Pathological Dependency Cycles in State-Space Planning: When control rules fail

Kevin Cleereman and Michael T. Cox
{kcleerem,mcox}@cs.wright.edu
Department of Computer Science & Engineering
Wright State University
Dayton, OH 45435-0001

Abstract

Pathological dependency cycles occur in state-space planners when control structures cannot efficiently determine a maximal matching for a bipartite operator/binding graph. Without proper search control, the planner will require many computationally expensive backtracks to arrive at a solution. We present a method for improving planning efficiency in the midst of pathological dependency cycles by employing informed resource re-allocation in lieu of uninformed backtracking. Empirical studies demonstrate significant improvement in search effort when search control is employed in backtracking. Existing theoretical results suggest that some form of informed resource re-allocation can be used to produce an approximately $O(n^{2.5})$ solution for many pathological domain classes, as opposed to the $O(k^n)$ solution produced in uninformed backtracking.

Introduction

Search techniques are ubiquitous in computer science and in artificial intelligence in particular. The typical uninformed tree search algorithms such as depth-first search and breadth-first search are often inadequate for problems of even moderate complexity, and so it becomes necessary to employ search controls in finding better solutions more quickly. For example, it is known that domain-independent planning is at least PSPACE-complete (Bylander, 1991; Chapman, 1987; Selman, 1994), so effective search must be guided by some form of search control. We will show in this paper that state-space planning search controls that possess the expressive power of the first order predicate calculus (FOPC) cannot effectively guide search in certain *pathological domains* possessing *pathological dependency cycles*, but may often be supplemented with search controls that guide backtracking to retain acceptable search efficiency.

State-space planners seek to produce sequences of actions that transform some initial state of the environment into a particular desired state. Such planners find a solution by searching through a set of action representations in the form of planning operators. Operators represent actions by associating the effects of the operator with the preconditions that must be true for the operator to be applied. For example, a FIT operator may achieve the effect of having a peg fitting inside a hole, contingent upon the operator's preconditions being satisfied. Both effects and preconditions are simply world-state predicates or relations between objects in the domain. For example, if peg1 has been FIT into hole1, then the world state would contain a predicate such as inside(peg1, hole1).

A domain is said to contain binding/operator dependencies if the use of certain operators or bindings prevents the use of certain other operators or bindings. For example, our pegs-into-holes domain possesses dependencies insofar as a hole can contain at most one peg, and a peg can be driven into at most one hole. Thus, if we FIT peg1 into hole1, then we may not also FIT peg1 into hole2. Note that such a dependency may be represented as a matching problem on a bipartite graph (Asratian, Denley, & Häggkvist, 1998), with pegs representing one node coloring and holes representing another. We will employ this isomorphism in estimating the computational complexity of our algorithm as outlined in section three.

An additional dependency arises if we expand our simple pegs-into-holes domain such that it contains multiple types of pegs (e.g., square, round, and hex pegs), as well as multiple types of holes (e.g., square, round, and hex holes). If we further allow that certain pegs may be FIT into multiple types of holes, then it becomes necessary that search control be employed in insuring that we successfully FIT as many pegs into as many holes as possible. If we accidentally fill all square holes with round pegs, only to find that our square pegs will not fit into round holes, then we must then employ computationally expensive backtracking to transfer our round pegs to round holes. Alternatively, we could have simply employed search control to insure that we use all of our square pegs to fill the square holes before we start to use round pegs to fill the square holes.

The problem that we confront in this paper relates to sets of domain rules whose graphical representation contains cycles[1]. Such pathological dependencies prevent efficient solutions during search for a plan. In the next section, we examine the problem of using search control in a domain containing a pathological dependency cycle. Section three proposes a solution to the problem examined in section two, and section four presents the empirical results of our solution's application. Finally, we conclude this paper with discussion and a roadmap for future research.

Pathological Domains

The standard logistics (or 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. Its basic operations are as follows.

DRIVE (loc1, loc2) – transfers a vehicle and its contexts from loc1 to loc2
LOAD (obj1) – loads an object of type obj1 onto a vehicle
UNLOAD (obj1) – unloads an object of type obj1

The preconditions for these operators are fairly intuitive. For example, to unload an object from a vehicle, the object must first be inside the vehicle (i.e., it must have been loaded), and to load an object into a vehicle both the object and vehicle must be at the same location.

We modified the basic logistics domain to create what we call the Pathological Logistics Domain. The purpose of this domain is to demonstrate the ineffectiveness of search control under certain conditions and to examine alternate control mechanisms. In this new domain, we specified that every truck could load only one object per problem, and every operator-binding match carried an equal utility (every graph edge had an equal weight).

For testing purposes, we implemented in Allegro Common Lisp 6.2 using CLOS (Common Lisp Object System) a multiple inheritance version (dubbed Sprodigy) of a state-space non-linear planning and learning architecture called PRODIGY (Carbonell, et al, 1992; Veloso, et al, 1995). The Object System's built-in multiple inheritance hierarchy allowed us to write a compact pathological domain containing only three operators.

LOAD-FREEZER_TRK (OR(Meat, Dairy))
LOAD-REFRIG_TRK (OR(Dairy, Produce))
LOAD-STD_TRK (OR(Produce, Meat))

[1] For example, a domain wherein square pegs can be driven into square or round holes, round pegs can be driven into round or hex holes, and hex pegs can be driven into hex or square holes.

This domain contains three different TRUCK types, each capable of loading a single object from a list of two PACKAGE types. In turn, each of the three different PACKAGE types is capable of being loaded onto a single truck from a list of two TRUCK types. In Figure 1, we represent this domain as a bipartite graph. Because there is a 1-to-1 correspondence (that can be generalized to an N-to-M correspondence) between Trucks and Objects, a planner will require some means of performing maximum matching between Trucks and Objects to effectively implement this domain.[2]

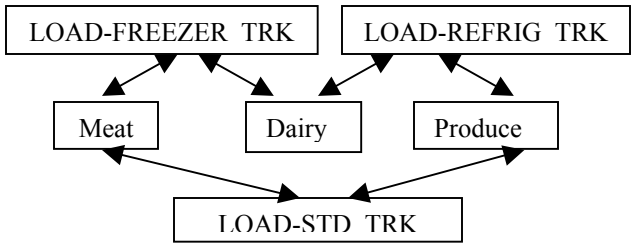

Figure 1. Pathological dependencies.

In general, a domain will be pathological if it contains a set of cyclic operators:

$1 \le i \le N, N \ge 2,$

$Op_i <v_i> (OR(t_i\ t_{(i \bmod N)+1}))$

Problem solving in PRODIGY is performed by *means-end analysis*, in which a goal is picked that is not yet satisfied and an attempt is made to find an operator to satisfy it. The open preconditions of this operator become sub-goals, and the process is repeated until all goals are attached to operators or are satisfied. The PRODIGY decision cycle has four decision points that can be heuristically manipulated by the application of control rules.

1. Given a list of pending goals, the system decides which goal to pursue next.
2. Given a goal, it decides with which operator it will try to achieve the goal.
3. Given an operator, it decides with what variable-bindings it will instantiate the operator.
4. Given instantiated operators with no open preconditions and a list of pending goals, it decides whether to apply an operator and hence change the current state, or whether to choose a pending goal to solve.

[2] We did not examine the more complex problem of finding a stable matching on a bipartite graph with weighted edges, which is analogous to a domain implementation containing inferential goal transformations (Cox and Veloso, 1998). However, the principle of the stable matching problem is identical to that of the maximum matching problem.

When PRODIGY applies an instantiated operator in decision number four above, an action's effects are projected from the current state. Although backtracking may reverse this decision, a final plan will consist of a sequence of these operator applications.

As shown in Table 1, we attempt to improve planning efficiency with control rules that order the goal/operator selection process (decision points 1 and 2 above). No well-defined entry point for the graph exists, so we arbitrarily choose our starting point at the Is-Loaded(Meat) goal.

Table 1. Pathological 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_TRK) **then** select op LOAD-FREEZER_TRK
5	**if** current-goal(Is-Loaded(Dairy)) & candidate-op(LOAD-FREEZER_TRK) **then** select op LOAD-FREEZER_TRK
6	**if** current-goal(Is-Loaded(Produce)) & candidate-op(LOAD-REFRIG_TRK) **then** select op LOAD-REFRIG_TRK

Unfortunately, this is a Pathological Domain. It contains one pathological dependency cycle (though in general, a Pathological Domain can contain any number of dependency cycles). When a pathological dependency cycle exists in a domain, then for any arbitrary set of FOPC control structures we can nevertheless generate a problem that will require backtracking.

For example, even when we employ our logistics control rules in the Pathological Logistics Domain, backtracking is needed to solve the problem with 2 Freezer_Trks, 1 Refrig_Trk, 3 Std_Trks, 2 Meat objects, 2 Dairy objects, and 2 Produce objects. In the initial state, all Truck and Food objects are at the same location. The goal is to Load all of the Food objects.

First, all Is-Loaded(Meat) goals are achieved (control-rules {1,2}) using up all Freezer_Trk resources (control-rule {4}). The planner then attempts to achieve all goals Is-Loaded(Dairy) (control-rule {3}), but there are insufficient resources for achieving this. Thus, the planner must backtrack to use a Std_Trk vehicle in achieving a goal Is-Loaded(Meat) so that a Freezer_Trk vehicle can be re-allocated to achieving a goal Is-Loaded(Dairy).

It is important to realize that no given set of control rules will be able to efficiently solve all problems in this domain. If the control rules are re-ordered, then the numbers of instances in the problem above can be similarly re-ordered to defeat the new set of control rules.

An Algorithm for Resource Reallocation

Our solution to this problem is to employ directed resource reallocation to improve backtracking efficiency. Presently, this essentially amounts to the use of control rules during backtracking. Given the problem example above, planning proceeds as usual. However, when the planner finds that insufficient carriers exist to solve all Is-Loaded(Dairy) goals, it does not immediately close the search node. Instead, it searches the problem-space for alternative bindings that it can employ in achieving the remaining goals Is-Loaded(Dairy) and then attempts to re-allocate those resources.

In this particular example, the planner attempts to free an instance of {Freezer_Trk, Refrig_Trk} to achieve Is-Loaded(Dairy). There are no instances of Refrig_Trk that are not already being employed in achieving goals of the type Is-Loaded(Dairy), but there are instances of Freezer_Trk that are not being employed in achieving goals of the type Is-Loaded(Dairy). The planner attempts to find operator/binding alternatives to an instance of Freezer_Trk, and finds that there is a free instance of Std_Trk that it can use in lieu of an instance of Freezer_Trk in achieving one of the instances of Is-Loaded(Meat). It de-allocates an instance of Freezer_Trk, uses the alternative binding Std_Trk to achieve the relevant instance of Is-Loaded(Meat), and re-allocates the freed instance of Freezer_Trk to achieve the unsolved goal Is-Loaded(Dairy). Ordinarily a planner will try all possible combinations of Freezer_Trk and Refrig_Trk in achieving goals Is-Loaded(Meat) and Is-Loaded(Dairy) before it exhausts its alternatives and employs Std_Trk in achieving goal Is-Loaded(Meat). Directed resource re-allocation bypasses this exhaustive and expensive search process using two basic functions called expand and de-allocate.

Expand

The expand function proceeds as follows.

(EXPAND *node*
if unallocated *binding* exists **then**
select *binding*
else if allocated *binding* exists and
(DE-ALLOCATE *binding*) **then**
select *binding*
else
no plan)

If an unallocated binding exists, then the search proceeds as normal. If no unallocated bindings exist but an allocated binding may be de-allocated, then the allocated

binding is de-allocated and used as if it were unallocated. A binding is "allocated" if it is already maximally bound. In the Pathological Logistics Domain, a truck can be used in a single LOAD binding before it is allocated, but in the general case an object can be used in N bindings before it is allocated.

The only allocated bindings considered are those of a different "kind," e.g., if a Carrier binding is needed for an Is-Loaded(Meat) goal then the algorithm will not DE-ALLOCATE bindings already bound by an Is-Loaded(Meat) goal. This amounts to the use of control rules during backtracking, though we are researching the possibilities of employing a domain-independent definition of "kind."

De-Allocate

The de-allocate function proceeds as follows.

(DE-ALLOCATE *binding*
  if *binding* is currently being DE-ALLOCATED **then**
    return nil
  else if unallocated *alternate-binding* exists **then**
    de-select *binding*
    select *alternate-binding*
    return t
  else if allocated *alternate-binding* exists and
    (DE-ALLOCATE *alternate-binding*) **then**
    de-select *binding*
    select *alternate-binding*
    return t
  else
    return nil)

If a binding is already being DE-ALLOCATED by a function call on the stack, then it will not be DE-ALLOCATED a second time. This avoids infinite de-allocation loops.

If a binding is de-selected, then the appropriate binding-node is closed, and the alternate-binding is then selected by the parent node. This is sufficient for a simple domain like the Pathological Logistics Domain, but we are examining the possibilities of this causing unintended side-effects in a more complex domain.

This algorithm is similar to the double-looping structure used in Hopcroft and Karp's bipartite matching algorithm (Hopcroft and Karp, 1973), which runs in $O(n^{2.5})$. (Further research is needed to determine if our solution fully matches this performance, but given Hopcroft and Karp's results we expect that a polynomial time algorithm exists that will efficiently solve *all* binding problems that can be represented as a bipartite matching problem, regardless of whether the domain is non-pathological or pathological.) Hopcroft and Karp's algorithm searches for an augmenting path for a given non-maximal matching, whereas PRODIGY's default search strategy may destroy large sections of the current matching path during the backtracking process. In terms of search, goal-directed backtracking will only close a node *after* a meta-search determines that the node can never be expanded, thus insuring that minimal destruction is performed on the search tree. In contrast, PRODIGY may close large sections of the search tree during backtracking, which often leads to the inadvertent destruction of many useful search nodes that will merely have to be re-expanded later.

Though a polynomial-time solution to searching the Pathological Logistics Domain exists, it is known that the 3-Color Bipartite Matching problem (being reducible to the problem of partitioning a graph into triangles) is NP-Complete (Garey and Johnson, 1979). This means that a modified Logistics domain that must match Drivers to Trucks, Trucks to Packages, *and* Drivers to Packages[3] may not have an efficient solution through directed backtracking. We examine the ramifications of this problem in our conclusion, but note that the problem is not fundamentally altered if the choice of Driver does not interfere with the choice of Package and vice versa, since said domain's graph representation would still possess only two colors.

Empirical Results

We ran tests on three different i-Pack[4] domains: the 3-Pack domain, the 4-Pack domain, and the 5-Pack domain (see Figures 2, 3, and 4 respectively). The 3-Pack domain is equivalent to the domain described in the Pathological Domain section, the 4-Pack domain contains an additional Load operator (for a total of 4), an additional vehicle type (for a total of 4), and an additional object type (for a total of 4), while the 5-Pack domain contained 5 Load operators, 5 vehicle types, and 5 object types. Each domain is represented by a 2-regular bipartite graph, with vehicle types and object types respectively composing the two colors.

The domains did not contain any control rules, but instead used Prodigy 4.0's Random-Behavior flag to vary the planning decisions. That is, PRODIGY normally chooses equal decision alternates (i.e., those without control rule guidance) at each of its four decision points in a left to right order within the search tree. The flag imposes an arbitrary order instead.

The test results are on the x1, x2, x3, and x4 problems, where x1 denotes that the problem contained one object of each Food type and one vehicle of each Truck type, x2 denotes that the problem contained two objects of each Food type and two vehicles of each Truck type, and so forth. The initial state for each problem placed all Truck

[3] Perhaps some Drivers have an aversion to transporting Produce, for example.

[4] With apologies to Hewlett Packard.

and Food objects at the Start location, and the goal state for each problem was to place all Food objects at the Destination location. Thus, to solve the problem, each vehicle had to Load a single object, then the vehicle had to Drive to the Destination where the object would be Unloaded.

We halted a search after 20,000 nodes were opened. This limit is represented by the "cutoff" line near the top of figures 2, 3, and 4. The default data begins to converge to the cutoff point as the domains and problems increase in complexity because complex problems are less likely to be solved within the 20,000 node limit. For example in Figure 4, 90% of the x3 problems in the 5-Pack domain were terminated at the cutoff point during Default search, whereas 100% of the x4 problems in the 5-Pack domain were terminated at the cutoff point during Default search. Had we allowed these tests to run to completion, then we would have expected to see an exponential increase in the average number of nodes expanded, but it is not practical to run multiple tests in which hundreds of thousands (or even millions) of search nodes are expanded. None of the directed tests ever reached the cutoff point.

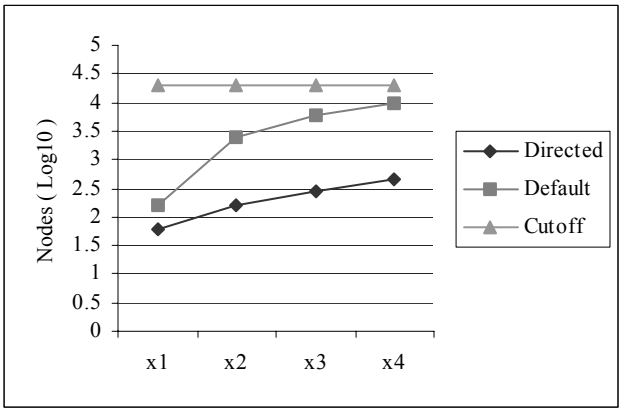

Figure 2. Search effort as a function of the number of object instances in the 3-Pack domain

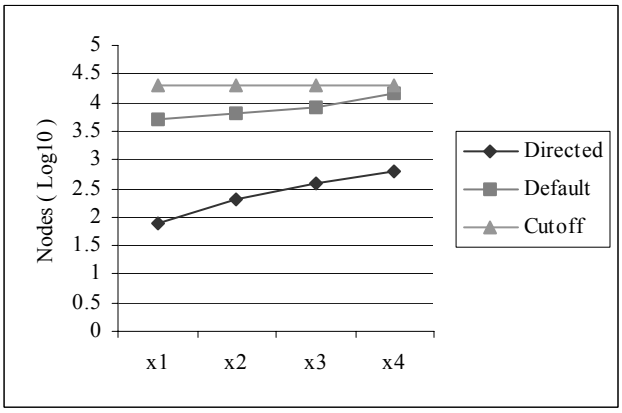

Figure 3. Search effort as a function of the number of object instances in the 4-Pack domain

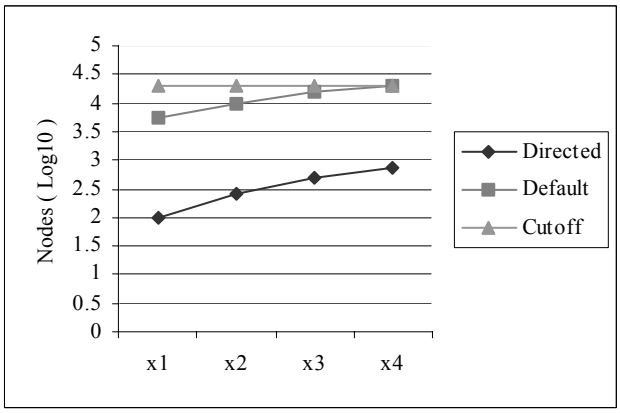

Figure 4. Search effort as a function of the number of object instances in the 5-Pack domain

Conclusions

Resource re-allocation presents itself as a viable solution to the problem of pathological looping within a planning domain. The catastrophic results produced without the aid of directed resource re-allocation justify the inclusion of this technique in solving pathological domains.

We seek to implement an inter-agent communication protocol for the *Prodigy/Agent* system (Cox *et. al*, 2003; Cox *et. al*, 2001; Elahi, 2003) to provide a means by which agents can intelligently distribute resources between one another. This is the first step in designing a perpetual agent that will be able to collaborate/compete with other agents for the purpose of recognizing and handling surprise (Cox, 1997). As we noted in the Resource Reallocation section, directed backtracking performs minimal destruction of the search tree. This should greatly improve the blame assignment process when an agent inevitably experiences reasoning failure. Ideally, we would prefer multiple agents with incomplete search trees (i.e., exhibiting partial success) to coalesce these into a more complete search tree (i.e., exhibiting greater, even total, success).

During our preliminary research, we found that single-agent planners could also benefit from a similar communication protocol, particularly in domains containing pathological dependency cycles. We have therefore begun to implement a single-agent communication protocol that we can easily extend into a multi-agent communication protocol. For example, multiple agents may exist that specialize in handling of packages of particular types. The agents would therefore need to coordinate with each other given individual and shared resources (i.e., motor-pools).

Although control structures with the expressive power of FOPC cannot be used in directing search in a pathological domain, we have determined that there exists a class of control structures beyond the power of FOPC

that may be employed to this end. We are therefore in the process of determining the various tradeoffs between employing classic FOPC control structures, our meta-FOPC control structures, and directed backtracking.

We are also in the process of investigating the computational complexity of the directed backtracking algorithm, as well as the possible benefits of integrating GA's (Genetic Algorithms) into the directed backtracking system. The Pathological Logistics Domain may be simple enough to have a polynomial-time solution, but, as noted in the introduction, domain-independent search is in general at least PSPACE-complete. We thus hope that GA's will provide a means of searching in complex problem spaces, as well as provide a means of integrating the results of multiple perpetual search agents into one search tree (solution). However, we realize that integrating GA's into PRODIGY will be a far more daunting task than that of integrating directed resource re-allocation. Nonetheless taken as a whole, the research contained in this paper provides a number of avenues for further research that promise to make significant contributions to planning and to a better understanding of the problems therein.

Acknowledgements

This research is funded by a scholarship from the Dayton Area Graduate Study Institute 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 the Wright State University Writing Center for their invaluable assistance in editing this paper, as well as Mr. Brenton Bostick at Wittenberg University for providing his perspective to this paper. Finally, we thank the anonymous reviewers of this paper for their valuable input.

References

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

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

Carbonell, J. G.; Blythe, J.; Etzioni, O.; Gil, Y.; Joseph, R.; Kahn, D.; Knoblock, C.; Minton, S.; Perez, A.; Reilly, S.; Veloso, M. M.; and Wang, X. (1992). *PRODIGY4.0: The Manual and Tutorial*, Technical Report, Computer Science Department, Carnegie Mellon University.

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

Cox, M. T., Edwin, G., Balasubramanian, K., & Elahi, M. (2001). Multiagent goal transformation and mixed-initiative planning using Prodigy/Agent. In N. Callaos, B. Sanchez, L. H. Encinas, & J. G. Busse (Eds.), *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics, Vol. VII* (pp. 1-6). Orlando, FL: International Institute of Informatics and Systemics.

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: Omnipress.

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.

Cox, M. T. (1997). An explicit representation of reasoning failures. In D. B. Leake & E. Plaza (Eds.), *Case-Based Reasoning Research and Development: Second International Conference on Case-Based Reasoning* (pp. 211-222). Berlin: Springer-Verlag.

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

Garey and Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New Jersey: Murray Hill. 68-69.

Hopcroft and Karp (1973). An $n^{2.5}$ algorithm for maximum matching in bipartite graphs. *Siam J. Comput. 2*, 225-231.

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.

Veloso, M. M., Carbonell, J. Perez, A., Borrajo, D., Fink, E., and Blythe, J. (1995). Integrating planning and learning: The PRODIGY Architecture. *Journal of Theoretical and Experimental Artificial Intelligence 7.*