 Lessons from Mental Model Theory

The optimal balance between diameter and degree of the navigation graph should minimize the cognitive complexity of the whole task $T_{\text{mental}} = c \cdot T_{\omega}$. Clearly, the optimum depends on how T_{ω} relates to the degree, or equivalently, to the cardinality of Ω. Our argument is based on (1) the formal analysis of the problem space given in the previous section, (2) empirical evidence about the use of mental models in spatial reasoning. We briefly summarize the relevant findings.

Reasoning with spatial constraints has been studied for more than 20 years in the *spatial-relational inference* paradigm (see [5]). Tasks in this paradigm consist of the non-universal entries of a constraint matrix which are presented in verbal form to sub-

jects of the experiment. In our case: "The red interval touches the green interval from the left" for X m Y. Subjects are then asked to specify instantiations for certain or all constraint variables ("Which relation must/could hold between the blue and the yellow interval?"). It was found that most human reasoners adopt a model-theoretic rather than proof-theoretic strategy for solving spatial relational inference tasks. They try to build alternative spatial layouts in visuo-spatial working memory and check which relations hold in the layouts rather then proving with inference rules that certain relations must/could hold. This finding led Johnson-Laird to formulate *mental model theory*, an explanatory framework unifying results about model-based reasoning strategies [9]. Results from the experiments conducted by Rauh and Schlieder [14] clearly indicate that interval constraint systems are mentally solved by reasoning with mental models. Human reasoners seem not to be able to effectively use constraint propagation — a mental proof strategy. Instead, they immediately try to find an instantiation by means of simple spatial layout strategies. The layout strategies for constructing interval configurations have been analyzed and described in [15]. It turns out that they are very efficient but incomplete, i.e. the strategies do not always produce an interval configuration. If the mental layout strategies succeed, they produce a first mental model, the *preferred mental model,* which constitutes the starting point for the further reasoning process. Interestingly, subjects agree considerably on which instantiation of an interval constraint system they prefer [10]. Fig. 6 shows all instantiations of the interval constraint system that consists of three non-universal constraints A di B, $B > C$, and C m D. According to the data of an experiment from [14], interval configuration 37 constitutes the mental model preferred by most subjects. Only few subjects preferred the somewhat similar configuration 36. None of the other configurations acted as preferred mental model .

Human reasoners seem not to possess different sets of layout strategies for constructing different instantiations of an interval constraint system. They obtain alternative instantiations by transforming the preferred mental model. As a consequence, cognitive complexity increases *linearly* with the number of mental models constructed, or equivalently, with the number of transformations applied. If the task is solved by mental reasoning without the possibility to externalize working memory contents then only few of the possible instantiations are found. This is where assistance systems enter. They provide operations for navigating through instantiation space that support the process of *mental model transformation*. The system checks which transformations produce an instantiation and gives the user a complete overview of all applicable transformations. In order to decide which navigation operation to take next, the user must anticipate the result of this operation and evaluate whether it brings him closer to a layout that satisfies the unformalized constraints. Generally, he can only partially anticipate the result. Based on a mental model of part of the layout, it is often impossible to see that a local transformation of the layout has global consequences.

Moving a single layout box to a new position can lead to the repositioning of all other layout boxes. Because the transformation of mental models is the basic reasoning strategy at the user's disposition for anticipating the effects of his navigation operation and because the cognitive complexity of mental model transformation increases linearly with the number of models that are constructed, we expect to find $T_\omega = O(|\Omega|)$.

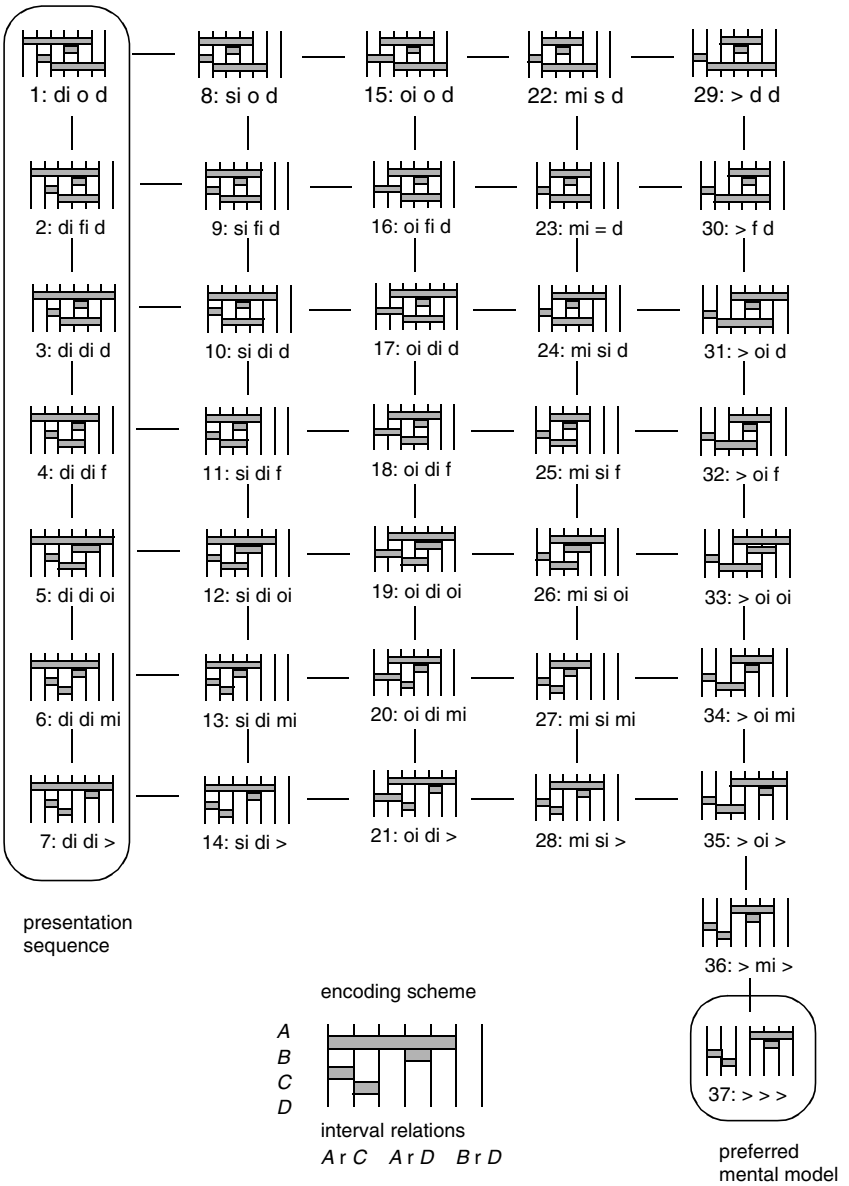

Fig. 6 Instantiation space for the interval constraint system A di B, $B > C$, C m D

6 Neighborhood Navigation Operations

As we have argued in the last section, $T_\omega = O(|\Omega|)$. In order to obtain a navigation graph that minimizes the cognitive complexity, we will have to choose Ω in such a way that $|\Omega| = O(n)$ where n denotes the number of layout objects (rectangles, intervals). Additionally, the navigation operations should correspond to transformations of mental models that can be computed with little cognitive effort. On the level of single interval relations, cognitively simple transformations are known to exist. They implement the *conceptual neighborhood* of interval relations [6]. Interval relations r_1 and r_2 are said to be conceptual neighbors if a configuration of two intervals X and Y satisfying $X\ r_1\ Y$ can be continuously transformed into a configuration of intervals X' and Y' satisfying $X'\ r_2\ Y'$ such that during the transformation no configuration arises in which a relations different from r_1 and r_2 holds. For example, there is no continuous transformation of $X < Y$ into X o Y which avoids a stage where X m Y: the relations < and m as well as m and o are conceptual neighbors whereas < and o are not. The edges of the graph in Fig. 7 specify conceptual neighborhood for the interval relations

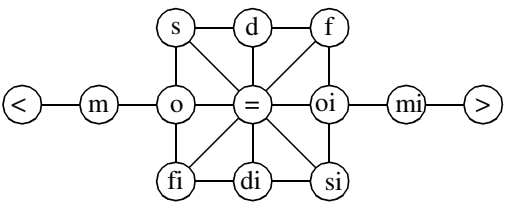

Fig. 7 Conceptual neighborhood graph

We generalize this idea to configurations of more than two intervals. Distance in the neighborhood graph of Fig. 7 is denoted by $d(r_1, r_2)$, e.g. $d(\mathrm{m}, \mathrm{si}) = 4$. For two instantiations $A = (a_{ij})$ and $B = (b_{ij})$ of an interval constraint system the *conceptual distance matrix* is defined as

$$\Delta(A, B) = (d(a_{ij}, b_{ij})).$$

The most strict notion of *instantiation neighborhood* considers instantiations A and B neighbors iff in the conceptual distance matrix $\Delta(A, B)$ only two entries, r_{ij} and r_{ji}, corresponding to the relation between interval X_i and X_j are 1 whereas all other entries vanish. In the instantiation space of Fig. 5 configuration 4 and 11 are conceptual neighbors; 4 and 18 are not. Transformations that map an instantiation into a neighboring one seem good candidates for navigation operations that are cognitively simple to compute. It is not difficult to show that the maximum number of transformations into conceptually neighbored configurations is bounded by $8 \cdot n = O(n)$. This means that the number of neighbors of an instantiation is small enough to allow a mental evaluation of all navigation alternatives. *Neighborhood navigation* relies on a set of navigation operations that move only from an instantiation to its conceptual neighbors.

Actually, a slightly more general notion of instantiation neighborhood was used to compute the navigation graph in Fig. 6. With our definition, the edge between instantiation 15 and 22 does not correspond to a neighborhood transformation since relations between more than two intervals change at once. However, the notion of neighborhood transformations can be generalized to capture such cases. Neighborhood navigation implements a local navigation scheme. It is not only local with respect to the instantiation space but also local with respect to the geometry of each single instantiation. Neighborhood transformations only change relations between objects that are close to each other.

In order to change relations between layout objects that are far apart, a sequence of transformations is needed. For this purpose, *presentation sequences* where all neighborhood transformation refer to the same interval are especially useful. Fig. 5. shows an example consisting of a sequence involving the instantiations 1, 2, ..., 7. The algorithmic problem of generating presentation sequences consists in modifying a path-consistency constraint solver for relational algebras such that it generates instantiations in a particular ordering. To achieve this goal the constraint variables must be assigned values in a particular order. A best first strategy can be used for this purpose: (1) assign variables that denote relations which involve the selected interval first, (2) assign values that denote relations which are conceptual neighbors of the last assignment first.

Because of their local nature, neighborhood navigation operations are easily visualized. Fig. 8 illustrates the structure of the visual interface of a system assisting the user to solve a PUCP in the document layout scenario. The formalized constraints are symbolized by flag icons which stand for rectangle relations: b1 {<}:{f} b2, b3 {s}:{>} b4, b4 {s}:{>} b5. To navigate through the instantiation space, the user selects a layout box. The system then computes the applicable neighborhood transformations and visualizes them using arrow icons. By selecting one of the arrow icons, the user communicates his navigation decision to the system which alters the layout respecting the formalized constraints. A prototype assistance system based on neighborhood navigation has been implemented. It serves as a testbed for evaluating the different sets of navigation operations that result from using definitions for instantiation neighborhood of different strength. Which notion of neighborhood minimizes cognitive complexity is an empirical question. However, the analysis of the cognitive complexity of partially unformalized constraint systems presented in this paper provides the conceptual framework for studying the issue.

7 Conclusion

The analysis of the cognitive complexity of navigation in instantiation space implies that the user should be given a number of navigation choices that does not increase more than linearly with the number of layout boxes. One way to implement this principle is to allow the user to select a box and to let the system generate an appropriate presentation sequence. This raises the issue of efficiently controlling the order in which a path-consistency constraint solver computes instantiations. In our prototype system,

this is done by using a best-first strategy for ordering variable-value assignments. Future work will move beyond single presentation sequences and try to formulate heuristics which allow to decompose a connected component of the instantiation space into sequences of presentation sequences.

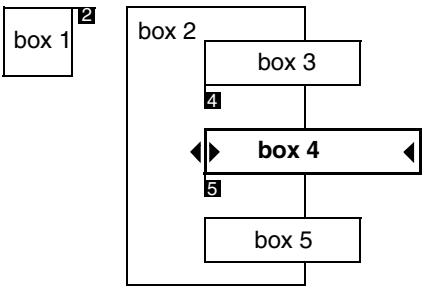

Fig. 8 Visualization of navigation operations

We conclude with a discussion of the limitations of local navigation and a brief sketch of how to overcome them. The most obvious limitation is due to the fact that, in general, the instantiation space consists of several connected components, not just one as in our example. Local navigation allows to navigate within a component, but since it moves only to conceptually neighboring instantiations, it will never reach any other component. A second limitation of local navigation consists in its relative slowness. With a single interaction step, the user can move only at unit distance from the present position in the navigation graph. It seems natural to provide the layout assistance system in addition to local navigation operations with operations for *global navigation*. Such operations should permit the user to move with a single interaction step distances $d > 1$ including the case $d = \infty$ which corresponds to a movement between connected components.

As we have seen, local navigation can be realized on the interface level by simply selecting a layout box. Many interactive constraint systems provide for a more complex type of graphical interaction such as the dragging and dropping of a layout box (e.g. [2], [7], [11]). Although these systems are designed for iteratively finding a solution for a strongly constrained layout task, global interactions (drag and drop) are useful also in the context of weakly constrained tasks. Selection of a layout box implements navigation within instantiation space while dragging can temporally cause navigation to leave instantiation space and to enter configuration space, i.e. the unconstrained configurations of layout boxes. *Configuration space navigation* raises a computational problem that was not present in *instantiation space navigation*. When the user produces a configuration that violates some layout constraints, then the layout assistance system must generate the instantiation that lies closest to the configuration under a suitable similarity metric on the configuration space.

Since the system's behavior becomes more complex, we expect that it is more difficult for the user to mentally anticipate the effects of his actions in configuration space navigation than in instantiation space navigation. More precisely, configuration space navigation requires the user to make a kind of analogical inference. The user knows the effect of a navigation operation *n* (dragging a particular box to a particular position) on a specific instantiation i_1: it causes the system to produce instantiation i_2. Currently, the system is displaying instantiation i_3 and the user has to infer what instantiation i_4 the system produces when *n* is applied. In other words, the user has to solve an analogy of the type $i_1 : i_2 = i_3 : i_4$. We plan to study this type of analogical inference in its connection of global navigation strategies in the near future.

References

1. J. Allen, Maintaining knowledge about temporal intervals, *Communications of the ACM*, 26, pp. 832-843, 1983.
2. A. Borning, Graphically defining new building blocks in ThingLab, In: E. Glinert (ed.) *Visual programming environments: paradigms and systems*, IEEE Computer Societey Press: Los Alamitos, CA, 450-469, 1990.
3. R. Coyne, M. Rosenman, A. Radford, M. Balachandran and J. Gero, *Knowledge-based design systems*, Addison-Wesley, Reading MA, 1990.
4. A. G. Cohn, Qualitative spatial representation and reasoning techniques. In: Proceedings KI-97: Advances in Artificial Intellligence, Springer, Berlin, pp. 1-30, 1997.
5. J. Evans, S. Newstead and R. Byrne, *Human reasoning: The psychology of deduction*, Lawrence Erlbaum, Hillsdale, NJ, 1993.
6. C. Freksa, Temporal reasoning based on semi-intervals, Artificial Intelligence, 54, pp. 199-227, 1992.
7. W. Graf, A. Kroender, S. Neurohr and R. Goebel, Experience in integrating AI and constraint programming methods for automated yellow pages layout, *Künstliche Intelligenz*, *2*, 79-85, 1998.
8. L. Hoebel, W. Lorenzen and K. Martin, Integrating graphics and abstract data to visualize temporal constraints, *SIGART Bulletin*, *9*, 18-23, 1998.
9. P. Johnson-Laird and R. Byrne, *Deduction*. Lawrence Erlbaum, Hillsdale, NJ, 1991.
10. M. Knauff, R. Rauh and C. Schlieder, Preferred mental models in qualitative spatial reasoning: A cognitive assessment of Allen´s calculus. In *Proceedings Conference of the Cogntive Science Society*, Lawrence Erlbaum, Mahwah, NJ, pp. 200-205, 1995.
11. D. Kurlander and S. Feiner, Inferring constraints from multiple snapshots, *ACM Transactions on Graphics*, 12, 277-304, 1993.
12. P. Ladkin and A. Reinefeld, Fast algebraic methods for interval constraint problems. *Annals of Mathematics and Artificial Intelligence*, 19, 1997.
13. B. Nebel and H. Bürckert, Reasoning about temporal relations: A maximal tractable subclass of Allen´s interval algebra. *Journal of the ACM*, 42, pp. 43-66, 1995.
14. R. Rauh and C. Schlieder, Symmetries of model construction in spatial relational inference, In *Proceedings Conference of the Cogntive Science Society*, Lawrence Erlbaum, Mahwah, NJ, pp. 638-643, 1997.

15. C. Schlieder and B. Berendt, Mental model construction in spatial reasoning: A comparison of two computational theories. In U. Schmid, J. Krems and F. Wysotzki (Eds.). *Mind modelling: A cognitive science approach to reasoning, learning and discovery*. Pabst Science Publishers, Berlin, 1998.
16. C. Schlieder, Diagrammatic transformation processes on two-dimensional relational maps, *Journal of Visual Languages and Computing*, 9, pp. 45-59, 1998.
17. L. Weitzman and K. Wittenburg, Grammar-based articulation for multimedia document design, *Multimedia Systems Journal*, 4, pp. 99-111, 1996.