

Linear Inequality Control Rules in State-Space Planning: Beyond the first order predicate calculus

Kevin Cleereman and Michael T. Cox

{kcleerem,mcox}@cs.wright.edu

Department of Computer Science & Engineering

Wright State University

Dayton, OH 45435-0001

Abstract

The general planning problem is known to be PSPACE-complete, and thus a planner must employ control rules to efficiently arrive at a solution to any non-trivial problem. These control rules are generally written in variants of the STRIPS or ADL languages, which are extensions of the first order predicate calculus. However, certain characteristics of a given domain may prevent efficient search due to cyclic dependencies in the bipartite relationships between candidate variable bindings. It is not possible for first order control rules to produce an efficient solution to many problems in the resulting pathological domain. We will outline our system of linear inequality control rules that produce efficient solutions in pathological domains by deviating from FOPC. We will further demonstrate that the use of linear inequality control rules will allow for greater flexibility in efficiently splitting a domain for a multi-agent system.

Keywords: Planning, search control, multi-agent systems, resource constraints, pathological dependency cycles.

1 Introduction

Effective planning comes about when a domain is engineered to provide an abstract model that retains the salient characteristics of the domain while removing unnecessary details, resulting in a compact search space. However, the computational complexity of domain-independent planning is known to be at least PSPACE-complete (Bylander, 1991; Chapman, 1987; Selman, 1994), so for any non-trivial problem search control is needed to further guide the planner through the search space. These control rules are generally written in variants of the STRIPS or ADL languages, which are extensions of the first order predicate calculus (FOPC). However, certain characteristics of a given domain may prevent efficient search due to cyclic dependencies in the bipartite relationships between candidate bindings and required variables in operator representations. This paper presents a new system of linear inequality (LI) control rules that

eliminates the need to employ costly backtracking in binding assignment problems containing a pathological dependency cycle.

An impediment to efficient search control arises when a planning domain possesses several possible binding compositions for its operators. A “*binding composition*” refers to a set of domain objects whose candidate variable bindings are mutually constrained. For instance, consider the operator FIT that associates pegs and holes. Because one peg can fit into at most one hole, and because one hole can contain at most one peg, the process of fitting pegs to holes is simply a bipartite matching problem (Asratian, Denley, & Häggkvist, 1998).

State-space planners employing FOPC control rules are able to match pegs to holes with control rules of the form “if you have a square hole, then fill it with a square peg” or “if you have fit a peg into a hole, then do not try to fit the same peg into another hole.” The task of fitting pegs into holes is much more complicated when, for example, a square peg is allowed to fit into either a square hole *or* a hex hole, while a round peg is allowed to fit into either a round hole *or* a square hole. It now becomes possible for the planner to make inappropriate binding choices by, e.g., fitting all of the square pegs into square holes and making it impossible to fill the hex-shaped holes. However, it is still possible to determine an absolute binding hierarchy in this domain, because all hex holes *must* be filled with square pegs and all round holes *must* be filled with round pegs. Therefore, a state-space planner employing FOPC control rules can avoid making mistakes by filling all hex and round holes before it fills square holes.

Unfortunately, it is no longer possible for FOPC control rules to impose an absolute binding hierarchy in a domain containing cycles in its bipartite graph representation due to particular binding compositions. We call these bipartite graph cycles *pathological dependency cycles*. A domain containing a pathological dependency cycle is called a *pathological domain*. Our non-pathological domain given above is transformed into a pathological domain if we also state that hex pegs may fit into hex holes *or* round holes, because then we can no

longer commit to any bindings without running the risk of having to backtrack over our binding decisions. State-space planners solve this relatively simple task of fitting pegs into holes in exponential time if the planning domain is pathological (Cleereman and Cox, 2004).

In the next section we will introduce our system of LI control rules in a non-pathological domain, and will compare them to FOPC control rules. In section 3 we will demonstrate that FOPC control rules are inadequate in a pathological domain, but that our LI control rules are sufficient for insuring that no backtracking will be required. In section 4 we will demonstrate the use of LI control rules in splitting a domain into three sub-domains for an asynchronous multi-agent system. In the fifth and final section we will offer our conclusions.

2 LI and FOPC Control Rules

The standard logistics (i.e., package delivery) domain (Veloso, 1994) is used by many researchers who study planning. This domain models the transfer of objects and vehicles between various locations. Standard operators include the following.

DRIVE (loc1 isa(LOCATION), loc2 isa(LOCATION)) – transfers a vehicle and its contexts from LOCATION loc1 to LOCATION loc2
 LOAD (obj1 isa(OBJECT), carrier1 isa(CARRIER)) – loads an OBJECT obj1 onto a CARRIER carrier1
 UNLOAD (obj1 isa(OBJECT), carrier1 isa(CARRIER)) – unloads an OBJECT obj1 from a CARRIER carrier1

The preconditions for these operators are fairly intuitive. For example, to UNLOAD an OBJECT from a CARRIER, the OBJECT must first be inside the CARRIER, and to LOAD an object into a CARRIER both the OBJECT and CARRIER must be at the same LOCATION.

To examine unusually hard problems, we modified the domain such that it would perform the basic operations as follows. We specify that once a CARRIER LOADs an OBJECT in a given problem, it may not LOAD another OBJECT for the remainder of that problem¹.

Control rules come into use when avoiding the various idiosyncrasies of planning algorithms. For example, in our domain outlined above the effects list of the LOAD operator would add the predicate IS-USED(CARRIER1) when the planner binds an OBJECT to the CARRIER Carrier1, and we would include as a precondition to the LOAD operator NOT(IS-USED(CARRIER)) to insure

that a previously used CARRIER could not be loaded again. However, a non-linear planner using back-chaining may inadvertently bind the same CARRIER Carrier1 to multiple OBJECT objects with the UNLOAD operator, forcing it to backtrack when it finds that it can LOAD only a single OBJECT (and thus can UNLOAD only a single OBJECT). Rather than re-engineer the entire domain, it is often far simpler to write a control rule that will, in this case, reject the CARRIER objects that have previously been employed in an UNLOAD operator when attempting to bind a CARRIER to the UNLOAD operator.

Control rules are also needed to direct the flow of the planning cycle. For example, in a Blocksworld problem with initial state {ON-TABLE(Block-A), ON-BLOCK(Block-B, Block-A), ON-TABLE(Block-C)} and goal state {ON-BLOCK(Block-A, Block-B), ON-BLOCK(Block-B, Block-C), ON-TABLE(Block-C)}, then the problem can be decomposed into the goals ON-BLOCK(Block-A, Block-B) and ON-BLOCK(Block-B, Block-C). However, uninformed search may first select the goal ON-BLOCK(Block-A, Block-B), resulting in the state {ON-TABLE(Block-C), ON-TABLE(Block-B), ON-BLOCK(Block-A, Block-B)}. This goal will then have to be undone when the goal ON-BLOCK(Block-B, Block-C) is next selected, because Block-B cannot be moved onto Block-C until it is cleared. A control rule can insure that the goal ON-BLOCK(Block-B, Block-C) is selected first, thus improving the planner's efficiency.

The distinction between these two types of rules can be expressed as follows. If a particular problem can be expressed as a maximum matching problem on a bipartite graph, then the first type of control rule is used to insure that the rules of the graph matching problem are not violated. The second type of control rule is used to increase the efficiency of the search to locate a maximum matching. While the first type of control rule is still important to the planning process, we will focus our attention on the second type.

We further modified the logistics domain to increase the difficulty of the matching problem. The LOAD operators were altered so that certain CARRIER objects could only load certain types of objects.

1. LOAD-STD_TRUCK (obj1 isa(PRODUCE), carrier1 isa(STD_TRUCK))
2. LOAD-REFRIG_TRUCK (obj1 isa(OR(PRODUCE, DAIRY)), carrier1 isa(REFRIG_TRUCK))
3. LOAD-FREEZER_TRUCK (obj1 isa(OR(DAIRY, MEAT)), carrier1 isa(FREEZER_TRUCK))

¹ The only exception to this rule is if the planner backtracks over a previous binding commitment. In either case, a CARRIER will LOAD at most one OBJECT in the final plan.

The relations between these LOAD operators can be expressed as a bipartite graph. Arrows represent the valid binding pairs.

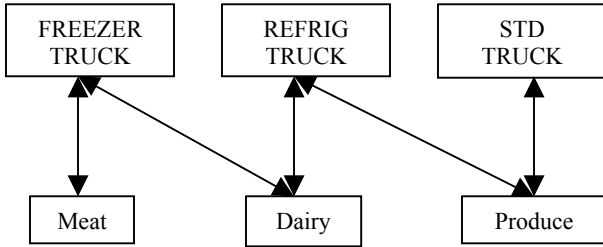


Figure 1. Bipartite Graph Representation.

Note that this graph is a simplified domain representation. Any particular problem has a given number of FREEZER_TRUCK, REFRIG_TRUCK, STD_TRUCK, MEAT, DAIRY, and PRODUCE nodes, but these nodes have edges as given by the domain graph (e.g., all MEAT nodes connect to all FREEZER_TRUCK nodes).

For any given problem, the number of FREEZER_TRUCK, REFRIG_TRUCK, STD_TRUCK, MEAT, DAIRY, and PRODUCE nodes is given by the variables *freezer*, *refrig*, *std*, *meat*, *dairy*, and *prod*, respectively². Thus, a problem possesses a solution iff

1. $std + refrig \geq produce$
2. $refrig + freezer \geq dairy + meat$
3. $freezer \geq meat$
4. $std + refrig + freezer \geq produce + dairy + meat$

This is supported by Hall's Theorem (Nering, 1993). If we operate on the assumption that all of these inequalities are satisfied at the start of a problem, then we can write control rules that will insure that these inequalities are not violated during the course of planning.

Linear inequality control rules as shown in Table 1 consist simply of determining if one of these inequalities will be violated by a binding decision before committing to that binding decision.

The meta-predicate "cand-match" (short for candidate-match) returns true if there is a CARRIER object of the given type that is available to LOAD a PACKAGE of the given type, else it returns false. If the meta-predicate and all inequalities return true, then the CARRIER and PACKAGE are bound to the LOAD operator, else the binding pair is rejected.

² A CARRIER or OBJECT will only have a node if it is available for binding in a LOAD operator. For example, a CARRIER Carrier1 that is IS-USED(Carrier1) in the initial state will not have a representative node, and an OBJECT that is already at its goal location will not have a representative node.

When using LI control rules for this domain, the planner need not select its goals or operators in any particular order. The matter changes when using FOPC control rules, because FOPC cannot explicitly confirm that the consistency of the inequalities is being maintained. FOPC is still able to insure that no backtracking occurs in a non-pathological domain, but it must explicitly order its goal and operator selection to do so.

Table 1. LI Control Rule Set

1	if cand-match(STD_TRUCK – PRODUCE) & ($std - 1 + refrig \geq produce - 1$) then select binding; <i>std--</i> ; <i>produce--</i> ; else reject binding
2	if cand-match(REFRIG_TRUCK – PRODUCE) & ($refrig - 1 + freezer \geq dairy + meat$) then select binding; <i>refrig--</i> ; <i>produce--</i> ; else reject binding
3	if cand-match(REFRIG_TRUCK – DAIRY) & ($std + refrig - 1 \geq produce$) then select binding; <i>refrig--</i> ; <i>dairy--</i> ; else reject binding
4	if cand-match(FREEZER_TRUCK – DAIRY) & ($freezer - 1 \geq meat$) then select binding; <i>freezer--</i> ; <i>dairy--</i> ; else reject binding
5	if cand-match(FREEZER_TRUCK – MEAT) & ($refrig + freezer - 1 \geq dairy + meat - 1$) then select binding; <i>freezer--</i> ; <i>meat--</i> ; else reject binding

In contrast, FOPC control rules as shown in Table 2 guide search by taking the given action(s) if a particular sentence in FOPC is true. FOPC cannot efficiently represent a linear inequality, and thus LI control rules have greater representational power.

The meta-predicate "candidate-goal" returns true if the given goal is queued, the meta-predicate "current-goal" returns true if the given goal has been selected, and the meta-predicate "candidate-op" returns true if the given operator is a candidate for achieving the current goal state.

By insuring that all Is-Loaded(Meat) goals are solved before all Is-Loaded(Dairy) and Is-Loaded(Produce) goals, FOPC is able to insure that the inequality $freezer \geq meat$ is not violated by an inauspicious FREEZER_TRUCK – DAIRY match. Similarly, by insuring that all Is-Loaded(Dairy) goals are solved before all Is-Loaded(Produce) goals, FOPC is able to insure that the inequality $refrig + freezer \geq dairy$ is not violated by an inauspicious REFRIG_TRUCK – PRODUCE match³.

³ Note that an alternative set of FOPC control rules could have been used instead. For example, we might have decided to solve all Is-Loaded(PRODUCE) goals first, followed by all Is-

Table 2. FOPC Control Rule Set

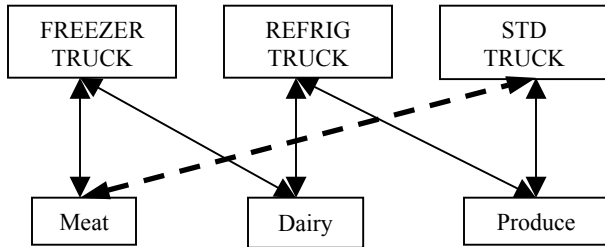
1	if candidate-goal(Is-Loaded(Meat)) & candidate-goal(Is-Loaded(Dairy)) then prefer goal Is-Loaded(Meat)
2	if candidate-goal(Is-Loaded(Meat)) & candidate-goal(Is-Loaded(Produce)) then prefer goal Is-Loaded(Meat)
3	if candidate-goal(Is-Loaded(Dairy)) & candidate-goal(Is-Loaded(Produce)) then prefer goal Is-Loaded(Dairy)
4	if current-goal(Is-Loaded(Meat)) & candidate-op(Load-Freezer_Truck) then prefer op Load-Freezer_Truck
5	if current-goal(Is-Loaded(Dairy)) & candidate-op(Load-Freezer_Truck) then prefer op Load-Freezer_Truck
6	if current-goal(Is-Loaded(Produce)) & candidate-op(Load-Refrigerator_Truck) then prefer op Load-Refrigerator_Truck

3 Pathological Domains

While LI and FOPC control rules have similar expressive powers on some domains, there are domains containing pathological dependency cycles for which no set of FOPC control rules can insure that backtracking will not occur. If one or more pathological dependency cycles occur within a domain, then that is a pathological domain. For example, if in our previously modified logistics domain we had specified that STD_TRUCK objects could move either PRODUCE or MEAT:

LOAD-STD_TRUCK (obj1 isa(OR(PRODUCE, MEAT)), carrier1 isa(STD_TRUCK))

Then the domain's graphical representation will change as follows. Note the new STD_TRUCK – MEAT edge.

**Figure 2. Pathological Domain.**

Loaded(Dairy) goals. We could not, however, have solved all Is-Loaded(Dairy) goals first, for as we will explain in Section 4 the $refrig + freezer \geq dairy$ equation creates a pathological sub-domain.

We call this relationship a pathological dependency cycle, because the addition of a cycle to the graph structure effectively disables FOPC control rules. A problem in the pathological domain will possess a solution iff

1. $std + refrig \geq produce$
2. $refrig + freezer \geq dairy$
3. $freezer + std \geq meat$
4. $std + refrig + freezer \geq produce + dairy + meat$

Note the change in inequality 3 from the non-pathological domain. Our previous set of FOPC control rules will no longer work in maintaining the consistency of these inequalities, for *any* possible match has the chance of causing the left-hand side of an inequality to possess a lesser value than the right-hand side. For example, given a simple problem with $freezer = 1$, $std = 1$, $meat = 1$, and $dairy = 1$, it is clear that a FREEZER_TRUCK – MEAT binding will violate equation 2, yet our FOPC control rules have no means of determining that this is the case and will thus have to backtrack to reverse this binding decision. If we alter our control rules so that they favor STD_TRUCK – MEAT bindings over FREEZER_TRUCK – MEAT bindings, then the simple problem with $freezer = 1$, $std = 1$, $meat = 1$, and $produce = 1$ will require backtracking to free up the STD_TRUCK object to load the PRODUCE object. In contrast, our LI control rules require little modification, and are still capable of insuring that no backtracking need occur.

Table 3. Pathological LI Control Rule Set

1	if cand-match(STD_TRUCK – PRODUCE) & $(freezer + std - 1 \geq meat)$ then select binding; $std--$; $produce--$; else reject binding
2	if cand-match(STD_TRUCK – MEAT) & $(std - 1 + refrig \geq produce)$ then select binding; $std--$; $meat--$; else reject binding
3	if cand-match(REFRIG_TRUCK – PRODUCE) & $(refrig - 1 + freezer \geq dairy)$ then select binding; $refrig--$; $produce--$; else reject binding
4	if cand-match(REFRIG_TRUCK – DAIRY) & $(std + refrig - 1 \geq produce)$ then select binding; $refrig--$; $dairy--$; else reject binding
5	if cand-match(FREEZER_TRUCK – DAIRY) & $(std + freezer - 1 \geq meat)$ then select binding; $freezer--$; $dairy--$; else reject binding
6	if cand-match(FREEZER_TRUCK – MEAT) & $(refrig + freezer - 1 \geq dairy)$ then select binding; $freezer--$; $meat--$; else reject binding

Note that when insufficient CARRIER resources exist, it will be necessary for goal transformations (Cox, 2003; Cox and Veloso, 1998) to eliminate nodes from the right-hand (OBJECT) side of one or more inequalities, else the LI control rules will prevent a partial solution from being achieved. For example, given a simple problem with $freezer = 1$, $meat = 1$, and $dairy = 1$, the single FREEZER_TRUCK object will be prevented from being bound to the MEAT object because this would make the goal of loading the DAIRY object unattainable, but the FREEZER_TRUCK object will also be prevented from being bound to the DAIRY object because this would make the goal of loading the MEAT object unattainable. When a goal transformation erodes or eliminates one of the $Is-Loaded(Object)$ goals then the LI control rules will achieve their desired effect.

4 Multi-Agent Systems

The advantages of using LI control rules in lieu of or in addition to FOPC control rules are most apparent when a multi-agent system splits a domain into sub-domains to delineate agent responsibility for planning problems (Cox, Elahi, & Cleereman, 2003; Elahi, 2003). For example, we may wish to split the non-pathological logistics domain containing 3 Load operators (given in section 2) into a 3-agent system so that each agent will possess a single Load operator. Specifically, Agent-1 possesses the Load-Std_Truck operator, Agent-2 possesses the Load-Refrig_Truck operator, and Agent-3 possesses the Load-Freezer_Truck operator. We will assume that all three agents have access to all OBJECT objects in a given problem, meaning that they can bind their CARRIER objects to any OBJECT they wish.

We can treat this domain as we did the single-agent domain by stipulating that only one agent may act on the problem space during any given clock cycle, effectively solving problems sequentially. For example, if we stipulate that Agent-2 cannot act until Agent-3 has either bound all of its FREEZER_TRUCK objects or has bound all of the MEAT and DAIRY objects then we are in effect applying control rules {1, 2, 4, 5} given in Table 2. If we further stipulate that Agent-1 cannot act until Agent-2 has bound all of its REFRIG_TRUCK objects then our multi-agent system will be essentially equivalent to the single-agent system in Section 2. However, it is often desirable for agents in a multi-agent system to act in parallel and/or asynchronously to maximize processor throughput and minimize the time needed to generate a plan. Because one or more sub-domains of a non-pathological domain may be pathological (as it is in our example), LI control rules will often be needed to maximize planning efficiency.

Agent-1:

$$1. \quad std + refrig \geq produce$$

Agent-2:

$$1. \quad std + refrig \geq produce$$

$$2. \quad refrig + freezer \geq dairy$$

Agent-3:

$$3. \quad refrig + freezer \geq dairy$$

$$4. \quad freezer \geq meat$$

Agent-1 and Agent-3 both possess non-pathological sub-domains, because any reduction in std will by necessity produce a reduction in $produce$, and likewise for Agent-3 any reduction in $freezer$ will produce a reduction in $meat$ when the Table 2 control rules are being applied. However, Agent-2 possesses a pathological sub-domain: any reduction in $refrig$ that is not accompanied by a reduction in $produce$ (i.e., any REFRIG_TRUCK – DAIRY match) may violate equation 1, and any reduction in $refrig$ that is not accompanied by a reduction in $dairy$ (i.e., any REFRIG_TRUCK – PRODUCE match) may violate equation 2. It is thus necessary for Agent-2 to explicitly confirm one of these equations via LI control rules before it commits to a match.

The problem with a multi-agent pathological domain is essentially equivalent to the problem with a single-agent pathological domain. If we divide up the Load operators of the pathological logistics domain given in section 3 as we divided up the Load operators of the non-pathological logistics domain earlier in this section, then we are left with the following system of inequalities to form the basis of multi-agent negotiation and coordination.

Agent-1:

$$1. \quad std + refrig \geq produce$$

$$2. \quad freezer + std \geq meat$$

Agent-2:

$$1. \quad std + refrig \geq produce$$

$$2. \quad refrig + freezer \geq dairy$$

Agent-3:

$$1. \quad refrig + freezer \geq dairy$$

$$2. \quad freezer + std \geq meat$$

This pathological domain has split into three pathological sub-domains, leaving all agents unable to commit to any bindings when using FOPC control rules. However, LI control rules behave no differently for the multi-agent domain than for the single-agent domain, allowing even pathological domains to benefit from the advantages of being represented as a multi-agent system.

5 Conclusions

The problem of efficiently allocating resources is ubiquitous in planning, but is often inefficiently

represented by STRIPS and ADL like languages. Linear Inequality control rules are capable of producing vast increases in planning efficiency, yet they require that only a domain's control rule representation be modified, *not* the domain itself.

More importantly, LI control rules define a means by which agents in a multi-agent system can communicate with each other and efficiently allocate planning resources between one another. Agents are not bound to a sequential order of operations as they may be under the direction of FOPC control rules, but are instead given the freedom to act in parallel and/or asynchronously with other agents.

We are currently in the process of investigating the use of LI control rules in automatic goal transformations, as well as the use of LI control rules in domains represented as a graph of 3 or more colors. We have already determined that LI control rules allow some 3-colored domains to produce efficient solutions, but we are still unsure of whether this is the exception or the rule. The general 3-colored matching problem is NP-Complete, so we would expect that either some domains cannot be solved with LI control rules, or that the task of formally generating LI control rules for a general 3-colored domain is NP-Complete, or both.

Acknowledgements

This research is funded by a contract from the Air Force Research Laboratory and by the Ohio Board of Regents. We especially thank Dr. Mateen Rizki of the Wright State University CECS Department for recognizing the parallel between our binding selection problem and the bipartite matching problem. We also thank Mr. Langdon and Mr. Somanchi of the Wright State University Collaboration and Cognition laboratory for their valuable input.

References

Asratian, A. S., Denley, T., and Häggkvist, R. (1998). *Bipartite Graphs and Their Applications*. Cambridge University Press: Cambridge.

Bylander, T. (1991). Complexity results for planning. In *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 274-279).

Chapman, D. (1987). *Planning for conjunctive goals*. *Artificial Intelligence* 32:333-377.

Cleereman, K., and Cox, M. T. (2004). Pathological Dependency Cycles in State-Space Planning: When Control Rules Fail. In *Proceedings of the Florida Artificial Intelligence Research Society (FLAIRS-04) Conference*. Menlo Park, CA: AAAI Press.

Cox, M. T. (2003). Planning as mixed-initiative goal manipulation. In *Proceedings of the Workshop on Mixed-Initiative Intelligent Systems at the 18th International Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Cox, M. T., Elahi, M., and Cleereman, K. (2003). A distributed planning approach using multiagent goal transformations. In Ralescu (Ed.), *Proceedings of the 14th Midwest Artificial Intelligence and Cognitive Science Conference* (pp 18-23). Cincinnati: Ompress.

Cox, M. T., and Veloso, M. M. (1998). Goal transformations in continuous planning. In M. desJardins (Ed.), *Proceedings of the 1998 AAAI Fall Symposium on Distributed Continual Planning* (pp. 23-30). Menlo Park, CA: AAAI Press / The MIT Press.

Elahi, M. M. (2003). *A distributed planning approach using multiagent goal transformations*. Masters Thesis, Department of Computer Science and Engineering, Wright State University.

Nering, E. D. (1993). *Linear Programs and Related Problems* (pp 293-298). Boston: Academic Press.

Selman, B. (1994). Near-Optimal Plans, Tractability, and Reactivity. In J. Doyle, E. Sandewall, & P. Torasso, (Eds.), *Proc. 4th Conference in Knowledge Representation* (pp. 521—529), San Francisco: Morgan Kaufmann.

Veloso, M. M. (1994). *Planning and Learning by Analogical Reasoning*. Berlin: Springer.