Predicate Projection in a Bimodal Spatial Reasoning System

Samuel Wintermute and John E. Laird

University of Michigan
2260 Hayward St.
Ann Arbor, MI 48109-2121
{swinterm, laird}@umich.edu

Abstract

Spatial reasoning is a fundamental aspect of intelligent behavior, which cognitive architectures must address in a problem-independent way. Bimodal systems, employing both qualitative and quantitative representations of spatial information, are efficient and psychologically plausible means for spatial reasoning. Any such system must employ a translation from the qualitative level to the quantitative, where new objects (images) are created through the process of predicate projection. This translation has received little scrutiny. We examine this issue in the context of a bimodal spatial reasoning system integrated with a cognitive architecture (Soar). As part of this system, we define an expressive language for predicate projection that supports general and flexible image creation. We demonstrate this system on multiple spatial reasoning problems in the ORTS real-time strategy game environment.

Introduction

Representations of spatial relations employed in symbolic systems are qualitative in nature. For example, in the blocks world, problems are solved by manipulating predicates like *on(a, b)* or *clear(c)*. These are qualitative properties, as opposed to quantitative properties such as the *x, y* coordinates of each block. Qualitative reasoning is a powerful approach, but the poverty conjecture posited by Forbus et al. (1991) states that qualitative reasoning alone might not be sufficient in the general case. Bimodal systems, where spatial reasoning occurs through the interaction of qualitative and quantitative representations, have been proposed as remedies to this problem (Chandrasekaran et al. 2004). Additionally, bimodal approaches have been proposed for explaining human mental imagery phenomena (Kosslyn et al. 2006).

A high-level diagram of our overall approach to bimodal representation is shown in Figure 1. Soar (Lehman et al., 1998) is our cognitive architecture at the top of the figure. Soar maintains mostly symbolic qualitative representations of the current situation. Below Soar is the diagram module, which uses quantitative representations. Soar can extract qualitative properties of objects from the diagram, or insert new objects (*images*) in the diagram. Both of these processes involve translation between qualitative and

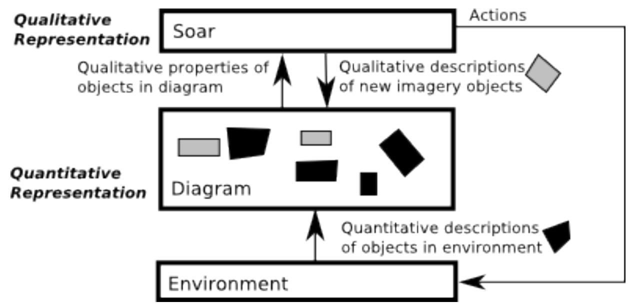

Figure 1. A bimodal spatial reasoning system.

quantitative representations. Following Chandrasekaran (1997), we call the first process *predicate extraction*, and the second *predicate projection*.

Many forms of spatial problem solving follow a loop where the agent inserts an image in the diagram, extracts properties of that image relative to other objects in the diagram, reasons over that information to create a new image, etc., until the problem is solved. For example, consider the minimalist system in Figure 2. The goal is to find a position for image **C** so that it does not intersect with **A** or **B**. The agent achieves this by repeated predicate projection, where it inserts images in the diagram, and predicate extraction, where it receives qualitative descriptions (whether the images intersect) of the current state. To support these processes, images created by the symbolic system are represented and processed identically to perceived environmental objects (except for annotations that distinguish between them) so that perception and imagery share a common buffer.

An alternative is to integrate qualitative and quantitative representations that they co-exist in Soar's working memory. However, there would be nothing to be gained by

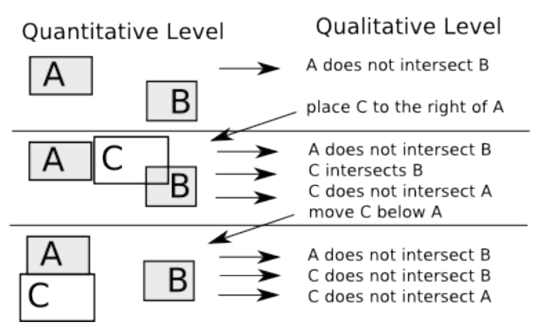

Figure 2. Simple bimodal spatial problem solving.

such a tight integration. Soar's existing reasoning involves matching, retrieving, and combining symbolic structures, while predicate extraction and projection require complex numeric calculations that would not intermix with symbolic processing or behavior control.

The translation from objects in a diagram to qualitative properties (predicate extraction) has been well studied (Cohn & Hazarika 2001). This is a translation from the specific to the general: relations such as 'object A intersects object B' are extracted from a diagram, condensing objects that share similar properties into well-defined qualitative categories. In this paper, we will examine the opposite problem, that of using a qualitative description to create a quantitatively represented image in the diagram. While sharing many similarities, this process is fundamentally different from predicate extraction, as it translates from the general to the specific.

Predicate Projection

Creating a new image involves translating a qualitative representation of the image (present in Soar) to a quantitative representation in the diagram. This problem has not been as well studied as predicate extraction (Chandrasekaran 1997). We call the qualitative representation of a new image the *description* of that image. Our goal is to create a system with broad applicability, which requires a qualitative language for describing new images that is as expressive and general as possible. Broadly, there are two kinds of possible descriptions: direct and indirect.

Indirect Descriptions

An *indirect description* for a new image is constructed from the same kind of abstract predicates as are extracted from the diagram. A description is a set of one or more predicates, such as "the image is to the right of object **A** and the left of object **B**". These predicates are *constraints*, each of which narrows down the space of potential images. Even with multiple constraints, though, there still may be an infinite number of images that satisfy those constraints.

For example, the image on the left of Figure 3 is described as 'shaped like object **E**, located inside of object **D**, and outside of object **C**'. This might describe an infinite number of images (top left of the figure), a finite number of images (middle left), or no images (bottom left), depending on the details of **C**, **D**, and **E**. Feeding the abstract predicates extracted from a diagram back as image constraints will usually result in an infinite set of potential images, of which the original is but one. Due to this property, descriptions of this form are *underdetermined*. In the general case, a set of constraints may result in an under-constrained problem (as in the top left of the figure), a critically constrained problem (middle left), or an overconstrained problem (bottom left).

To support efficient spatial reasoning, all objects in the diagram must have a unique representation there, grounded

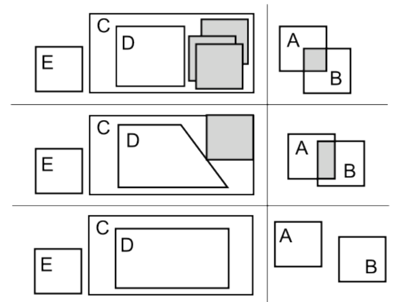

Figure 3. Directly vs. indirectly described images. Left: indirect image 'inside C, outside D, shaped like E'; Right: direct image 'intersection of A and B'.

in specific quantitative values (such as coordinates). If the description of an image is underdetermined, exactly one image must be created by the projection process, either via an arbitrary choice, or (as we will see) using more information to select a single image.

Direct Descriptions

In contrast to indirect descriptions, a *direct description* is always unambiguous – it describes only one image. An example is 'the image is a straight line from point **A** to point **B**'. For any diagram that has points **A** and **B**, there is exactly one such line. Other descriptions are those such as 'the image is the intersection of objects **A** and **B**'. For any convex objects **A** and **B** in the diagram, the image may or may not exist, depending on their positions in the diagram, but if it does exist, there is only one. Thus, a direct description describes exactly one image, not a category of images. An example is shown on the right of Figure 3 where we are describing the region of intersection between two existing objects, **A** and **B**. In each of the top two cases, the description "the intersection of **A** and **B**" corresponds to a single image, which varies based on **A** and **B**. In the bottom case, no such image exists.

Adding More Information to Indirect Descriptions

Since our goal is an image description system that is as expressive as possible, we consider whether more information can be added to indirect descriptions. Adding constraints is not always sufficient because there is no guarantee that the result will be single image – adding a single constraint may change an under-constrained problem into an over-constrained problem. Since an over-constrained image description is usually undesirable, images described solely with constraints will often be under-constrained. It is easy to choose randomly from the possible images, but in some cases, additional knowledge can be available to select among the alternatives.

One approach is to reason over the images meeting the constraints and use qualitative properties to select a single image. This appears difficult because there can be an infinite number of such images. However, it is possible to consider qualitative properties that select images that are 'extreme' in some way relative to the other possible images, and where the extremes can be computed without explicitly representing the entire set. For example, if we select the potential image which is nearest to a given object, we are likely to greatly reduce the number of images under consideration, and (as we will see) this selection can be efficiently implemented.

This suggests a two-stage process: first, the constraints are applied to the diagram to find the set of images fitting them, then *preferences* are applied, which describe properties that the final image should have compared to the other potential images, such as 'nearest to object **A**'. Applying a preference will not always result in a single image (many potential images could be equidistant to **A**), but it will usually reduce (and never increase) the number of potential solutions. Applying a preference will *never* result in an overconstrained problem. In this way, preferences are fundamentally different from constraints.

Since applying preferences will not always reduce the problem to a single solution, the process can be extended to any number of preference applications to refine the image. The process may not always result in one image, and the system will be forced to choose arbitrarily, but the specification of preferences allows for much more precise indirect qualitative descriptions of images.

The SRS Spatial Reasoning System

SRS (Spatial Reasoning for Soar) was developed to enhance Soar's spatial reasoning ability, creating the bimodal system in Figure 1[2]. For image creation, both direct and indirect descriptions (with constraints and preferences) were implemented. Tables 1 and 2 show the available image description predicates. All descriptions both refer to and describe objects, and can be composed together – for example, it is legal to create an image that is outside of the hull of object A and B, a constraint expressed in this notation as *outside(hull(A,B))*.

Image Description	Meaning
hull($O_1..O_n$)	Image is the convex hull of objects $O_1... O_n$.
intersection($O_1..O_n$)	Image is the geometric intersection of convex objects $O_1... O_n$.
scaled(O,<amount>)	Image is object O, expanded by <amount> units in every direction.
rectangle(<width>, <height>)	Image is an axis-aligned rectangle with the given dimensions.
line(O_1,O_2)	Image is a line intersecting the centroids of objects O_1 and O_2.
perpendicularLine (O_1,O_2)	Image is a line intersecting the centroid of O_1, and perpendicular to the longest edge of O_2.
indirect($C_1..C_n$, $P_1..P_m$)	Image is indirectly specified, based on constraints $C_1...C_n$ and preferences $P_1...P_m$. Note that preferences are order-dependent. P_1 is applied first, P_m last.

Table 1. Direct image description predicates

Name	Type	Meaning
shapedLike(O)	constraint	Image has the same shape as object O.
inside(O)	constraint	Image is located entirely inside object O.
outside(O)	constraint	Image does not intersect object O.
near(O)	preference	Image must be chosen such that the distance to O is minimal.
farFrom(O)	preference	Image must be chosen such that the distance to O is maximal.

Table 2. Indirect image constraints and preferences

The indirect image descriptions are implemented in SRS such that all of the possible locations for a new image are represented in a *possibility space* - a set of geometric structures that encompass all points where the image could be placed while meeting the description. The constraints and preferences can all be mapped onto simple geometric operations modifying this possibility space.

In SRS, the environment is bounded by a polygon. Before any constraints are applied, the initial possibility space is this polygon, and the initial image is a point located somewhere inside it. Applying constraints and preferences results in modifications of this space in predictable ways. For example, the *inside(O)* constraint transforms the possibility space to be the intersection of the prior possibility space and object O. Applying a *shapedLike* constraint transforms the space from the valid locations for a point to the valid locations for the centroid of a two-dimensional object, which can be accomplished through a simple geometric operation (a Minkowski sum).

Preferences also operate on this possibility space. For the *near* preference, the space is changed in one of several ways: if the object that the image is *near* to intersects the possibility space, the new possibility space is the intersection of the space and the object (which is the region where distance is at its absolute minimum, 0). Otherwise, the image must lie at the edge of the possibility space, and the space can either be reduced to a single point, or, if it has an edge that lies equidistant (and closest) to the object, the possibility space is reduced to a line. Once all constraints and preferences are applied, an arbitrary point in the possibility space is chosen as the location of the image.

[2] The only Soar-dependent aspects of SRS are the low-level interfaces used to connect to Soar.

Considering how these constraints and preferences are implemented also partly indicates *why* they are appropriate to implement. Outside of being very general properties of objects, they all correspond to efficient geometric operations in our possibility space representation. As we will demonstrate, their generality allows them to be applicable in many contexts; however, that would be irrelevant if the process of applying a predicate was NP-complete. The appropriateness of any given predicate is then partially a function of the underlying implementation, but the nature of the predicates used in general (constraints, preferences, direct descriptions) is not.

The system also extracts qualitative predicates from the diagram, including RCC relationships (Cohn et al. 2001), orientation relationships such as 'object A is to the right of object B', and distances between objects. Although distance is not a qualitative property, it is typically reasoned over in Soar by using an operator such as less-than, resulting in qualitative properties such as 'object A is closer to object B than object C is'.

The ORTS Environment

This system was applied to several problems that arise in creating a complete agent in the ORTS real-time strategy computer game (Buro & Furtak 2003). Real-time strategy (RTS) games are class of computer games characterized by continuous action, multiple players, control of multiple units, resource economies, and complicated planning and execution. Popular RTS games include StarCraft, WarCraft, and Command and Conquer. The goal of the game is to eliminate an opponent by exploring, gathering resources, building structures, and building an army.

In an RTS, the player is not embedded as an entity in the world, but is a commander viewing the world from above. In ORTS, the world is perceived as a two-dimensional map of convex polygons. A typical game operation might involve the player commanding a worker unit to build a building. To do this, a human player would select a worker unit and click on the location of the new building in the GUI. The worker would then autonomously move to the location and create the building. Soar has a similar interface to the game. For Soar to command a unit to build somewhere, it must generate an *x, y* coordinate of that location. Since imagery and perception share a common buffer in our system, this is accomplished by Soar creating an image of the new building, and commanding a unit to build at the location of that image. Most other game actions have a similar structure – the location of the action must be visualized, and used as part of the command issued to the environment.

Implemented Agents

Evaluation of a system such as SRS is difficult, since our goal is to extend the range of problems Soar can address, rather than directly improve performance. To demonstrate this range, we will describe how images are created and used in the context of three problems in ORTS.

Base Layout. In this problem, the agent must find a location for a new building. This location will become the center of the new building, and no part of that building can intersect another object. In addition to finding a large enough empty space, many other details must be considered. In most cases, the new building should be close to existing buildings, so that it is easier to defend the buildings and any movement between bases is minimized. Buildings have properties specific to their type that influence where they should be placed. Some buildings are able to defend against attack, so they should be placed toward the enemy, while others are exceptionally weak and should be away from the enemy. A simplified scenario is shown in Figure 4, where the new building is weak, and must be hidden. These are the objects present:

A: Existing buildings owned by the player (real objects)
B: Existing buildings owned by the enemy (real objects)
C: Player's base: an image, *hull*($\boldsymbol{A}_1..\boldsymbol{A}_n$).
D: Enemy base: an image, *hull*($\boldsymbol{B}_1..\boldsymbol{B}_n$).
E: New building location: an image,
*indirect(rectangle(*w,h*), outside(**C**), near(**C**), farFrom(**D**)),*
where w and h are the dimensions of the new building.

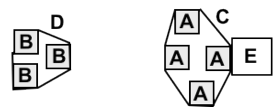

Figure 4. A base layout problem.

Once the final image (**E**) is placed, the agent can reparse the information from the diagram, and reason accordingly. For example, some object not previously considered could intersect **E**, in which case the agent should modify **E**'s description to be outside of that object. After a collision-free image is created, it can be used in a command to create the actual building. This is a form of the spatial problem-solving loop described in the introduction.

A complete base layout agent has been developed, with Soar using knowledge specific to each building type. Since the knowledge Soar applies to building placement maps directly onto the predicates available in SRS, the agent solves this problem very well.

Obstacle Avoidance. In this problem, the agent must determine a waypoint for a moving unit to divert around an obstacle. This is the base functionality upon which general path finding can be built. Path finding is a well-studied problem with a plethora of algorithms for different situations. In an RTS game, the interface software takes care of it in most cases, but there are situations where a human player must manually find a path, such as when knowledge needs to be taken into account that the built-in path finder cannot use (such as that the path must have a wide berth around a particular enemy). For our purposes, path finding is an interesting problem to address not only to account for this case, but also to determine if our general approach to spatial reasoning can address a problem that has been solved with problem-specific techniques.

We will demonstrate one component of our complete path-finding agent using the abstract obstacle avoidance

problem in Figure 5. These are the objects present:
A: The moving unit that is to be controlled (a real object)
B: The obstacle (a real object)
C: The unit at its destination: an image, specified based on objects not in the figure
D: The straight-line path for **A** to follow to **C**: an image, *hull(**A**,**C**)*.
E: An expanded version of the obstacle: an image, *scaled(**B**, x)*, where x is a small buffer amount.
F: Line image, *perpendicularLine(**B**, **D**)* .
G: Waypoint image,
*indirect(shapedLike(**A**), outside(**E**), near(**F**), near(**D**))*.

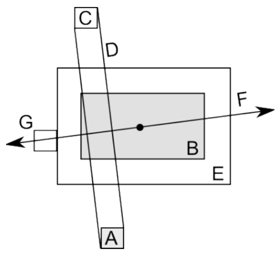

Figure 5. An obstacle avoidance problem.

Objects **E** and **F** are only used in the description of object **G**, so they do not need to be explicitly created. In this case, the description of **G** is:

*indirect(shapedLike(**A**), outside(scaled(**B**,x)), near(perpendicularLine(**B**,**D**)), near(**D**))*.

An image created with this compact description is equivalent to one created by the step-by-step process of creating images **E**, **F**, and **G** in order, but allows the agent to keep track of less information – images **E** and **F** are hidden from it.

After creating image **G**, the agent can then carry out more imagery and qualitative extraction procedures to determine the suitability of the proposed waypoint (such as checking whether the path to the waypoint collides with any other obstacles), eventually building up to a path finding algorithm.

A full agent was implemented using this approach to path finding, and was run on several hundred problem instances, with varying number of randomly placed obstacles between the initial location of the unit and the destination. The algorithm greedily moves toward the goal, diverting around obstacles if they intersect the path from the unit to the goal (as in Figure 5). Some capability is present to divert around groups of obstacles if a waypoint is unreachable. The agent's paths were compared to those found by the path finding system built into ORTS, with the goal being to determine if a path finding agent implemented in our system can solve the same problems as the conventional approach, and how well it can solve them.

An agent using this approach for generating waypoints worked well on simple problems, often finding shorter paths than the (non-optimal) built-in pathfinder, but it could fail on more complicated problems where backtracking was required. The reason for these failures was not that the agent did not posses powerful enough mechanisms for image creation, but that it used the greedy search technique. Additional work is required to completely address pathfinding by extending the high-level strategy employed. The underlying spatial reasoning mechanisms are sufficient to implement that strategy.

Path Following. Another approach to the problem is to recognize potential paths in the empty space in the map, instead of generating waypoints around obstacles as above. For domains with clumps of obstacles, this might be more efficient, if a suitable recognition system is present. In addition to continuing our investigation of high-level pathfinding techniques, this approach provides another context to show how the imagery system in SRS can be used in different problems.

A simple path-recognition system was built to investigate this capability. As shown in Figure 6, potential paths are represented as overlapping rectangles. The agent must reason about which rectangles must be traversed to reach the goal, and issue commands to move the agent accordingly. These are the objects present in the figure:
A: The moving unit (real object)
B: Obstacles (real objects)
C1-5: Path rectangles (provided by recognition system)
D: The unit at its destination: an image, defined based on objects not in the figure
E1-4: Waypoints for **A** to move to **D** staying inside **C1-5**

The agent can determine that boxes **C1-5** must be traversed in order before reaching the goal, and generates waypoints **E1-4** to do this. **E1** is described as

*indirect(shapedLike(**A**), inside(**C1**), inside(**C2**), near(**C3**), near(**A**))*

Further waypoints are defined similarly, incrementing the **C** objects and using the prior waypoint image as the final *near* object. This can be considered the inverse of the approach presented above, instead of guiding the unit such that it never intersects any object, the unit must be guided such that it is always inside an object. An agent using a very similar algorithm has been implemented, and can solve all problems where the recognition system provides paths that lead to the goal.

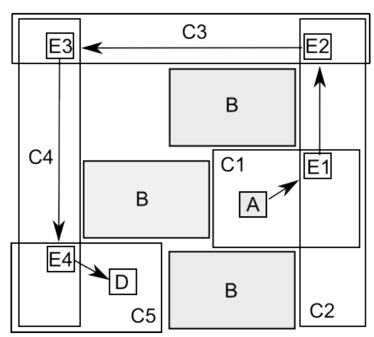

Figure 6. A path following problem.

Extensions

The system implemented in SRS does not represent the entire range of what is possible in a system using this framework. While constraints and preferences like *inside*, *outside*, *near*, and *far* allow for a wide range of possible images, there are many useful images that cannot be expressed using only those predicates.

For example, consider the problem of constructing an arch out of five blocks, as shown in Figure 7. The world is initially given as a table (**F**) with one block on it (**A**). The other blocks to place are present somewhere else in the world (**B, C, D,** and **E**). Each block is placed in turn, by first creating an image (for example, **B'** in the second step), which is then replaced with a real block in the next step.

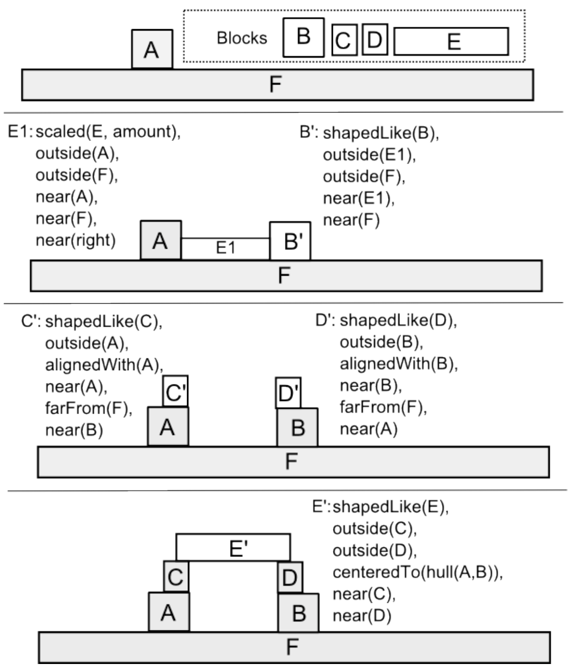

Figure 7. Building an arch in the blocks world.

The predicates used to construct the images are given in the figure. Two new constraints were used, *alignedWith* and *centeredTo*. An image is *alignedWith* a target object if it lies entirely within a region bounded by normals projected from the ends of an edge of that object (Figure 8, left). This constraint is needed to place blocks **C** and **D**. They cannot merely be considered *near* the blocks they are on top of and *farFrom* the table, as the *near* preference does not differentiate between **C** fully on top of **A** and **C** hanging over the edge of **A**, since in both cases the distance between the polygons is 0. An image is *centeredTo* an object if its centroid lies on a normal projected from the center of an edge of that object (Figure 8, right). This is needed to center block **F** over the arch. These constraints, similar to those in SRS, can be easily computed with geometric operations in a possibility space.

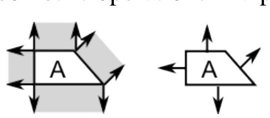

Figure 8. The alignedWith and centeredTo constraints.

Conclusion

Predicate projection must be addressed in any bimodal spatial reasoning system. Unlike predicate extraction, projection entails a translation from the abstract to the specific, requiring its own techniques. We have analyzed the problem, and arrived at a scheme where images are described directly, or built up through constraints and preferences in an indirect description. The predicates used in this framework are not only useful for problem solving, but are also efficiently implemented. The combination of these two factors makes them appropriate for a general bimodal spatial reasoning system.

More broadly, this analysis is a step towards a tighter integration between qualitative and quantitative spatial reasoning techniques. While some problems (such as building the arch in Figure 7) can be solved by simply translating qualitative predicates into quantitative images, many others (like those solved by our ORTS agents) require not only precise quantitative reasoning to correctly place images in the diagram, but complex qualitative reasoning to interpret and modify the diagram. The spatial reasoning ability of Soar is greatly enhanced by careful integration with a quantitative system, and the use of Soar provides powerful qualitative reasoning that would not be possible in a purely quantitative system.

References

Buro, M., Furtak, T. RTS Games as Test-Bed for Real-Time Research, Invited Paper at the Workshop on Game AI, JCIS 2003.

Chandrasekaran, B. Diagrammatic Representation and Reasoning: Some Distinctions. Invited Paper at AAAI Fall 97 Symposium Series, Diagrammatic Reasoning, 1997

Chandrasekaran, B., Kurup, U., Banerjee, B., Josephson, J.R., Winkler, R. An Architecture for Problem Solving with Diagrams. in Diagrammatic Representation and Inference, A. Blackwell, K. Marriott, A. Shimojima, eds., Springer-Verlang, 2004.

Cohn, A.G., Hazarika, S.M., Qualitative Spatial Representation and Reasoning: An Overview. *Fundamenta Informaticae*, 46(1-2):1–29, 2001.

Forbus, K.D., Nielsen, P., Faltings, B. Qualitative Spatial Reasoning: the CLOCK Project. *Artificial Intelligence*, 51(1-3):417–471, 1991.

Kosslyn, S.., Thompson, W., Ganis, G., *The Case for Mental Imagery* . Oxford University Press, 2006.

Lehman, J. F., Laird, J. E., Rosenbloom, P. S., A Gentle Introduction to Soar, an Architecture for Human Cognition, in Invitation to Cognitive Science, Vol. 4, S. Sternberg, D. Scarborough, eds., MIT Press, 1998.