

Control Rule Failure in State-Space Planning

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

By

KEVIN CLEEREMAN
B.Sc., Wittenberg University, 2002

2004

Wright State University

WRIGHT STATE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

May 28, 2004

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Kevin Cleereman ENTITLED Control Rule Failure In State-Space Planning BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Michael T. Cox, Ph.D.
Thesis Director

Forouzan Golshani, Ph.D.
Department Chair

Committee on
Final Examination

Michael T. Cox, Ph.D.

Krishnaprasad Thirunarayan, Ph.D.

Mateen Rizki, Ph.D.

Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

Abstract

Cleereman, Kevin. M.S., Department of Computer Science and Engineering, Wright State University, 2004. Control Rule Failure in State-Space Planning.

The general planning problem, the problem of finding a fully or partially ordered set of actions that transform an initial environment state into a final environment state, is at least PSPACE-Complete. As such, it is necessary to employ search controls to efficiently arrive at a solution to a planning problem. However, many autonomous planning systems use variants of the STRIPS or ADL domain languages that are specialized extensions of first order logic. Using theoretical and experimental results, this thesis will prove that first order logic is incapable of efficiently encoding the search controls that are necessary to solve the general bipartite graph-matching problem. The thesis will also demonstrate that bipartite graph-matching problems compose a subset of general planning problems and that common planning languages are therefore incapable of efficiently solving a significant number of problems. Finally, the thesis will introduce linear inequality control rules and will prove using theoretical and experimental results that they are able to encode search controls capable of efficiently solving all bipartite graph-matching problems.

Table of Contents

1	Introduction.....	1
2	State-Space Planning	7
3	Pathological Domains	20
4	Linear Inequality Control Rules.....	37
4.1	Experimental Setup.....	37
4.2	Experimental Results	63
5	Applications and Future Research	73
5.1	Stable-Matching.....	73
5.2	Parallel Planning and Multiagent Negotiation.....	82
5.3	Automatic and Mixed-Initiative Goal Transformations	88
6	Conclusion	97
7	References.....	100

List of Figures

Figure 1.1: Graphical representation of a marriage problem.....	3
Figure 1.2: Marriage problem partitioned among two virtual agents.	3
Figure 1.3: Pathological domain.....	5
Figure 2.1: Load-Truck Operator.....	9
Figure 2.2: Drive-Truck Operator.....	10
Figure 2.3: Unload-Truck Operator.....	10
Figure 2.4: Sample problem from the bipartite logistics domain.	12
Figure 2.5: Graphical representation of the bipartite logistics domain.....	13
Figure 2.6: Sample problem.....	13
Figure 2.7: Non-pathological version of Figure 1.3.	14
Figure 2.8: Linear inequality set for the 3-Pack Non-Pathological domain.	15
Figure 2.9: Control rule 1 for the Non-Pathological domain.....	16
Figure 2.10: Control rule 2 for the Non-Pathological domain.....	16
Figure 2.11: Control rule 3 for the Non-Pathological domain.....	16
Figure 2.12: Control rule 4 for the Non-Pathological domain.....	17
Figure 3.1: Pathological domain, isomorphic to Figure 1.3.	20
Figure 3.2: The linear inequalities for the 3-Pack Pathological domain.....	21
Figure 3.3: Control rule 3 for the 3-Pack Pathological domain.....	22
Figure 3.4: Control rule 4 for the 3-Pack Pathological domain.....	23
Figure 3.5: Control rule 5 for the 3-Pack Pathological domain.....	23

Figure 3.6: Initial State	24
Figure 3.7: State after one match	25
Figure 3.8: State after two matches	25
Figure 3.9: State after three matches	26
Figure 3.10: State after four matches	26
Figure 3.11: State after five matches	26
Figure 3.12: State after six attempted matches	27
Figure 3.13: 4-Pack Pathological domain.....	28
Figure 3.14: Linear inequality set for the 4-Pack Pathological domain.....	29
Figure 3.15: The new equation (top) is subsumed by an equation already in the set (bottom).....	29
Figure 3.16: 4-Pack-6-Truck Pathological domain.....	30
Figure 3.17: Linear inequality set for the 4-Pack-6-Truck Pathological domain.....	31
Figure 3.18: 3-Ary-4-Pack domain.....	32
Figure 3.19: Linear inequality set for the 3-Ary-4-Pack domain.....	32
Figure 3.20: 3-Color-3-Pack Pathological domain.....	34
Figure 3.21: First isomorphic 3-Pack Pathological sub-domain.....	35
Figure 3.22: Second isomorphic 3-Pack Pathological sub-domain.....	35
Figure 3.23: 5-Pack Pathological domain.....	36
Figure 4.1: Load-Truck-AB operator for the 5-Pack Pathological domain.....	39
Figure 4.2: Load-Truck-BC operator for the 5-Pack Pathological domain.....	39
Figure 4.3: Load-Truck-CD operator for the 5-Pack Pathological domain.....	40
Figure 4.4: Load-Truck-DE operator for the 5-Pack Pathological domain.....	40

Figure 4.5: Load-Truck-EA operator for the 5-Pack Pathological domain.	41
Figure 4.6: Unload-Truck-AB operator for the 5-Pack Pathological domain.	41
Figure 4.7: Unload-Truck-BC operator for the 5-Pack Pathological domain.....	42
Figure 4.8: Unload-Truck-CD operator for the 5-Pack Pathological domain.	42
Figure 4.9: Unload-Truck-DE operator for the 5-Pack Pathological domain.....	42
Figure 4.10: Unload-Truck-EA operator for the 5-Pack Pathological domain.....	43
Figure 4.11: Decoupled 3-Pack Pathological domain.....	43
Figure 4.12: Decoupled 4-Pack Pathological domain.....	44
Figure 4.13: Decoupled 5-Pack Pathological domain.....	44
Figure 4.14: Full linear inequality set for the 5-Pack Pathological domain.	45
Figure 4.15: Linear inequality set for the decoupled 5-Pack Pathological domain.	45
Figure 4.16: Package-A objects take first priority.	46
Figure 4.17: Package-B objects take second priority.....	46
Figure 4.18: Package-C objects take third priority.	47
Figure 4.19: Package-D objects take fourth priority.....	47
Figure 4.20: Package-E objects take fifth priority.	47
Figure 4.21: Package-A objects are first partitioned into the Package-AB1 subtype.....	48
Figure 4.22: Package-A objects are then partitioned into the Package-AB2 subtype.	48
Figure 4.23: Truck-AB matches take precedence over Truck-BC matches.	49
Figure 4.24: Truck-BC matches take precedence over Truck-CD matches.	49
Figure 4.25: Truck-CD matches take precedence over Truck-DE matches.	50
Figure 4.26: Truck-DE matches take precedence over Truck-EA matches.....	50
Figure 4.27: Initial state of an x2 problem in the 5-Pack Pathological domain.....	50

Figure 4.28: The planner has matched all Package-A objects.	51
Figure 4.29: The planner has matched all Package-B objects.	52
Figure 4.30: The planner has matched all Package-C objects.	52
Figure 4.31: The planner has matched all Package-D objects.	53
Figure 4.32: The planner has successfully matched all Package objects.	53
Figure 4.33: Reject invalid Truck-AB – Package-A matches.	54
Figure 4.34: Reject invalid Truck-AB – Package-B matches.	55
Figure 4.35: Reject invalid Truck-BC – Package-B matches.	55
Figure 4.36: Reject invalid Truck-BC – Package-C matches.	56
Figure 4.37: Reject invalid Truck-CD – Package-C matches.	56
Figure 4.38: Reject invalid Truck-CD – Package-D matches.	57
Figure 4.39: Reject invalid Truck-DE – Package-D matches.	57
Figure 4.40: Reject invalid Truck-DE – Package-E matches.	58
Figure 4.41: Reject invalid Truck-EA – Package-E matches.	58
Figure 4.42: Reject invalid Truck-EA – Package-A matches.	59
Figure 4.43: FOL-CR operator precedence control rule.	61
Figure 4.44: Initial state of sample problem A.	61
Figure 4.45: The FOL-CR search controls have maintained the linear inequality set.	62
Figure 4.46: Initial state of sample problem B.	62
Figure 4.47: The FOL-CR search controls have violated the linear inequality set.	62
Figure 4.48: Experimental results for the Full-CR domain set.	64
Figure 4.49: Experimental results for the Part-CR domain set.	64

Figure 4.50: Experimental results for the FOL-CR domain set, tracking only those problems solved within the node limit.....	65
Figure 4.51: Experimental results for the No-CR domain set, tracking only those problems solved within the node limit.....	65
Figure 4.52: Percentage of problems solved in the FOL-CR domain set.	66
Figure 4.53: Percentage of problems solved in the No-CR domain set.....	67
Figure 4.54: Experimental results for all problems from the FOL-CR domain set.....	68
Figure 4.55: Experimental results for all problems from the No-CR domain set.....	68
Figure 4.56: Experimental results for problems solved within the 20,000-node limit in the 3-Pack domain set.....	69
Figure 4.57: Experimental results for problems solved within the 20,000-node limit in the 4-Pack domain set.....	70
Figure 4.58: Experimental results for problems solved within the 20,000-node limit in the 5-Pack domain set.....	70
Figure 4.59: Experimental results for all problems in the 3-Pack domain set.....	71
Figure 4.60: Experimental results for all problems in the 4-Pack domain set.....	71
Figure 4.61: Experimental results for all problems in the 5-Pack domain set.....	72
Figure 5.1: 3-Pack Stable-Matching domain.	74
Figure 5.2: Parameter enumeration for the 2-Pack Stable-Matching domain.	76
Figure 5.3: Precedence orderings for the 2-Pack Stable-Matching domain.	77
Figure 5.4: Decomposed 3-Pack Stable-Matching domain.	78
Figure 5.5: Linear inequality control rule used for multiagent coordination in the 3-Pack Non-Pathological domain.	83

Figure 5.6: Agent-A's sub-domain.....	84
Figure 5.7: Agent-AB's sub-domain.....	84
Figure 5.8: Agent-BC's sub-domain.....	84
Figure 5.9: Full linear inequality set for Agent-AB's sub-domain from the 5-Pack Pathological domain.....	85
Figure 5.10: Reduced linear inequality set for Agent-AB's sub-domain from the 5-Pack Pathological domain.....	86
Figure 5.11: No-Soln problem for the 5-Pack Pathological domain.....	89
Figure 5.12: Automatic goal transformation for equation 1 (Figure 4.14).....	90
Figure 5.13: Automatic goal transformation for equation 2 (Figure 4.14).....	90
Figure 5.14: Automatic goal transformation for equation 3 (Figure 4.14).....	90
Figure 5.15: Automatic goal transformation for equation 4 (Figure 4.14).....	90
Figure 5.16: Automatic goal transformation for equation 5 (Figure 4.14).....	91
Figure 5.17: An automatic goal transformation (Figure 5.14) has deleted one Package-C object.....	91
Figure 5.18: Mixed-initiative goal transformation for equation 6 (Figure 4.14).....	92
Figure 5.19: Mixed-initiative goal transformation for equation 7 (Figure 4.14).....	92
Figure 5.20: Mixed-initiative goal transformation for equation 8 (Figure 4.14).....	92
Figure 5.21: Mixed-initiative goal transformation for equation 9 (Figure 4.14).....	93
Figure 5.22: Mixed-initiative goal transformation for equation 10 (Figure 4.14).....	93
Figure 5.23: The user has removed one Package-B object.....	94
Figure 5.24: N-ary linear inequality for the 5-Pack Pathological domain.....	94

Figure 5.25: Mixed-initiative control rule for the 5-Pack Pathological domain's n-ary linear inequality.	95
Figure 5.26: The user has removed one Package-E object.	95

List of Tables

Table 1: Edge weights for the 3-Pack Stable-Matching domain.	74
Table 2: A set of sample edge weights that permit an efficient solution using FOL control rules.	75
Table 3: A set of sample edge weights that do <i>not</i> permit an efficient solution using FOL control rules.	75
Table 4	76

Acknowledgments

This research is funded by grants #665485 and #665784 from the Wright Brothers Institute – ITRI, by BSES subcontract #Z71000 (WSU #66) from Ball Aerospace and Technologies Corporation, by the Ohio Board of Regents, and by a full scholarship from DAGSI. I thank my thesis advisor Dr. Michael T. Cox, and my committee members Dr. Krishnaprasad Thirunarayan and Dr. Mateen Rizki. I also thank Mr. Brenton Bostick and the Wright State University Writing Center for their valuable input.

1 Introduction

Multiagent computing is a powerful paradigm that allows us to break up difficult problems in order to solve them in a more efficient manner and to integrate resources otherwise separated by logical and/or geographic barriers. One of the most important properties of a multiagent system, arguably *the* most important property of any multiagent system, is the communication protocol that links disparate agents into one cohesive system. A protocol lacking adequate expressive power will severely impair a system's effectiveness, yet a protocol lacking succinctness will severely impair the system's designer. For these reasons we committed to research into multiagent system communication protocols (particularly protocols devoted to multiagent planning systems) with the intention of developing a protocol that struck a balance between expressive power and succinctness.

As a first step towards developing this protocol, we examined logically distributed planning problems on a single-agent system. We partitioned and then allocated these problems to virtual agents, with the resulting system mimicking the communicative aspects of a multiagent system while eliminating any possible interference from the operating system or computer network. The particular system under study had a static population of virtual agents that mimicked the behavior of a static population of true

agents. For example, given a single-agent system capable of performing the actions “walk” and “chew gum,” we would assign the “walk” action to one virtual agent and the “chew gum” action to another virtual agent. Given that the “walk” and “chew gum” actions are logically independent (i.e., given that a system can walk and chew gum at the same time), our single-agent system would be logically equivalent to a two-agent multiagent system when broken up along such functional boundaries.

The planning problems studied in this thesis belong to a particular class of domains that correspond to bipartite graph-matching problems (Asratian *et al*, 1998), which are commonly known as marriage problems (Gusfield and Irving, 1989). The classic bipartite graph-matching problem is, given a set of girls and boys, to marry off (match) every girl to a boy with whom she is acquainted. This problem is trivial if every girl knows every boy and if there are at least as many boys as girls, for then any set of marriages would suffice. However, this problem is much more complex if the girls only know a subset of the boys, perhaps as a result of traveling in different social circles or of living in different towns. We can solve the general bipartite graph-matching problem in polynomial time, although we note that the tripartite graph-matching problem is much more computationally complex.

The specific marriage problems we studied involve sets of towns each containing a subset of the boys and girls whose marriages we are arranging. Each town contains either all boys or all girls, and if two towns are adjacent, then all of the boys and girls in the two towns are acquainted with each other. For example, Figure 1.1 illustrates a problem

containing the all-boys towns Bayville and Boston and the all-girls towns Greenfield and Glendale. Bayville is adjacent to both Greenfield and Glendale while Boston is only adjacent to Glendale, and so the boys in Bayville may marry girls from Greenfield or from Glendale but the boys in Boston may only marry girls from Glendale.¹

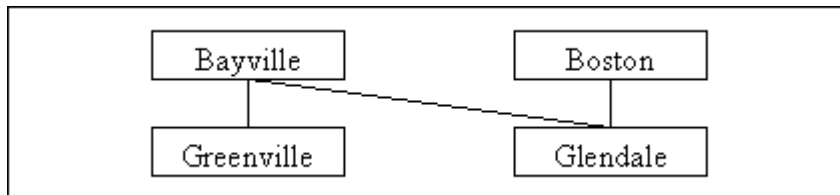


Figure 1.1: Graphical representation of a marriage problem.

This is an example of a single-agent system with a single goal: find suitable brides for all of the boys. However, we can further decompose the system into a multiagent system composed of two virtual agents (Figure 1.2), one tasked with finding brides for all of the boys in Bayville and the other tasked with finding brides for all of the boys in Boston. (Alternatively, we could have decomposed the system along the logical divide between Greenville and Glendale instead of the logical divide between Bayville and Boston.)

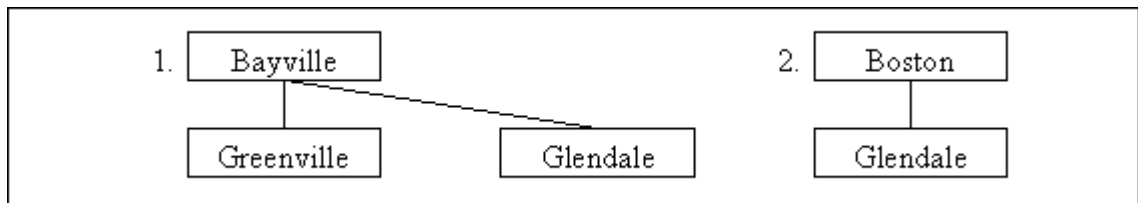


Figure 1.2: Marriage problem partitioned among two virtual agents.

This decomposition leaves us with a virtual multiagent system that is equivalent to the original single-agent system but with the exception that the resource integration points are now explicitly noted by duplicate labels. The Bayville-agent and the Boston-agent

¹ Note that we have specified that these towns are gender-homogenous to simplify the resulting graph, because a bipartite graph cannot have an edge connecting a node to itself (e.g., if Bayville contained both boys and girls, then we could not draw an edge connecting Bayville to itself, but instead we would have to split Bayville into two separate nodes).

are both capable of drawing bachelorettes away from Glendale, and so they must negotiate with each other to ensure that they do not inadvertently draw away so many Glendale bachelorettes that the system fails to accomplish its goals. For example, if there is one boy in Bayville, one boy in Boston, one girl in Greenville, and one girl in Glendale, then the Boston-agent will not be able to find brides for all of the boys in Boston if the Bayville-agent matches the girl in Glendale to the boy in Bayville.

Inter-agent negotiation in many such virtual multiagent systems is trivial, particularly for small toy systems like the one depicted in Figure 1.2. In the system above, if a marriage problem has a solution, then it is not possible for the Boston-agent to impede the performance of the Bayville-agent, and the Bayville-agent can avoid impeding the Boston-agent by marrying off the Bayville boys to the Greenville girls before marrying the Bayville boys off to the Glendale girls. Thus, the planner only needs to employ simple search controls to efficiently solve problems in this domain.

However, such simple search controls do not work in all planning domains. Some domains contain *pathological dependency cycles*, which are cycles arising in incomplete subgraphs within the domain's graphical representation. A domain containing at least one pathological dependency cycle is called a *pathological domain*. Inter-agent communication becomes much more complex in these pathological domains (Figure 1.3).

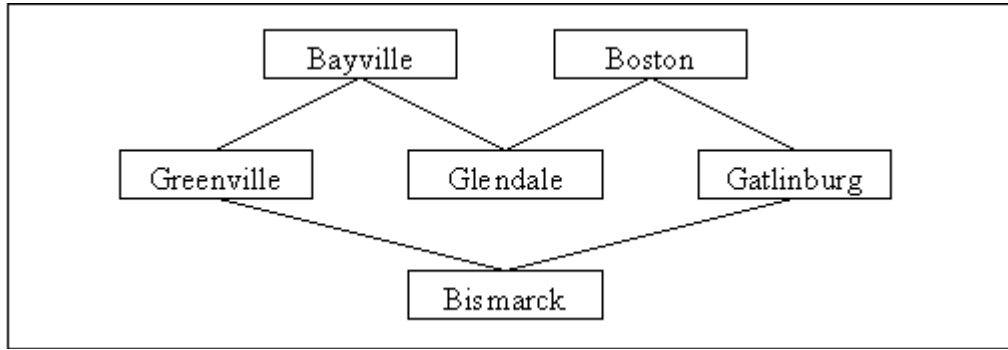


Figure 1.3: Pathological domain.

We will be returning to the pathological planning domain in Figure 1.3 throughout this thesis. For now, we will note that the important feature is that there are three nodes for each object class in the domain graph, i.e., 3 towns of boys and 3 towns of girls. This domain is pathological because the six graph edges form a pathological dependency cycle, and thus there is no inherently “safe” set of matches for the system’s virtual agents. That is, any given Boy-Girl marriage has the potential of impeding upon another virtual agent if the planner does not use appropriate search controls to restrict these potential marriages.

Planning algorithms do not possess the expressive power needed to efficiently guide search in pathological domains because these algorithms typically use variants of the STRIPS or ADL languages, which are specializations of first order logic (FOL). Even those algorithms whose domain languages use a logical representation that is more powerful than FOL (e.g., the first order temporal logic used in TLPlan) lack the power needed to efficiently guide search in pathological domains. If the communication protocol between a single-agent planning system’s component virtual agents is inadequate, then so too is the communication protocol between a multiagent planning

system's component agents. Our solution and contribution to this efficiency problem is the integration of linear constraints to planners' search controls, resulting in vast improvements in planning efficiency.

In the next chapter we will briefly introduce the PRODIGY state-space planner, and will discuss the marriage problem as it relates to state-space planning. The choice of planners is relatively unimportant, because all planning algorithms lack the expressive power necessary to use linear constraints in search controls. In Chapter 3, we will discuss the difference between Pathological and Non-Pathological domains, and will examine in detail the cause of inter-agent negotiation failure between the virtual agents in pathological domains such as the one depicted in Figure 1.3. In Chapter 4, we will provide a solution to the inter-agent negotiation problem for pathological domains using Linear Inequality (linear constraint) control rules (search controls), and will provide empirical results to demonstrate our solution's efficacy. In Chapter 5, we will discuss the application of Linear Inequality control rules towards a variety of planning problems, such as goal transformations and mixed-initiative planning. Finally, we will give our conclusions in Chapter 6.

2 State-Space Planning

State-space planners seek to produce a sequence of actions that will transform a domain environment from an initial state to a desired goal state. A planning domain will contain a set of operators that effect these environment transformations such that each operator contains a precondition list (i.e., the environment state in which the operator is applicable) and an effects list (i.e., the environment transformation effected by the operator). For state-space planners using domain languages that are specializations of FOL (which is the vast majority of state-space planners), the precondition list contains a list of predicates that must be true in the current environment state for the operator to be applicable. The effects list contains a set of predicates that the planner adds to the environment when the operator fires (the add list) and a set of predicates that the planner deletes from the environment when the operator fires (the delete list).

State-space planners typically generate the necessary action (operator) sequence through forward chaining, backward chaining, or both. Forward chaining begins the search process at the current environment state and branches on those operators whose preconditions are satisfied, then continues to expand the search tree until the current environment state matches the goal state. Backward chaining begins the search process at the goal state and branches on those operators whose effects lists will effect the

necessary change(s) in the world state to transform the current environment state into the goal state, and continues to expand the search tree until the current environment state matches the initial state.

Prodigy 4.0 (Veloso, Carbonell, Perez, Borrajo, Fink, and Blythe, 1995) is a state-space planner that employs both backward chaining and forward chaining in its tree search. There are four decision points in the planning cycle (or branch points on the search tree): goal selection (determining which predicate to add to or delete from the current environment state), operator selection (determining which operator to use to satisfy the selected goal), binding selection (determining which objects to bind to the selected operator), and the decision to either immediately apply an instantiated operator (i.e., forward-chain) or to subgoal (i.e., backward-chain). The general planning problem is at least PSPACE-Complete (Bylander, 1991; Chapman, 1987; Selman, 1994), so the use of search controls to guide search in PRODIGY's search tree is essential to effective planning.

Veloso's logistics domain (Veloso, 1994) is of particular interest to our research. This domain generates plans to deliver packages to various locations using trucks (which move packages between locations within cities) and airplanes (which move packages between cities). We shall examine a simplified version of this domain, henceforth called the *bipartite logistics domain*, which contains only trucks. A specialized bipartite graph-matching algorithm can solve problems in this and other bipartite domains in $O(n^{2.5})$ time

(Hopcroft and Karp, 1973), though of course a bipartite graph-matching algorithm lacks the general problem-solving capabilities of a planning algorithm.

This domain's operators are the Load-Truck operator (Figure 2.1), the Drive-Truck operator (Figure 2.2), and the Unload-Truck operator (Figure 2.3). The Load-Truck operator will move an unmatched Package object from outside of an available Truck to inside of the Truck, the Drive-Truck operator will move a Truck object from one Location to a different Location, and the Unload-Truck operator will move a Package object from inside of a Truck to outside of the Truck. Note that, like the Marriage domain and the bipartite logistics domain, a single Truck object can move at most one Package object.

```
(Operator LOAD-TRUCK
  (<obj> <trk> <loc>)
  (precondition
    ((<obj> PACKAGE)
     (<trk> TRUCK)
     (<loc> LOCATION))
    (and
      (at-truck <trk> <loc>)
      (at-obj <obj> <loc>)
      (is-ready <trk>)))
  (effects ()
    (del (at-obj <obj> <loc>))
    (del (is-ready <trk>))
    (add (inside-truck <obj> <trk>))))
```

Figure 2.1: Load-Truck Operator.

```

(Operator DRIVE-TRUCK
 (<trk> <loc1> <loc2>)
 (precondition
  ((<trk> TRUCK)
   (<loc1> LOCATION)
   (<loc2> (and LOCATION (diff <loc1> <loc2>))))))
 (and
  (at-truck <trk> <loc1>)))
 (effects ()
  (del (at-truck <trk> <loc1>))
  (add (at-truck <trk> <loc2>))))

```

Figure 2.2: Drive-Truck Operator.

```

(Operator UNLOAD-TRUCK
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> PACKAGE)
   (<trk> TRUCK)
   (<loc> LOCATION))
  (and
   (inside-truck <obj> <trk>)
   (at-truck <trk> <loc>)))
 (effects ()
  (del (inside-truck <obj> <trk>))
  (add (at-obj <obj> <trk>))))

```

Figure 2.3: Unload-Truck Operator.

The syntax of the operators used in the bipartite logistics domain (and in all other logistics domains discussed in this thesis) is straightforward. The operator contains a list of variables used within the operator: for example, in the Load-Truck operator, the variables are <obj>, <trk>, and <loc>. The precondition list follows, beginning with the set of allowable bindings for the declared variables: for example, in the Load-Truck operator, <obj> must be an object of type Package, and <trk> must be an object of type Truck. The second part of the precondition list contains the set of predicates that must be true or false in the current environment state. For example, in the Load-Truck operator,

the current environment state must contain the predicate “at-truck” applied to the objects bound to the <trk> and <loc> variables, so if the object Truck1 were bound to <trk> and the object Loc2 were bound to <loc> then the current environment state would need to contain the predicate `at-truck(Truck1 Loc2)` for the planner to apply the operator. Finally, the effects list contains the set of predicates that the planner adds to or deletes from the current environment state when the operator fires. For example, the Load-Truck operator will delete the predicate `is-ready(Truck1)` from the current environment state, and will add the predicate `inside-truck(Package4 Truck1)` to the current environment state. We will be using slightly more complex variable declarations in the domains discussed in Chapter 0, but we will not otherwise increase the complexity of these domain operators.

Problems consist of an object list, the predicates contained in the initial state, and the predicates contained in the goal state. PRODIGY employs a closed-world assumption, so it assumes that all world predicates are false unless otherwise given as true. Negative predicates are therefore unnecessary in giving the initial state, but are necessary in giving negative goals (e.g., “do not have obj1 at loc1,” written $(\sim (\text{at-obj obj1 loc1}))$).

Figure 2.4 contains a sample problem for the bipartite logistics domain. The problem has one Truck object, one Package object, and two Location objects. The truck begins at Location1, the package begins at Location2, and the goal is to move the package to Location1.

```
(Objects
  (truck1 TRUCK)
  (obj1 PACKAGE)
  (loc1 LOCATION)
  (loc2 LOCATION))
(Initial-State
  (at-truck truck1 loc1)
  (at-obj obj1 loc2)
  (is-ready truck1))
(Goal
  (at-obj obj1 loc1))
```

Figure 2.4: Sample problem from the bipartite logistics domain.

One solution to this problem is

```
DRIVE-TRUCK truck1 loc1 loc2
LOAD-TRUCK truck1 obj1 loc2
DRIVE-TRUCK truck1 loc2 loc1
UNLOAD-TRUCK truck1 obj1 loc1
```

The planner relocates the Truck object to Location2, loads the Package into the Truck, drives the Truck (with Package inside) back to Location1, and unloads the Package.

Note that the Load-Truck operator disables itself (the result of our modification that converts the logistics domain into the bipartite logistics domain), so a single Truck can load at most one Package per problem. If we abstract away the Location field from the Load-Truck operator by acting on the assumptions that all Truck objects begin in an is-ready state, that none of the Package objects begin in their final state, and that all Truck objects are able to move to any Location, then we can use the same graphical domain representation as we used in Figure 1.1, Figure 1.2, and Figure 1.3 (Figure 2.5).

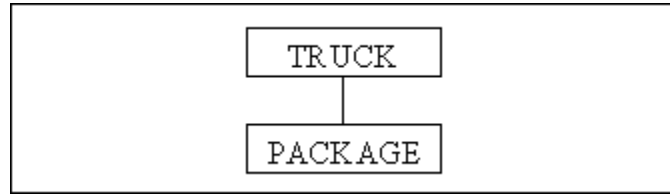


Figure 2.5: Graphical representation of the bipartite logistics domain.

To represent problems and not just domains, we make a minor extension on this representation. For example, Figure 2.6 represents a problem in the bipartite logistics domain containing four Truck objects and three Package objects. The numbers bound to the object subtypes denote the quantity of available objects of the given subtype.

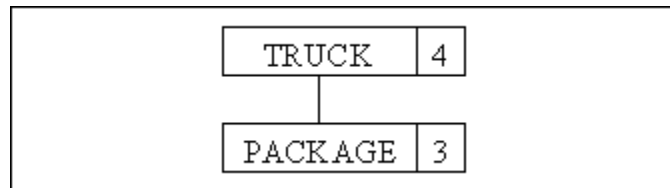


Figure 2.6: Sample problem.

The bipartite logistics domain can benefit from the addition of control rules, but these rules are not needed for the actual graph-matching portion of the problem but instead maneuver through the vagaries of planning algorithms. The control rules in this domain are not needed to order the goal, operator, and binding orderings, but instead are only needed to prematurely reject invalid branches of the search tree that cannot possibly be used in a valid solution.² However, a more complex form of the logistics domain will benefit from the use of control rules in the graph-matching portion of the problem.

² For example, in the Load-Truck operator in Figure 2.1, the truck and package must be co-located for the package to be loaded. If the truck and package are in different locations, then PRODIGY will attempt to either move the truck object to the same location as the package object, or else it will attempt to move the package object to the same location as the truck object. However, if the planner attempts to move the package object to the same location as the truck object then it will run into a search loop and must backtrack. We can use control rules to force PRODIGY to move the truck to the same location as the package to avoid these goal loops.

Figure 2.7 has the graphical representation of a non-pathological version of the domain given in Figure 1.3 but with Truck and Package subtypes instead of Town subtypes.

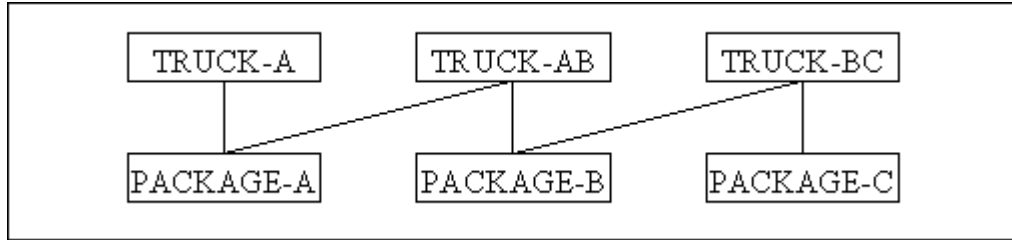


Figure 2.7: Non-pathological version of Figure 1.3.

This Non-Pathological domain has three Load-Truck operators and three Unload-Truck operators, one per Truck subtype. These six operators will only allow permissible package bindings. For example, the operator Load-Truck-AB has the binding declarations

```
(Operator LOAD-TRUCK-AB
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-A PACKAGE-B))
   (<trk> TRUCK-AB))
```

Thus, the object may be of type Package-A or type Package-B but not type Package-C, corresponding to the outgoing edges of the center-top node in Figure 2.7.

Just as with the domains represented in Figure 1.1 and Figure 1.2, the Non-Pathological domain in Figure 2.7 requires search controls to avoid needless backtracking. The operators overlap such that an inappropriate application of one operator may disable another operator, for example, if the Load-Truck-AB operator employs too many

Package-A bindings then the Load-Truck-A operator may be inadvertently disabled, and if the Load-Truck-AB operator employs too many Package-B bindings then the Load-Truck-BC operator may be inadvertently disabled. Thus, to guide the planner's search, we define the variables ta , tab , tbc , pa , pb , pc , to respectively denote the number of Truck-A, Truck-AB, Truck-BC, Package-A, Package-B, and Package-C objects that have not been matched in the current world state. Then we define a set of equations whose consistency the planner must maintain to avoid backtracking (Figure 2.8). These equations are the *linear inequality* equation set.

1.	$ta + tab \geq pa$
2.	$tab + tbc \geq pb + pc$
3.	$tbc \geq pc$
4.	$ta + tab + tbc \geq pa + pb + pc$

Figure 2.8: Linear inequality set for the 3-Pack Non-Pathological domain.

According to Hall's Theorem (Nering, 1993), satisfaction of this equation set constitutes a necessary and sufficient condition for a solution existing to the problem of matching all packages in a bipartite logistics domain. Therefore, if one assumes that the equations are satisfied in the initial state (and that a solution must therefore exist), one can employ control rules to ensure that the equations will still be satisfied by any changes to be made to the initial and subsequent states. For example, before matching a Truck-AB object to a Package-A object, the planner must first ensure that it will not violate equation 2 by decrementing tab . It is unnecessary to confirm that the planner will not violate equation 1, for both tab and pa would be decremented.

If the planner uses the following control rules (Figure 2.9, Figure 2.10, Figure 2.11, Figure 2.12) then it will ensure that it will not violate these linear inequalities during search.

1.

```
(if (and
    (candidate-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A))
    (then select goal (at-obj <obja> <loc>)))
```

Figure 2.9: Control rule 1 for the Non-Pathological domain.

2.

```
(if (and
    (candidate-goal (at-obj <objb> <loc>))
    (type-of-object <objb> PACKAGE-B)
    (~ (candidate-goal (at-obj <obja> <loc>)))
    (type-of-object <obja> PACKAGE-A))
    (then select goal (at-obj <objb> <loc>)))
```

Figure 2.10: Control rule 2 for the Non-Pathological domain.

3.

```
(if (and
    (current-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A)
    (known (is-ready <trka>))
    (type-of-object <trka> TRUCK-A))
    (then select operator UNLOAD-TRUCK-A))
```

Figure 2.11: Control rule 3 for the Non-Pathological domain.

4.

```
(if (and
    (current-goal (at-obj <objb> <loc>))
    (type-of-object <objb> PACKAGE-B)
    (known (is-ready <trkab>))
    (type-of-object <trkab> TRUCK-AB))
    (then select operator UNLOAD-TRUCK-AB))
```

Figure 2.12: Control rule 4 for the Non-Pathological domain.

Control rule 1 (Figure 2.9) ensures that Package-A objects are loaded before all other objects. It accomplishes this task by determining if (at-obj Package-A location) is a candidate goal, i.e., a goal that PRODIGY can choose to pursue in the present environment state. If it is a candidate goal, then the planner selects it and rejects all other candidate goals.

Control rule 2 (Figure 2.10) ensures that all Package-B objects are loaded before all Package-C objects, but after all Package-A objects. If there are candidate (at-obj Package-B location) goals, and there are no candidate (at-obj Package-A location) goals, then PRODIGY selects a (at-obj Package-B location) goal and rejects all others.

Control rule 3 (Figure 2.11) ensures that Truck-A objects are preferred over Truck-AB objects in moving Package-A objects. If there are is-ready (i.e., unmatched) Truck-A objects, then the planner selects the Unload-Truck-A operator to satisfy the (at-obj Package-A location) goal. This prevents a Truck-AB object from satisfying the goal instead.

Finally, control rule 4 (Figure 2.12) ensures that Truck-AB objects are preferred over Truck-BC objects in moving Package-B objects. If there are is-ready Truck-AB objects, then the planner selects the Unload-Truck-AB operator to satisfy the goal, and rejects the Unload-Truck-BC operator.

Thus, the planner begins the planning process by matching Truck-A objects to Package-A objects and decrementing the values of ta and pa accordingly.

$$1. \quad ta-- + tab \geq pa--$$

The relevant equations remain consistent after the values of ta and pa are decremented. The values of ta and pa continue to be decremented until either all Package-A objects have been matched or all Truck-A objects have been matched. In the former case, the first equation is no longer relevant because the planner has moved on to moving Package-B objects. In the latter case, we rewrite the first equation to

$$1. \quad tab \geq pa$$

because $ta == 0$.

The planner will then proceed to match the remaining Package-A objects to Truck-AB objects, which reduces the values of *tab* and *pa* accordingly. We only show equation 2 below, for equation 2 subsumes equation 1.

$$2. \quad tab-- + tbc \geq pa-- + pb$$

Again, the planner has maintained the consistency of the equations. The Non-Pathological domain is “non-pathological” precisely because it is possible for Prodigy 4.0 to efficiently use its built-in control rule capabilities to ensure that it will not need to backtrack during the search process. Note that PRODIGY’s control rules have the expressive power of first order logic, insofar as they can express existential quantifiers (“there exists a Truck-A object that is ready”) but cannot express metrics (“there exists at least three and no more than five Truck-A objects that are ready”).

We have discussed the necessity of search controls in state-space planning and the relationship between graph theory and state-space planning, and we have introduced the non-pathological bipartite logistics domain. In the next chapter we will introduce the pathological bipartite logistics domain and will discuss the aspects of state-space planning that cause this domain to be “pathological.”

3 Pathological Domains

In terms of search control, the difference in complexity between the Non-Pathological bipartite logistics domain (Figure 2.7) and the Pathological bipartite logistics domain (Figure 3.1, isomorphic to the Pathological domain given in Figure 1.3) is surprisingly great, despite our adding only one more edge to the graph. This complexity is due to the emergence of a *pathological dependency cycle*, a set of cyclical dependencies in the domain's linear inequalities. A domain is a *pathological domain* (Cleereman and Cox, 2004b) if it contains one or more pathological dependency cycles.

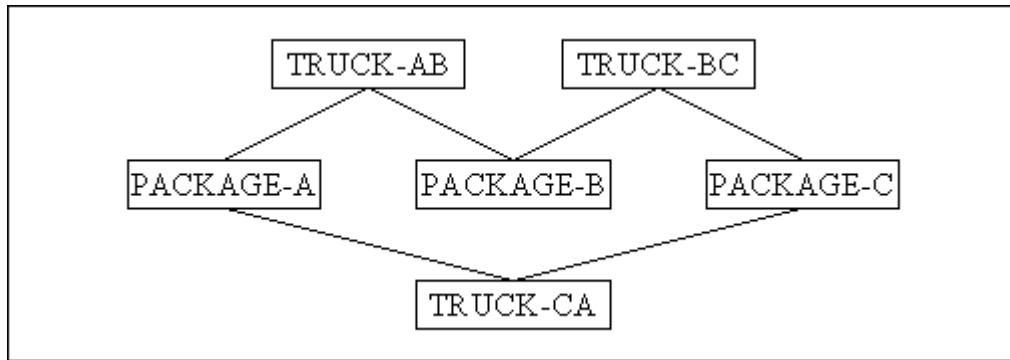


Figure 3.1: Pathological domain, isomorphic to Figure 1.3.

Henceforth we will refer to the Non-Pathological bipartite logistics domain as the 3-Pack Non-Pathological domain and to the Pathological bipartite logistics domain as the 3-Pack Pathological domain, because these domains both contain three Package subtypes and three Truck subtypes. In general, an i -Pack domain contains i Package subtypes and i Truck subtypes.

In the Non-Pathological domain a Truck-A object could only move Package-A objects, but the Pathological domain has replaced the Truck-A subtype with the Truck-CA subtype, and Truck-CA objects are able to move either Package-A objects or Package-C objects. We give the resulting linear inequalities for this domain in Figure 3.2. The ta variable used in Figure 2.8 to denote the number of unmatched Truck-A objects has been replaced with a tca variable to denote the number of unmatched Truck-CA objects.

1.	$tca + tab \geq pa$
2.	$tab + tbc \geq pb$
3.	$tbc + tca \geq pc$
4.	$tca + tab + tbc \geq pa + pb + pc$

Figure 3.2: The linear inequalities for the 3-Pack Pathological domain.

Implicit consistency checks are no longer sufficient to maintain this system of equations. If we use the same control rules for this domain as we did for the previous domain (replacing instances of Truck-A with instances of Truck-CA) then we may inadvertently violate equation 3 when assigning Truck-CA objects to Package-A objects. (Note that we only apply the decrement operator to the variable tca once per match. We show the operator in both equation 1 and equation 3 to demonstrate that the value of tca has changed for both equations.)

1.	$tca-- + tab \geq pa--$
3.	$tbc + tca-- \geq pc$

Although it is a given that $tbc + tca \geq pc$, it is *not* a given that $tbc + tca - 1 \geq pc$. The planner must explicitly confirm that equation 3 is not being violated by counting the number of available Truck-BC objects, the number of available Truck-CA objects, and the number of unmatched Package-C objects, before committing to a Truck-CA – Package-A match.

Thus if our control rules are written in domain languages that are specializations of FOL then we can no longer use practical search controls to guarantee that backtracking will not occur, because unguided search has exponential complexity while effective FOL search controls have super-exponential complexity (Fischer and Rabin, 1974). We require control rules capable of making efficient consistency checks if we are to effectively guide search in this domain. For example, if we rewrite control rule 3 from Figure 2.11 (see Figure 3.3), control rule 4 from Figure 2.12 (see Figure 3.4), and add a new control rule 5 (see Figure 3.5), then we will once again ensure that no backtracking occurs and will therefore ensure that we are planning efficiently.

3a.

```
(if (and
  (current-goal (at-obj <obja> <loc>))
  (type-of-object <obja> PACKAGE-A)
  (known (is-ready <trkca>))
  (type-of-object <trkac> TRUCK-CA))
  (then prefer operator
    Unload-Truck-CA Unload-Truck-AB))
```

Figure 3.3: Control rule 3 for the 3-Pack Pathological domain.

4a.

```
(if (and
    (current-goal (at-obj <objb> <loc>))
    (type-of-object <objb> PACKAGE-B)
    (known (is-ready <trkab>))
    (type-of-object <trkab> TRUCK-AB)
    (known (is-ready <trkbc>))
    (type-of-object <trkbc> TRUCK-BC))
    (then prefer operator
      Unload-Truck-AB Unload-Truck-BC))
```

Figure 3.4: Control rule 4 for the 3-Pack Pathological domain.**5.**

```
(if (and
    (current-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A)
    (~ (greater-than (TRUCK-CA TRUCK-BC) PACKAGE-C)))
    (then reject operator Unload-Truck-CA))
```

Figure 3.5: Control rule 5 for the 3-Pack Pathological domain.

The meta-predicate “greater-than” takes as arguments a list of Truck subtypes ($\{\text{Truck-CA, Truck-BC}\}$) and a single Package subtype (Package-C), and returns true if the number of Truck objects of the given subtypes for which (is-ready Truck) is true-in-state exceeds the number of Package objects of the given subtype, else returns false. Thus, control rule 5 serves as explicit confirmation of equation 3 given in Figure 3.2.

The modified versions of control rules 3 and 4 are functionally equivalent to their original versions in that we have not altered the binding precedence of either. The difference is simply one of control rule precedence, for PRODIGY applies reject rules before it applies prefer rules but after it applies select rules. Control rule 5 has greater precedence than control rule 3a (Figure 3.3) and is able to explicitly confirm the linear inequality set, but

control rule 5 has lesser precedence than control rule 3 (Figure 2.11) and will not fire in time to explicitly confirm the linear inequality set. We have rewritten control rule 4 to keep it consistent with control rule 3.

To illustrate the difference in control rule functionality, we will show a short planning run using two different sets of control rules. The first domain (on the left) will use control rules 1, 2, 3a, 4a, and 5, while the second domain (on the right) will use control rules 1, 2, 3, and 4. The problem illustrated here is a simple matching problem: we must match all Package objects to the available Truck objects.

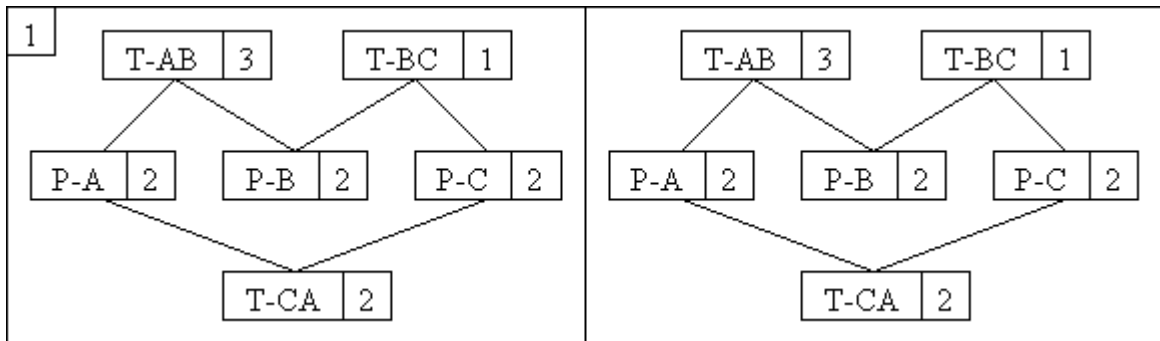


Figure 3.6: Initial State

Figure 3.6. In the initial state, there are two Packages of each subtype, two Truck-CA objects, three Truck-AB objects, and one Truck-BC object. We have abbreviated the type identifiers, e.g., Truck-AB becomes T-AB, and Package-A becomes P-A. Note that the linear inequalities in Figure 3.2 are all satisfied in this state.

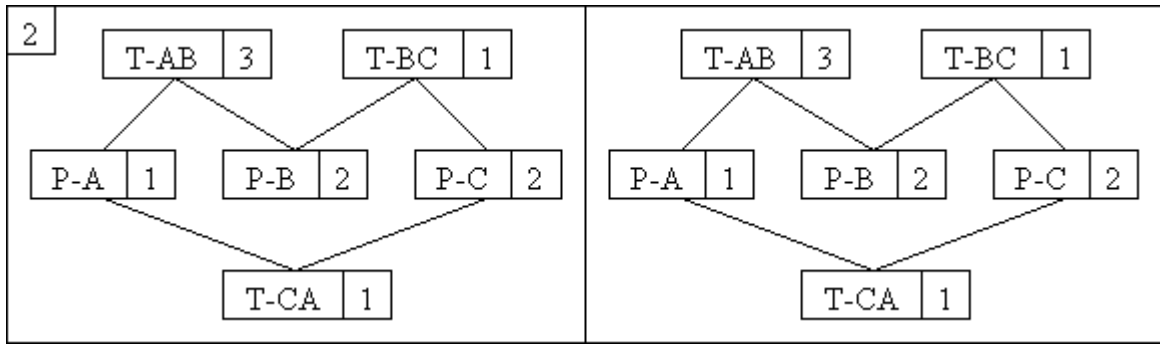


Figure 3.7: State after one match

Figure 3.7. After one cycle, the planner has matched a single Truck-CA object to a single Package-A object. The linear inequality set is still satisfied in this state.

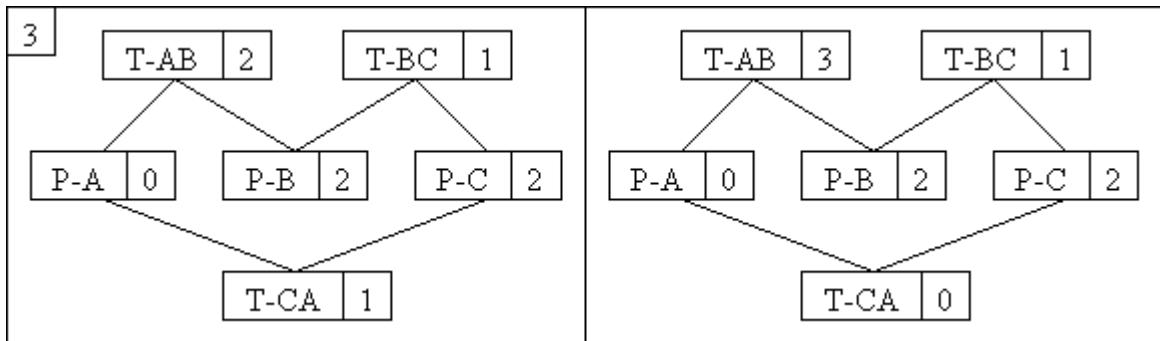


Figure 3.8: State after two matches

Figure 3.8. In the second cycle, control rule 5 rejects the binding of the remaining Truck-CA object to the remaining Package-A object in the domain on the left. Without control rule 5, the planner matches the remaining Truck-CA object to the remaining Package-A object in the domain on the right. Note that the linear inequality set is still satisfied for the domain on the left. However, equation 3 on the right is now violated, because there are no longer enough Truck-BC and Truck-CA objects to match all of the Package-C objects.

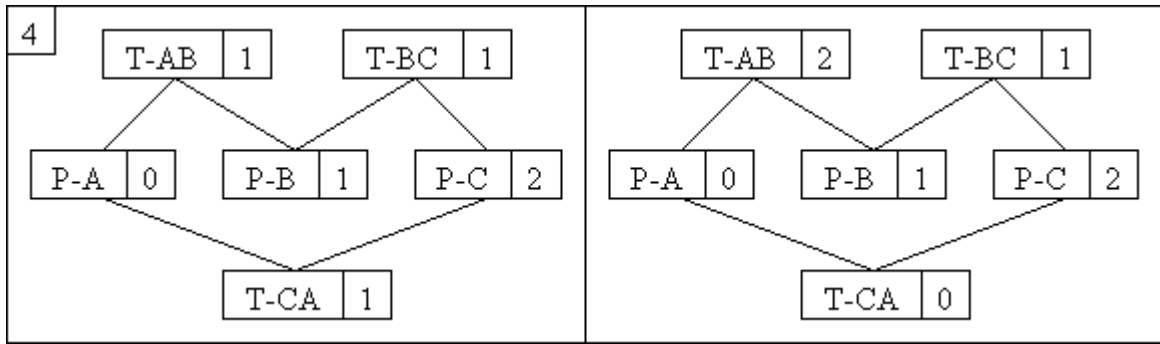


Figure 3.9: State after three matches

Figure 3.9. In the next cycle, the planner matches a Truck-AB to a Package-B object.

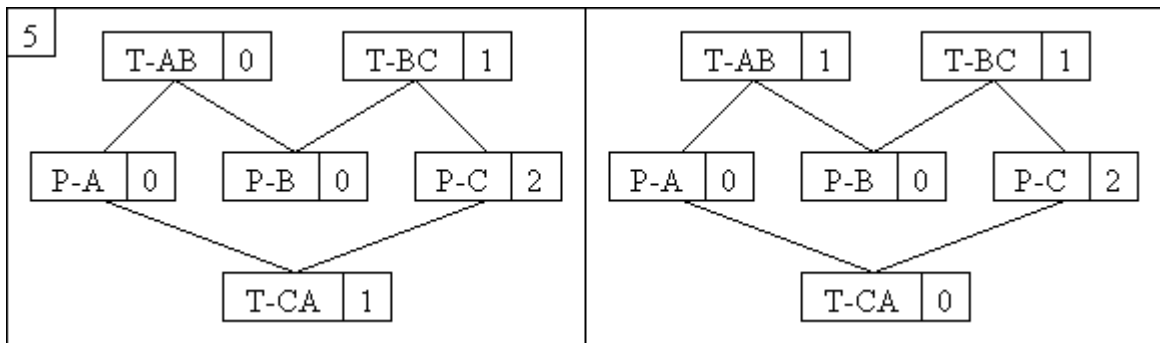


Figure 3.10: State after four matches

Figure 3.10. In the next cycle, the planner matches a Truck-AB to a Package-B object.

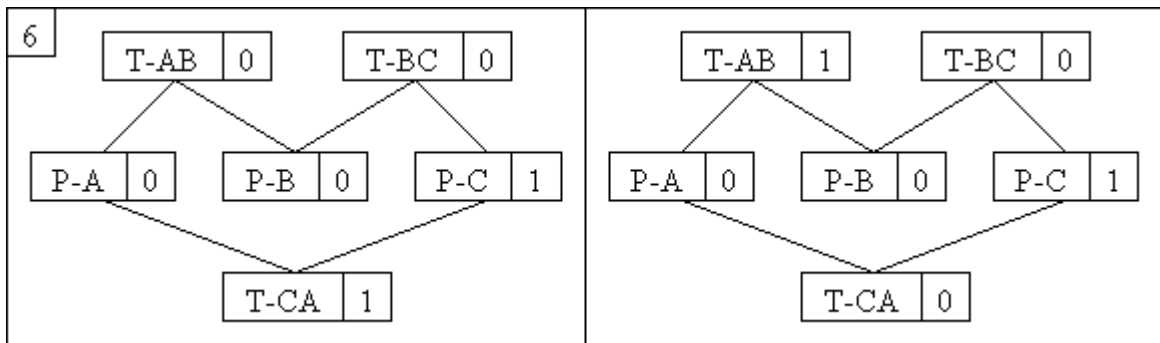


Figure 3.11: State after five matches

Figure 3.11. In the next cycle, the planner matches a Truck-BC to a Package-C object.

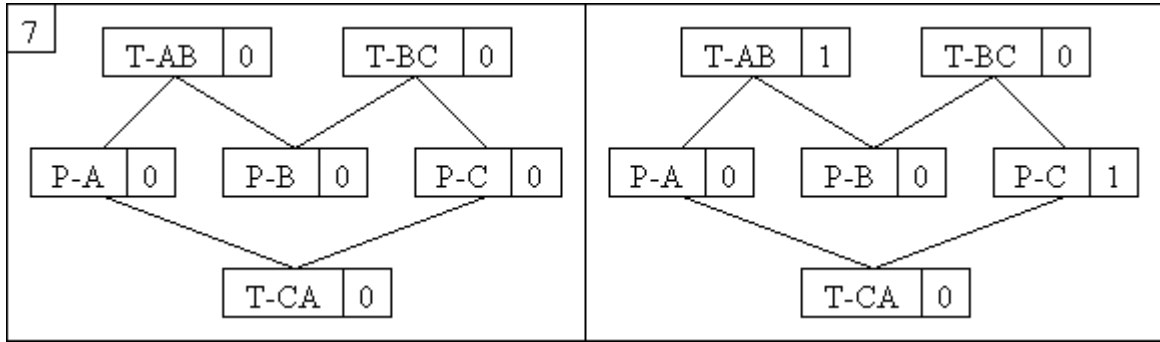


Figure 3.12: State after six attempted matches

Figure 3.12. The domain on the left employing control rule 5 matches the final Package object to the remaining Truck-CA object in the final planning cycle. However, the domain on the right that did not employ control rule 5 must now backtrack. Note that planning failure on the right was a foregone conclusion in Step 3 (Figure 3.8), but that the planner did not detect it until Step 7.

The pathological dependency cycle in the 3-Pack Pathological domain exhibits itself in the interplay between equations 1, 2, and 3 in the linear inequality set. It is not possible to decrement a value on the left-hand side of an equation (i.e., to match a Truck object to a Package object) without also decrementing the same value on the left-hand side of another equation on which the right-hand side is *not* being decremented. For example, if the planner decrements the value of *tca* (after matching a Truck-CA object to a Package object) then this affects both equation 1 and equation 3. However, the planner then decrements the value on the right-hand side (*pa* or *pc*) of only *one* of these equations, leaving the possibility that the match has violated the other equation unless search control explicitly verifies that the inequality holds.

Note that this fault rests in the power of classical planning languages, and is *not* unique to the PRODIGY planner. Partial-order planners such as UCPOP (Penberthy and Weld, 1992) are subject to the same efficiency problems as full-order planners such as PRODIGY. Graphplan (Blum and Furst, 1997) and Graphplan extensions such as STAN (Long and Fox, 1999) also require further extensions to their logical representations to efficiently represent the linear inequalities required by Hall's Theorem. Even many algorithms that employ extended logical representations, such as the first order temporal logic representation used in TLPlan (Bacchus and Kabanza, 2000), use logical representations that are still incapable of practically representing a simple linear inequality.

The problems arising from pathological dependency cycles are present in many domains, among them the 4-Pack Pathological domain (Figure 3.13).

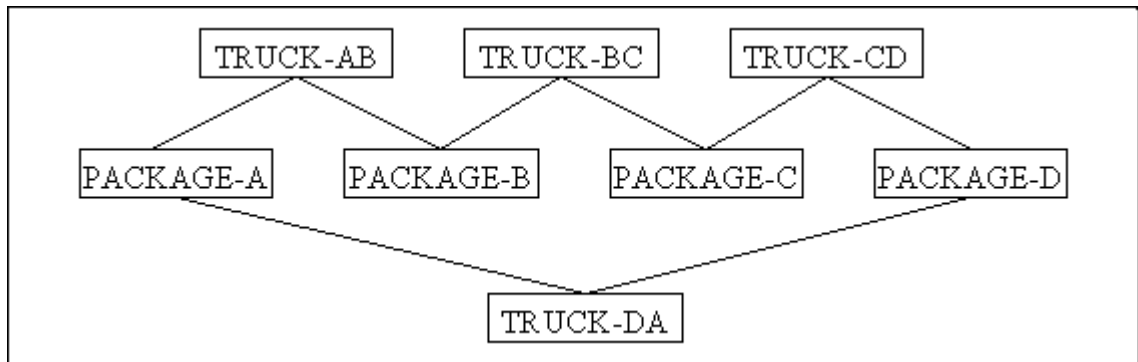


Figure 3.13: 4-Pack Pathological domain.

This 4-Pack domain is analogous to the 3-Pack Pathological domain, except that it contains one additional Truck subtype and one additional Package subtype and possesses a larger linear inequality set (Figure 3.14).

1.	$tda + tab \geq pa$
2.	$tab + tbc \geq pb$
3.	$tbc + tcd \geq pc$
4.	$tcd + tda \geq pd$
5.	$tda + tab + tbc \geq pa + pb$
6.	$tab + tbc + tcd \geq pb + pc$
7.	$tbc + tcd + tda \geq pc + pd$
8.	$tcd + tda + tab \geq pd + pa$
9.	$tab + tbc + tcd + tda \geq pa + pb + pc + pd$

Figure 3.14: Linear inequality set for the 4-Pack Pathological domain.

Note that the linear inequality set for the 4-Pack domain has approximately doubled in size from that of the 3-Pack domain. In the 3-Pack domain, the unions of the linear inequalities did not produce new equations, because equations already in the set subsumed the new equations. For example, in Figure 3.15, we discard a newly generated equation (top) because an equation already in the set subsumes it (bottom). However, this subsumption does not always occur in the 4-Pack domain: for example, equation 5 is the union of equations 1 and 2.

$tab + tbc \geq pb \cup tbc + tca \geq pc$ $\rightarrow tab + tbc + tca \geq pb + pc$
$tab + tbc + tca \geq pb + pc \subset$ $tab + tbc + tca \geq pa + pb + pc$

Figure 3.15: The new equation (top) is subsumed by an equation already in the set (bottom).

The 4-Pack domain suffers the same flaw as the 3-Pack domain, in that the planner cannot match any of the Truck objects without potentially violating one of the equations from the linear inequality set. This is apparent from the first four equations, though it is also the case with equations 5 through 8 as well. For example, the planner cannot decrement the variable tbc without potentially violating either equation 5 or equation 7, whose right hand sides are disjoint.

In addition to extending the 3-Pack domain to generate a new pathological domain, it is also the case that a pathological domain may contain multiple pathological dependency cycles, as in the 4-Pack-6-Truck Pathological domain (Figure 3.16) with its attendant linear inequality set (Figure 3.17). The 4-Pack-6-Truck Pathological domain contains multiple pathological dependency cycles because we would need to remove more than one link to convert the domain to a non-pathological domain, whereas we would only need to remove one link from the 3-Pack Pathological domain to convert it to a non-pathological domain.

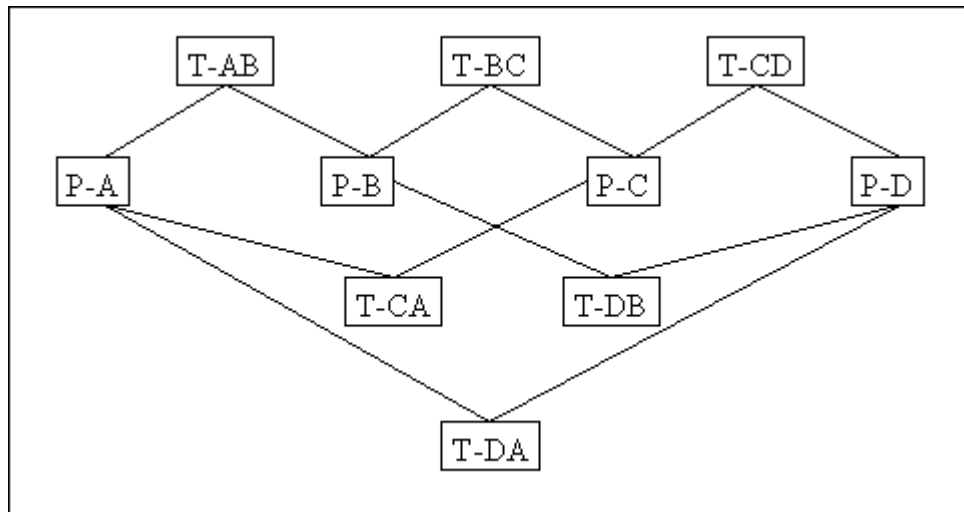


Figure 3.16: 4-Pack-6-Truck Pathological domain.

1. $t_{da} + t_{ca} + t_{ab} \geq p_a$
2. $t_{ab} + t_{bc} + t_{db} \geq p_b$
3. $t_{bc} + t_{cd} + t_{ca} \geq p_c$
4. $t_{cd} + t_{da} + t_{db} \geq p_d$
5. $t_{da} + t_{ca} + t_{ab} + t_{bc} + t_{db} \geq p_a + p_b$
6. $t_{da} + t_{ca} + t_{ab} + t_{bc} + t_{cd} \geq p_a + p_c$
7. $t_{ab} + t_{bc} + t_{db} + t_{ca} + t_{cd} \geq p_b + p_c$
8. $t_{ab} + t_{bc} + t_{db} + t_{cd} + t_{da} \geq p_b + p_d$
9. $t_{bc} + t_{cd} + t_{ca} + t_{da} + t_{db} \geq p_c + p_d$
10. $t_{cd} + t_{da} + t_{ca} + t_{ab} + t_{db} \geq p_d + p_a$
11. $t_{ab} + t_{bc} + t_{cd} + t_{da} + t_{ca} + t_{db} \geq p_a + p_b + p_c + p_d$

Figure 3.17: Linear inequality set for the 4-Pack-6-Truck Pathological domain.

This domain contains 4 Package subtypes and 6 Truck subtypes, and contains 3 pathological dependency cycles. One cyclic dependency is the domain's 4-Cycle, composed of the four Package subtypes and the Truck subtypes Truck-AB, Truck-BC, Truck-CD, and Truck-DA. In addition, there is a 3-Cycle that remains if we remove the Package-D subtype, and another 3-Cycle that remains if we remove the Package-A subtype instead. Note that if we remove any of the Package subtypes from the 4-Pack Pathological domain (Figure 3.13) that the dependency cycle has been broken and the resulting domain is non-pathological.

A pathological dependency cycle may also include non-binary graph edge sets, as in the 3-Ary 4-Pack domain (Figure 3.18) with its attendant linear inequality set (Figure 3.19).

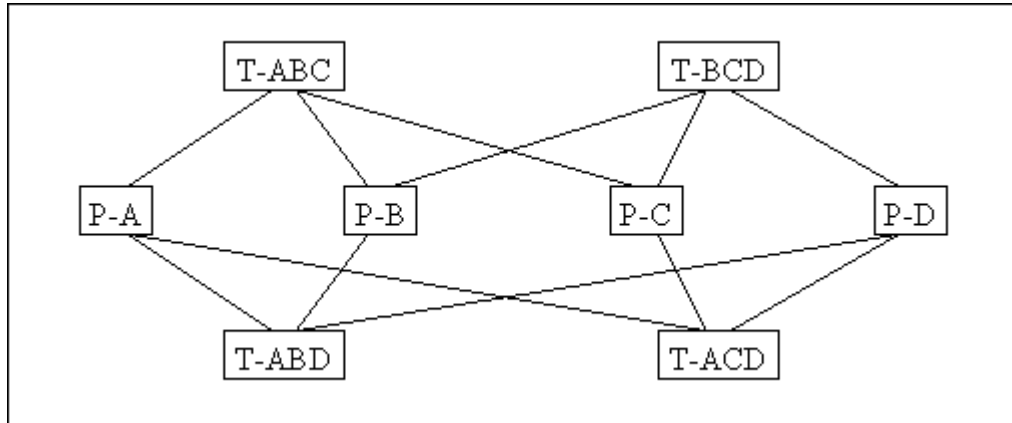


Figure 3.18: 3-Ary-4-Pack domain.

1. $tabc + tabd + tacd \geq pa$
2. $tabc + tbcd + tabd \geq pb$
3. $tabc + tacd + tbcd \geq pc$
4. $tabd + tacd + tbcd \geq pd$
5. $tabc + tbcd + tacd + tabd \geq pa + pb + pc + pd$

Figure 3.19: Linear inequality set for the 3-Ary-4-Pack domain.

This domain's nodes have an arity of 3 in that a Load/Unload operator for a Truck subtype will allow object bindings between objects of 3 different Package subtypes and the operator's Package variable, for example:

```
(Operator LOAD-TRUCK-ABC
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-A PACKAGE-B PACKAGE-C))
   (<trk> TRUCK-ABC))
```

The domain is still pathological, though one should note that the linear inequality set is smaller than that associated with the 4-Pack Pathological domain (Figure 3.13) because pre-existing equations subsume more new equations generated by Hall's Theorem than in the 2-Ary domain.

Unfortunately, Hall's Theorem only constitutes a necessary and sufficient condition for *bipartite* matching problems; the tripartite graph-matching problem and all 3+-partite graph-matching problems are NP-Complete (Garey and Johnson, 1979). However, Hall's Theorem still constitutes a *necessary* condition for tripartite graph matching, and thus provides some benefit to planning efficiency. For example, if we modify the logistics domain above such that we must also match Driver objects to Truck objects, with the restriction that certain Driver types are only able to transport certain Package types (e.g., for certain hazardous wastes), then we are left with a 3-Color 3-Pack Pathological domain (Figure 3.20).

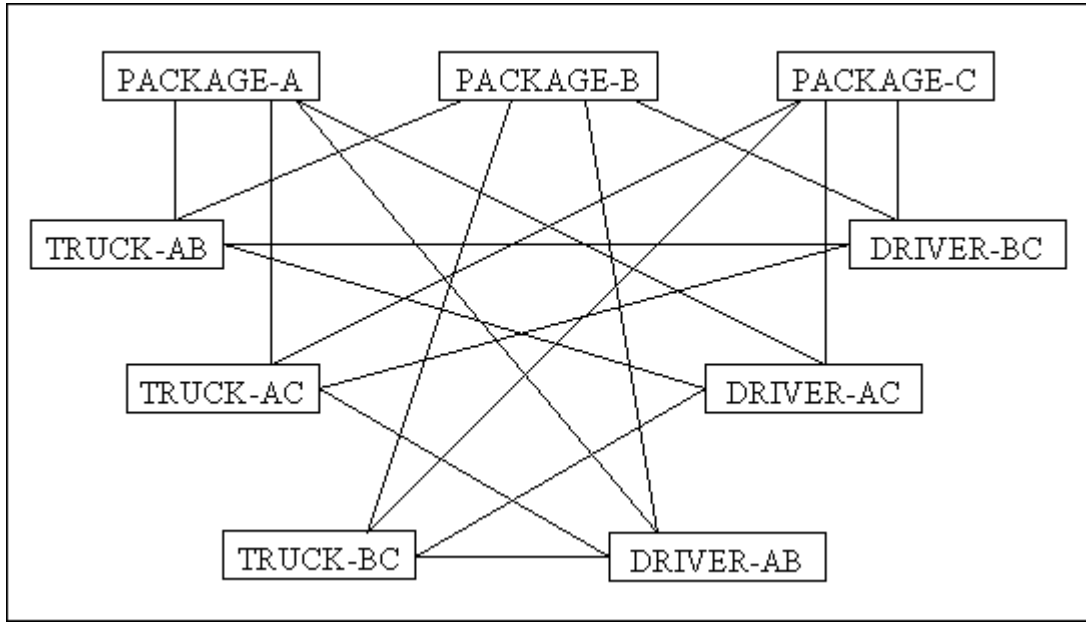


Figure 3.20: 3-Color-3-Pack Pathological domain.

This is essentially the union of three pathological domains: the familiar 3-Pack Pathological domain (Figure 3.1), the isomorphic 3-Pack Pathological domain with Driver subtypes replacing Truck subtypes (Figure 3.21), and the isomorphic 3-Pack Pathological domain with Driver subtypes replacing Package subtypes (Figure 3.22).

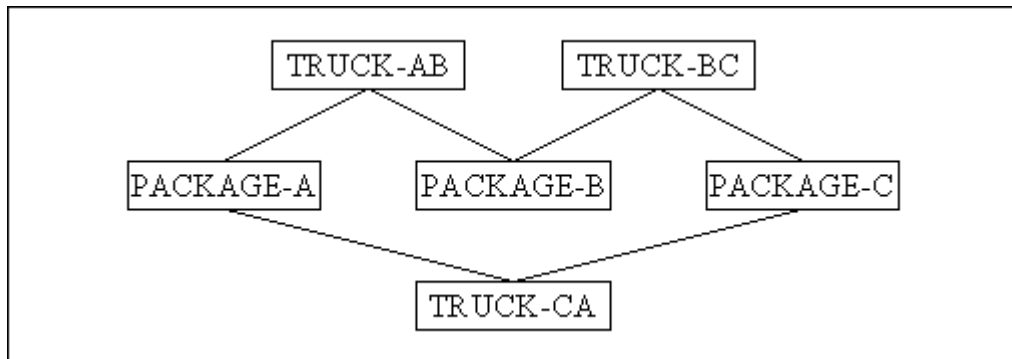


Figure 3.1: Pathological domain, isomorphic to Figure 1.3.

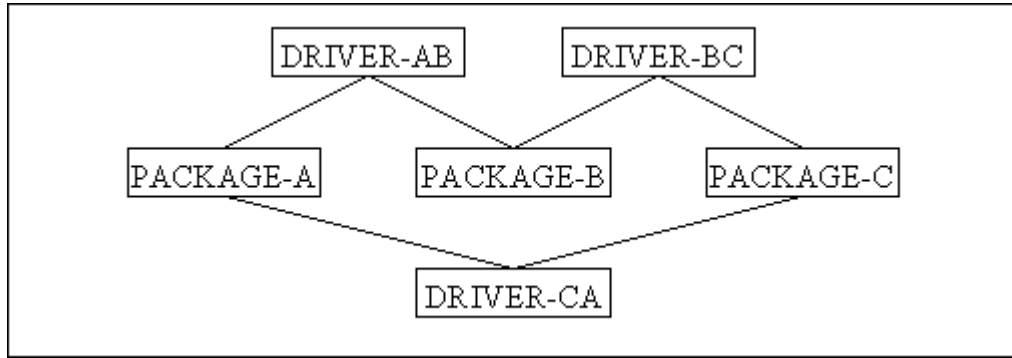


Figure 3.21: First isomorphic 3-Pack Pathological sub-domain.

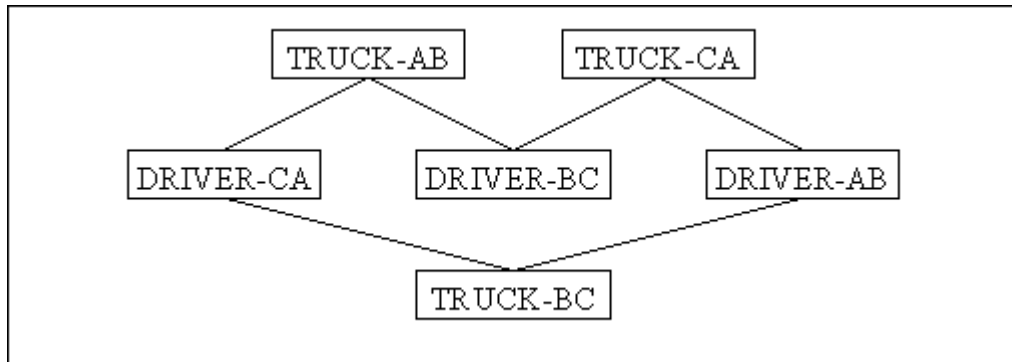


Figure 3.22: Second isomorphic 3-Pack Pathological sub-domain.

The linear inequality sets for these sub-domains are analogous to the linear inequality set for the original 3-Pack Pathological domain, and so we do not show them here. As per Hall's Theorem, these linear inequality sets constitute necessary and sufficient conditions for the bipartite matching problems in these sub-domains, and as such, they constitute necessary conditions for the tripartite matching problem in the 3-color union of these domains. Thus, these linear inequalities provide a powerful heuristic for finding a satisficing solution to the resulting NP-Complete problem.

These various sample domains make it clear that pathological domains present a severe problem in classical planning. They are not unique to small toy domains such as the 3-Pack Pathological domain, but rather they can appear in any arbitrarily complex form in a

vast number of problem domains. However, the fact that the planner cannot effectively solve even toy problems through FOL is troubling, in that few classical planners are capable of handling metric values (Smith, Frank, and Jónsson, 2000) as they would need to for this problem. This efficiency problem is compounded by the fact that a domain can be pathological despite being defined with FOL, as it is in PRODIGY.

We have already alluded to our solution to this problem of coding pathological domains with our new 3-Pack Pathological domain control rule in Figure 3.5. In the next section, we will give the Prodigy 4.0 domain encoding for three pathological domains: the 3-Pack Pathological domain (Figure 3.1), the 4-Pack Pathological domain (Figure 3.13), and the 5-Pack Pathological domain (Figure 3.23). We will then provide two different yet functionally equivalent control rule encodings that prevent the need to backtrack in these domains, and we will compare the resulting domain performance to the performance that results when no linear inequality control rules are used.

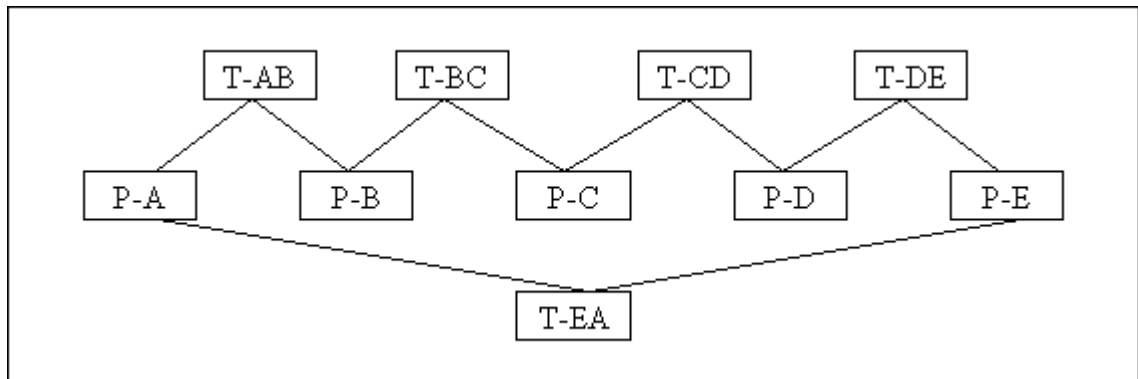


Figure 3.23: 5-Pack Pathological domain.

4 Linear Inequality Control Rules

It is clear from our domain analysis in the previous chapter that search controls coded in the STRIPS or ADL planning languages are not able to effectively solve matching problems in pathological domains. However, as we alluded to with control rule 5 (Figure 3.5), it *is* possible to solve pathological matching problems if we employ control rules with the capability of explicitly confirming the domain’s linear inequality set. We call these control rules *linear inequality control rules* (Cleereman and Cox, 2004a), and distinguish them from Prodigy 4.0’s standard FOL-based control rules.

4.1 Experimental Setup

We encoded four copies of three pathological PRODIGY domains: the 3-Pack Pathological domain (Figure 3.1), the 4-Pack Pathological domain (Figure 3.13), and the 5-Pack Pathological domain (Figure 3.23). The first copy, the Part-CR (Partial Control Rule) copy, was given a subset of the domain’s possible linear inequality control rules that were then supplemented with FOL control rules to ensure that the planner never needed to backtrack. The second copy, the Full-CR (Full Control Rule) copy, was given a full set of the domain’s possible linear inequality control rules to ensure that the planner never needed to backtrack. The third copy, the FOL-CR (First Order Logic Control Rule) copy, was given a full complement of FOL control rules without the added benefit of

linear inequality control rules as in the Part-CR domains, thus requiring the planner to backtrack in some domain problems. The fourth copy, the No-CR (No Control Rule) copy, was given zero control rules, thus requiring the planner to backtrack in some domain problems.

Figure 4.1, Figure 4.2, Figure 4.3, Figure 4.4, and Figure 4.5 give the Load-Truck operators for the 5-Pack Pathological domain, and Figure 4.6, Figure 4.7, Figure 4.8, Figure 4.9, and Figure 4.10 give the Unload-Truck operators for the 5-Pack Pathological domain. The Drive-Truck operator is identical to that used in the logistics domain in Figure 2.2. The 4-Pack domain differs from the 5-Pack domain in that the 4-Pack domain uses only four Load-Truck and four Unload-Truck operators, because it only contains four Truck subtypes and four Package subtypes. Likewise, the 3-Pack domain differs from the 4-Pack domain in that the 3-Pack domain uses only three Load-Truck and three Unload-Truck operators, because it only contains three Truck subtypes and three Package subtypes. We will only give the operators for the 5-Pack domain, as it is trivial to interpolate the operators for the 4-Pack and 3-Pack domains from the 5-Pack domain operators.

```

(Operator LOAD-TRUCK-AB
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-A PACKAGE-B))
   (<trk> TRUCK-AB)
   (<loc> LOCATION))
  (and
   (at-truck <trk> <loc>)
   (at-obj <obj> <loc>)
   (is-ready <trk>)))
 (effects ()
  (del (at-obj <obj> <loc>))
  (del (is-ready <trk>))
  (add (inside-truck <obj> <trk>))))

```

Figure 4.1: Load-Truck-AB operator for the 5-Pack Pathological domain.

```

(Operator LOAD-TRUCK-BC
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-B PACKAGE-C))
   (<trk> TRUCK-BC)
   (<loc> LOCATION))
  (and
   (at-truck <trk> <loc>)
   (at-obj <obj> <loc>)
   (is-ready <trk>)))
 (effects ()
  (del (at-obj <obj> <loc>))
  (del (is-ready <trk>))
  (add (inside-truck <obj> <trk>))))

```

Figure 4.2: Load-Truck-BC operator for the 5-Pack Pathological domain.

```

(Operator LOAD-TRUCK-CD
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-C PACKAGE-D))
   (<trk> TRUCK-CD)
   (<loc> LOCATION))
  (and
   (at-truck <trk> <loc>)
   (at-obj <obj> <loc>)
   (is-ready <trk>)))
 (effects ()
  (del (at-obj <obj> <loc>))
  (del (is-ready <trk>))
  (add (inside-truck <obj> <trk>))))

```

Figure 4.3: Load-Truck-CD operator for the 5-Pack Pathological domain.

```

(Operator LOAD-TRUCK-DE
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-D PACKAGE-E))
   (<trk> TRUCK-DE)
   (<loc> LOCATION))
  (and
   (at-truck <trk> <loc>)
   (at-obj <obj> <loc>)
   (is-ready <trk>)))
 (effects ()
  (del (at-obj <obj> <loc>))
  (del (is-ready <trk>))
  (add (inside-truck <obj> <trk>))))

```

Figure 4.4: Load-Truck-DE operator for the 5-Pack Pathological domain.

```

(Operator LOAD-TRUCK-EA
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-E PACKAGE-A))
   (<trk> TRUCK-EA)
   (<loc> LOCATION))
  (and
   (at-truck <trk> <loc>)
   (at-obj <obj> <loc>)
   (is-ready <trk>)))
 (effects ()
  (del (at-obj <obj> <loc>))
  (del (is-ready <trk>))
  (add (inside-truck <obj> <trk>))))

```

Figure 4.5: Load-Truck-EA operator for the 5-Pack Pathological domain.

```

(Operator UNLOAD-TRUCK-AB
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-A PACKAGE-B))
   (<trk> TRUCK-AB)
   (<loc> LOCATION))
  (and
   (inside-truck <obj> <trk>)
   (at-truck <trk> <loc>)))
 (effects ()
  (del (inside-truck <obj> <trk>))
  (add (at-obj <obj> <trk>))))

```

Figure 4.6: Unload-Truck-AB operator for the 5-Pack Pathological domain.

```

(Operator UNLOAD-TRUCK-BC
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-B PACKAGE-C))
   (<trk> TRUCK-BC)
   (<loc> LOCATION))
  (and
   (inside-truck <obj> <trk>)
   (at-truck <trk> <loc>)))
 (effects ()
  (del (inside-truck <obj> <trk>))
  (add (at-obj <obj> <trk>))))

```

Figure 4.7: Unload-Truck-BC operator for the 5-Pack Pathological domain.

```

(Operator UNLOAD-TRUCK-CD
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-C PACKAGE-D))
   (<trk> TRUCK-CD)
   (<loc> LOCATION))
  (and
   (inside-truck <obj> <trk>)
   (at-truck <trk> <loc>)))
 (effects ()
  (del (inside-truck <obj> <trk>))
  (add (at-obj <obj> <trk>))))

```

Figure 4.8: Unload-Truck-CD operator for the 5-Pack Pathological domain.

```

(Operator UNLOAD-TRUCK-DE
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-D PACKAGE-E))
   (<trk> TRUCK-DE)
   (<loc> LOCATION))
  (and
   (inside-truck <obj> <trk>)
   (at-truck <trk> <loc>)))
 (effects ()
  (del (inside-truck <obj> <trk>))
  (add (at-obj <obj> <trk>))))

```

Figure 4.9: Unload-Truck-DE operator for the 5-Pack Pathological domain.

```

(Operator UNLOAD-TRUCK-EA
 (<obj> <trk> <loc>)
 (precondition
  ((<obj> (or PACKAGE-E PACKAGE-A))
   (<trk> TRUCK-EA)
   (<loc> LOCATION))
  (and
   (inside-truck <obj> <trk>)
   (at-truck <trk> <loc>)))
 (effects ()
  (del (inside-truck <obj> <trk>))
  (add (at-obj <obj> <trk>))))

```

Figure 4.10: Unload-Truck-EA operator for the 5-Pack Pathological domain.

The Part-CR copy of the i-Pack domains employs a technique called *decoupling* to break the pathological dependency cycle within the domain, thereby converting a pathological domain to a non-pathological domain. We will decouple the i-Pack domains at the Truck-AB subtype (Figure 4.11, Figure 4.12, Figure 4.13), because this subtype is common to all three domain sets. In all three cases, we decouple the Truck-AB subtype into the Truck-AB1 and Truck-AB2 subtypes.

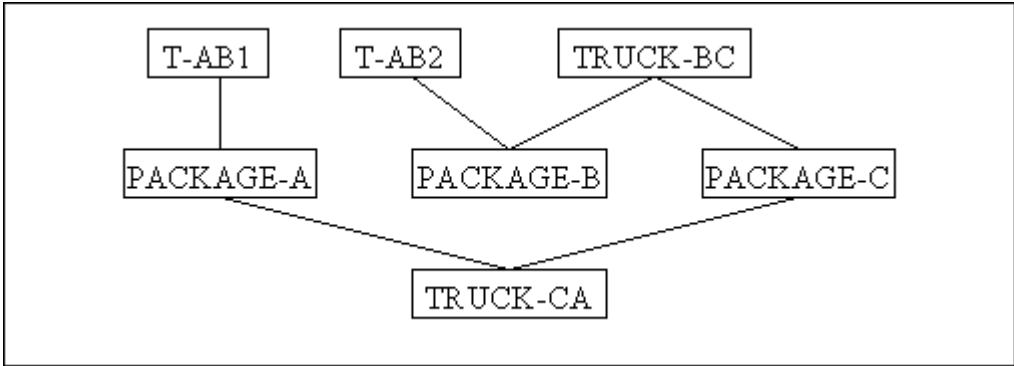


Figure 4.11: Decoupled 3-Pack Pathological domain.

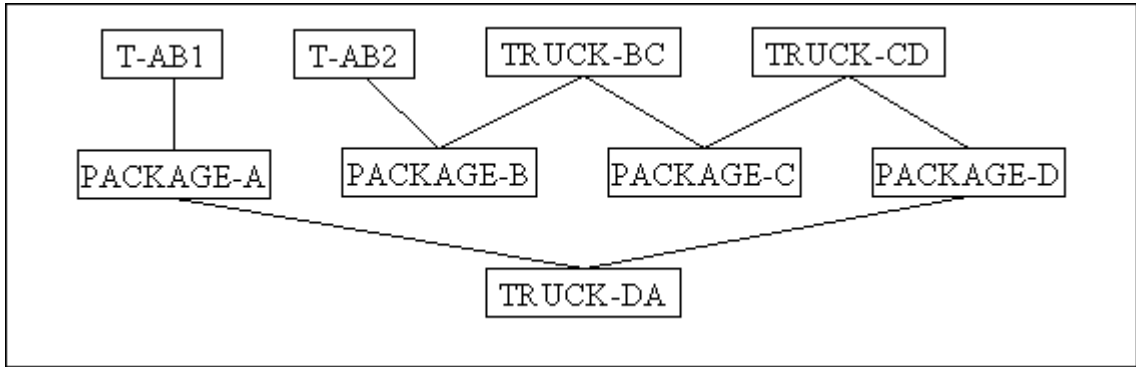


Figure 4.12: Decoupled 4-Pack Pathological domain.

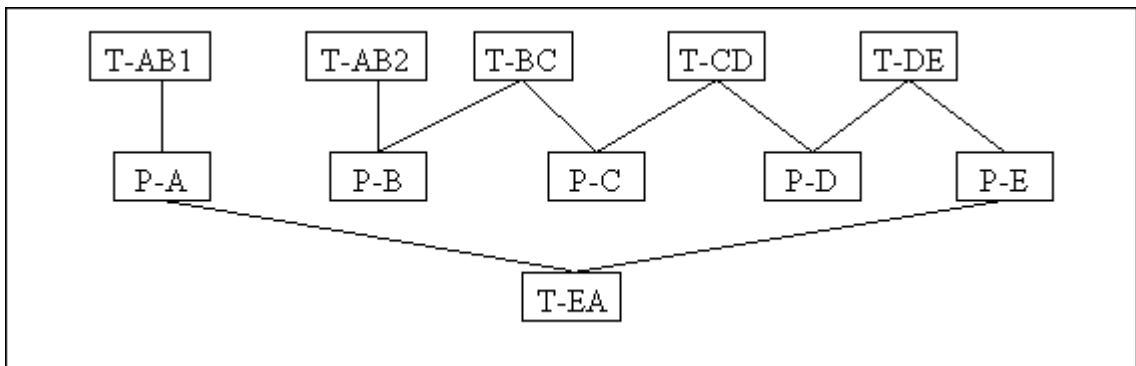


Figure 4.13: Decoupled 5-Pack Pathological domain.

In all three cases, the set of Truck-AB1 objects is disjoint with the set of Truck-AB2 objects, and the union of the set of Truck-AB1 objects with the set of Truck-AB2 objects is equivalent to the set of Truck-AB objects. Once the decoupling process has split the set of Truck-AB objects into its disjoint subsets such that the domain's linear inequalities hold, then the domain effectively becomes a non-pathological domain and FOL control rules can resume control over planning. By decoupling the domain, we reduce the domain's full linear inequality set (shown for the 5-Pack domain in Figure 4.14) to a much more manageable size (shown for the 5-Pack domain in Figure 4.15).

1. $tea + tab \geq pa$
2. $tab + tbc \geq pb$
3. $tbc + tcd \geq pc$
4. $tcd + tde \geq pd$
5. $tde + tea \geq pe$
6. $tea + tab + tbc \geq pa + pb$
7. $tab + tbc + tcd \geq pb + pc$
8. $tbc + tcd + tde \geq pc + pd$
9. $tcd + tde + tea \geq pd + pe$
10. $tde + tea + tab \geq pe + pa$
11. $tea + tab + tbc + tcd \geq pa + pb + pc$
12. $tab + tbc + tcd + tde \geq pb + pc + pd$
13. $tbc + tcd + tde + tea \geq pc + pd + pe$
14. $tcd + tde + tea + tab \geq pd + pe + pa$
15. $tde + tea + tab + tbc \geq pe + pa + pb$

Figure 4.14: Full linear inequality set for the 5-Pack Pathological domain.

1. $tab2 + tbc \geq pb$
2. $tab2 + tbc + tcd \geq pb + pc$
3. $tab2 + tbc + tcd + tde \geq pb + pc + pd$

Figure 4.15: Linear inequality set for the decoupled 5-Pack Pathological domain.

Decoupling of the 4-Pack and 3-Pack domains follows the same principle, and so we have not shown the decoupled linear inequalities for these domains. Note that the only

inequalities that remain in the decoupled 5-Pack domain are those that contain the *tab2* variable and do not contain the *tab1* variable.

The 5-Pack Part-CR domain applies precedence rules to the Package delivery goals so that Package-A objects take first priority (Figure 4.16), Package-B objects take second priority (Figure 4.17), Package-C objects take third priority (Figure 4.18), Package-D objects take fourth priority (Figure 4.19), and Package-E objects take fifth priority (Figure 4.20). The priorities remain identical for the 4-Pack Part-CR and 3-Pack Part-CR domains after removal of the absent Package subtypes from the queue.

```
(if (and
    (candidate-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A))
    (then select goal (at-obj <obja> <loc>)))
```

Figure 4.16: Package-A objects take first priority.

```
(if (and
    (candidate-goal (at-obj <objb> <loc>))
    (type-of-object <objb> PACKAGE-B)
    (~ (candidate-goal (at-obj <obja> <loc>)))
    (type-of-object <obja> PACKAGE-A))
    (then select goal (at-obj <objb> <loc>)))
```

Figure 4.17: Package-B objects take second priority.

```
(if (and
    (candidate-goal (at-obj <objc> <loc>))
    (type-of-object <objc> PACKAGE-C)
    (~ (candidate-goal (at-obj <objb> <loc>)))
    (type-of-object <objb> PACKAGE-B)
    (~ (candidate-goal (at-obj <obja> <loc>)))
    (type-of-object <obja> PACKAGE-A))
    (then select goal (at-obj <objc> <loc>)))
```

Figure 4.18: Package-C objects take third priority.

```
(if (and
    (candidate-goal (at-obj <objd> <loc>))
    (type-of-object <objd> PACKAGE-D)
    (~ (candidate-goal (at-obj <objc> <loc>)))
    (type-of-object <objc> PACKAGE-C)
    (~ (candidate-goal (at-obj <objb> <loc>)))
    (type-of-object <objb> PACKAGE-B)
    (~ (candidate-goal (at-obj <obja> <loc>)))
    (type-of-object <obja> PACKAGE-A))
    (then select goal (at-obj <objd> <loc>)))
```

Figure 4.19: Package-D objects take fourth priority.

```
(if (and
    (candidate-goal (at-obj <obje> <loc>))
    (type-of-object <obje> PACKAGE-E)
    (~ (candidate-goal (at-obj <objd> <loc>)))
    (type-of-object <objd> PACKAGE-D)
    (~ (candidate-goal (at-obj <objc> <loc>)))
    (type-of-object <objc> PACKAGE-C)
    (~ (candidate-goal (at-obj <objb> <loc>)))
    (type-of-object <objb> PACKAGE-B)
    (~ (candidate-goal (at-obj <obja> <loc>)))
    (type-of-object <obja> PACKAGE-A))
    (then select goal (at-obj <obje> <loc>)))
```

Figure 4.20: Package-E objects take fifth priority.

The domain is then decoupled by matching Truck-AB1 objects to Package-A objects if the decoupled inequality set would not be violated by such a match (Figure 4.21), else matching the Package-A object to a Truck-EA object instead (Figure 4.22).

```
(if (and
    (current-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A)
    (greater-than (TRUCK-AB TRUCK-BC)
                  (PACKAGE-B))
    (greater-than (TRUCK-AB TRUCK-BC TRUCK-CD)
                  (PACKAGE-B PACKAGE-C))
    (greater-than (TRUCK-AB TRUCK-BC
                  TRUCK-CD TRUCK-DE)
                  PACKAGE-B PACKAGE-C PACKAGE-D)))
    (then select operator UNLOAD-TRUCK-AB))
```

Figure 4.21: Package-A objects are first partitioned into the Package-AB1 subtype.

```
(if (and
    (current-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A)
    (or
     (~ (greater-than (TRUCK-AB TRUCK-BC)
                      (PACKAGE-B)))
     (~ (greater-than (TRUCK-AB TRUCK-BC TRUCK-CD)
                      (PACKAGE-B PACKAGE-C)))
     (~ (greater-than (TRUCK-AB TRUCK-BC
                      TRUCK-CD TRUCK-DE)
                      (PACKAGE-B PACKAGE-C
                      PACKAGE-D))))))
    (then select operator UNLOAD-TRUCK-EA))
```

Figure 4.22: Package-A objects are then partitioned into the Package-AB2 subtype.

Recall from control rule 5 in Figure 3.5 that the greater-than function is passed two lists of object subtypes and returns true if there are more objects of the first set of subtypes than there are objects of the second set of subtypes, else returns false. For example, if there are two unmatched Truck-AB objects, two unmatched Truck-BC objects, and three unmatched Package-B objects, then the linear inequality $tab + tbc \geq pb$ will hold true

even if the number of Truck-AB objects is reduced by 1 (i.e., $tab-- + tbc \geq pb$ will hold true). However, if there are two unmatched Truck-AB objects, two unmatched Truck-BC objects, and four unmatched Package-B objects, then the linear inequality $tab + tbc \geq pb$ will be violated if the number of Truck-AB objects is reduced by 1, and thus it is not possible to match any more Truck-AB objects to Package-A objects.

Once the planner applies the goal precedence rules and the linear inequality control rules then it has decoupled the domain. FOL control rules then apply operator/binding precedence, because the planner has effectively split the Truck-AB subtype into the Truck-AB1 subset (all objects of which have already been matched to Package-A objects) leaving only the Truck-AB2 subset. Truck-AB matches take first priority (Figure 4.23), Truck-BC matches take second priority (Figure 4.23, Figure 4.24), Truck-CD matches take third priority (Figure 4.24, Figure 4.25), Truck-DE matches take fourth priority (Figure 4.25, Figure 4.26), and Truck-EA matches take fifth priority (Figure 4.26).

```
(if (and
    (current-goal (at-obj <objb> <loc>))
    (type-of-object <objb> PACKAGE-B))
    (then prefer operator UNLOAD-TRUCK-AB UNLOAD-TRUCK-BC))
```

Figure 4.23: Truck-AB matches take precedence over Truck-BC matches.

```
(if (and
    (current-goal (at-obj <objc> <loc>))
    (type-of-object <objc> PACKAGE-C))
    (then prefer operator UNLOAD-TRUCK-BC UNLOAD-TRUCK-CD))
```

Figure 4.24: Truck-BC matches take precedence over Truck-CD matches.

```
(if (and
    (current-goal (at-obj <objd> <loc>))
    (type-of-object <objd> PACKAGE-D))
    (then prefer operator UNLOAD-TRUCK-CD UNLOAD-TRUCK-DE))
```

Figure 4.25: Truck-CD matches take precedence over Truck-DE matches.

```
(if (and
    (current-goal (at-obj <obje> <loc>))
    (type-of-object <obje> PACKAGE-E))
    (then prefer operator UNLOAD-TRUCK-DE UNLOAD-TRUCK-EA))
```

Figure 4.26: Truck-DE matches take precedence over Truck-EA matches.

PRODIGY can now efficiently find a solution to any problem in this domain if a solution exists. The x2 problem whose initial state we give in Figure 4.27 will exemplify the use of the linear inequality and FOL control rules. (In general, an xN problem is a problem for a given domain with N Packages of each subtype and an equal number of Trucks and Packages for which an exact solution exists.)

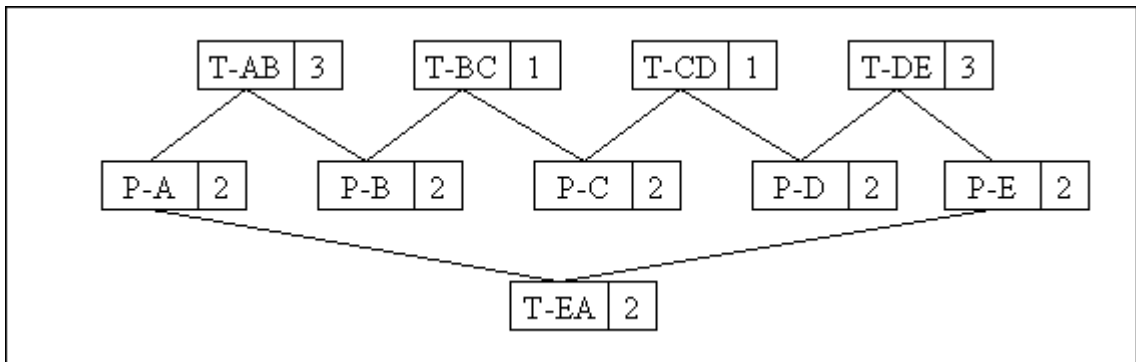


Figure 4.27: Initial state of an x2 problem in the 5-Pack Pathological domain.

As with all xN problems, the domain's linear inequality set is satisfied in the problem's initial state. Initially, the planner matches the two Package-A objects to the appropriate Truck objects (Figure 4.28).

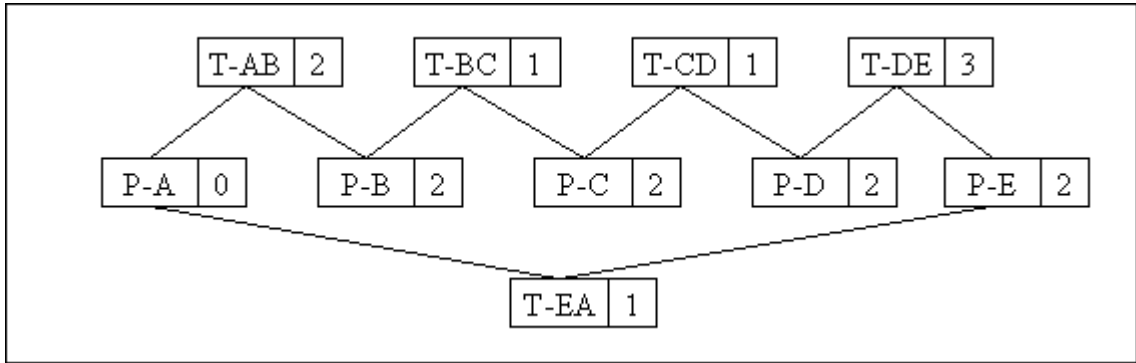


Figure 4.28: The planner has matched all Package-A objects.

The planner initially selects the Unload-Truck-AB operator to move the first Package-A object, because the three Greater-Than functions are all satisfied in the first linear inequality control rule (Figure 4.21). The Unload-Truck-AB operator selection decrements the value of *tab*, for the Unload-Truck-AB operator always takes a Truck-AB binding for the <trk> variable. The planner then selects the Unload-Truck-EA operator to move the remaining Package-A object, for not all of the Greater-Than functions are satisfied in the second linear inequality control rule (Figure 4.22). Specifically, $tab + tbc + tcd == pb + pc$, therefore $tab-- + tbc + tcd < pb + pc$ and the problem will have no solution. The Unload-Truck-EA operator selection decrements the value of *tea* when a Truck-EA object is bound to the operator's <trk> variable.

The planner then matches all Package-B objects (Figure 4.29).

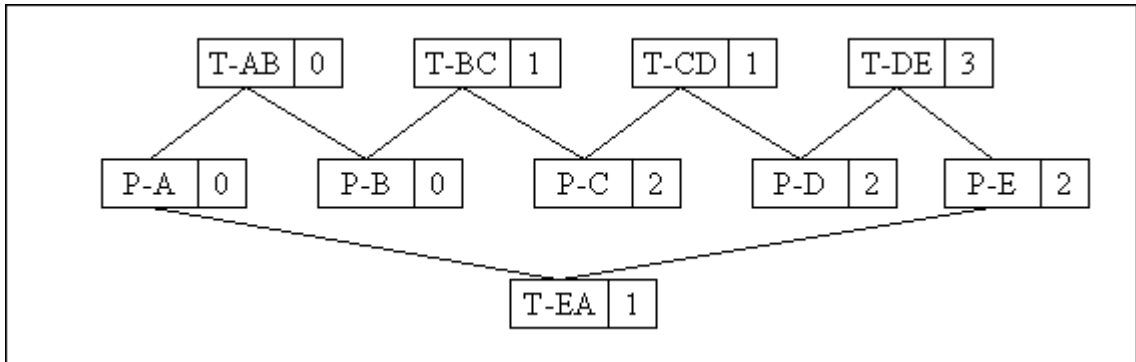


Figure 4.29: The planner has matched all Package-B objects.

The Unload-Truck-AB operator takes precedence over the Unload-Truck-BC operator, so the planner matches both Package-A objects to Truck-AB objects.

The planner then matches all Package-C objects (Figure 4.30).

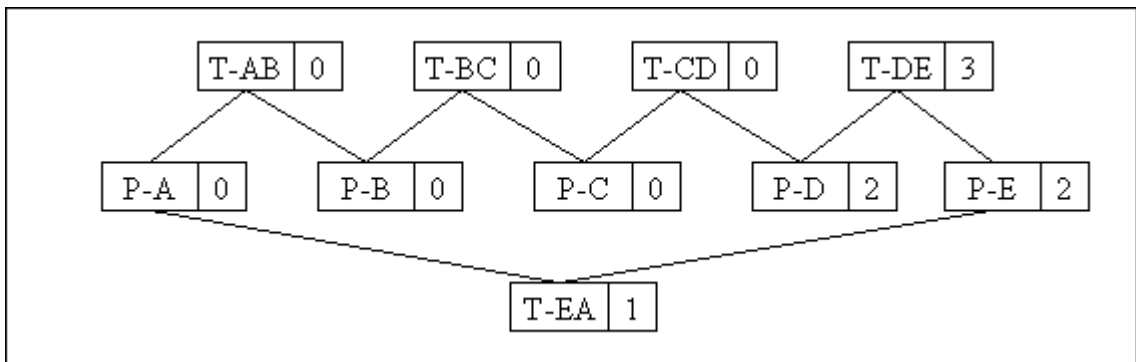


Figure 4.30: The planner has matched all Package-C objects.

There are only two Truck objects capable of moving the two Package-C objects, so operator precedence is not a factor.

The planner then matches all Package-D objects (Figure 4.31).

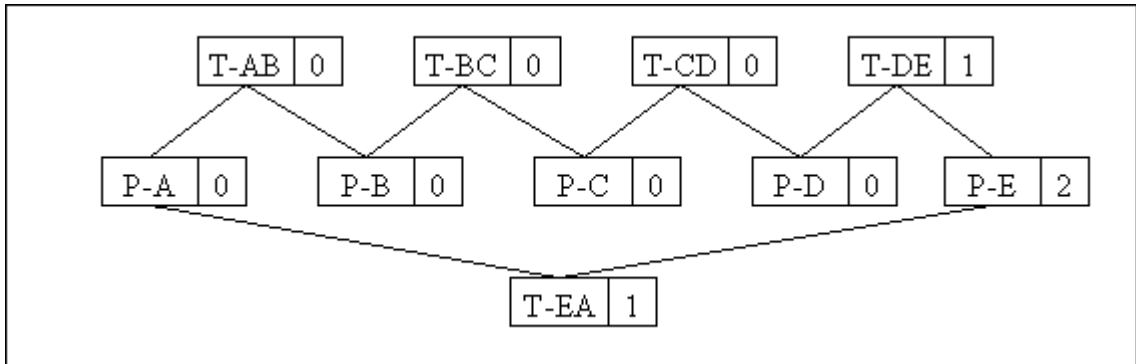


Figure 4.31: The planner has matched all Package-D objects.

The Unload-Truck-CD operator takes precedence over the Unload-Truck-DE operator, but the planner will automatically select the Unload-Truck-DE operator when it finds that there are no remaining Truck-CD bindings.

Finally, the planner matches the Package-E objects (Figure 4.32).

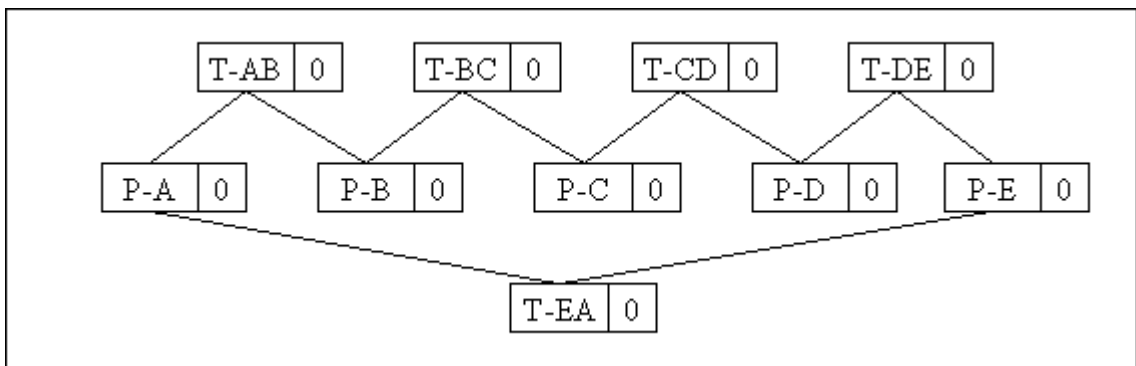


Figure 4.32: The planner has successfully matched all Package objects.

The planner has successfully matched all Package objects, and the planning session ends.

The i-Pack domains contain only one pathological dependency cycle, and so it is relatively simple to decouple the cycles and convert the pathological domains to non-pathological domains. However, decoupling is more difficult to apply to domains with

multiple pathological dependency cycles (Figure 3.16) or with greater arities in their operators (Figure 3.18), for it may be necessary to employ multiple decoupling points. In addition, the domain's precedence queue may not allow for an easy FOL search control encoding (e.g., the natural precedence of moving Package-C objects may be higher than that of moving Package-A objects but lower than that of moving Package-B objects), or it may be inappropriate to encode any sort of precedence queue of the domain's goals. To demonstrate a method for combating these potential problems, we encoded a set of Full-CR domains that avoid the need to use any FOL control rules and instead rely entirely on linear inequality control rules.

The Full-CR domain encodes the i-Pack domains' full sets of linear inequalities (Figure 3.2, Figure 3.14, and Figure 4.14) as operator reject rules (Figure 4.33, Figure 4.34, Figure 4.35, Figure 4.36, Figure 4.37, Figure 4.38, Figure 4.39, Figure 4.40, Figure 4.41, Figure 4.42).

```
(if (and
  (current-goal (at-obj <obja> <loc>))
  (type-of-object <obja> PACKAGE-A)
  (or
    (~ (greater-than (TRUCK-AB TRUCK-BC)
                     (PACKAGE-B)))
    (~ (greater-than (TRUCK-AB TRUCK-BC TRUCK-CD)
                     (PACKAGE-B PACKAGE-C)))
    (~ (greater-than (TRUCK-AB TRUCK-BC
                     TRUCK-CD TRUCK-DE)
                     (PACKAGE-B PACKAGE-C
                     PACKAGE-D))))))
  (then reject operator UNLOAD-TRUCK-AB))
```

Figure 4.33: Reject invalid Truck-AB – Package-A matches.

```

(if (and
  (current-goal (at-obj <objb> <loc>))
  (type-of-object <objb> PACKAGE-B)
  (or
    (~ (greater-than (TRUCK-EA TRUCK-AB)
                     (PACKAGE-A)))
    (~ (greater-than (TRUCK-DE TRUCK-EA TRUCK-AB)
                     (PACKAGE-E PACKAGE-A)))
    (~ (greater-than (TRUCK-CD TRUCK-DE
                     TRUCK-EA TRUCK-AB)
                     (PACKAGE-D PACKAGE-E
                     PACKAGE-A))))))
  (then reject operator UNLOAD-TRUCK-AB))

```

Figure 4.34: Reject invalid Truck-AB – Package-B matches.

```

(if (and
  (current-goal (at-obj <objb> <loc>))
  (type-of-object <objb> PACKAGE-B)
  (or
    (~ (greater-than (TRUCK-BC TRUCK-CD)
                     (PACKAGE-C)))
    (~ (greater-than (TRUCK-BC TRUCK-CD TRUCK-DE)
                     (PACKAGE-C PACKAGE-D)))
    (~ (greater-than (TRUCK-BC TRUCK-CD
                     TRUCK-DE TRUCK-EA)
                     (PACKAGE-C PACKAGE-D
                     PACKAGE-E))))))
  (then reject operator UNLOAD-TRUCK-BC))

```

Figure 4.35: Reject invalid Truck-BC – Package-B matches.

```

(if (and
    (current-goal (at-obj <objc> <loc>))
    (type-of-object <objc> PACKAGE-C)
    (or
     (~ (greater-than (TRUCK-AB TRUCK-BC)
                      (PACKAGE-B)))
      (~ (greater-than (TRUCK-EA TRUCK-AB TRUCK-BC)
                      (PACKAGE-A PACKAGE-B)))
      (~ (greater-than (TRUCK-DE TRUCK-EA
                      TRUCK-AB TRUCK-BC)
                      (PACKAGE-E PACKAGE-A
                      PACKAGE-B))))))
    (then reject operator UNLOAD-TRUCK-BC))

```

Figure 4.36: Reject invalid Truck-BC – Package-C matches.

```

(if (and
    (current-goal (at-obj <objc> <loc>))
    (type-of-object <objc> PACKAGE-C)
    (or
     (~ (greater-than (TRUCK-CD TRUCK-DE)
                      (PACKAGE-D)))
      (~ (greater-than (TRUCK-CD TRUCK-DE TRUCK-EA)
                      (PACKAGE-D PACKAGE-E)))
      (~ (greater-than (TRUCK-CD TRUCK-DE
                      TRUCK-EA TRUCK-AB)
                      (PACKAGE-D PACKAGE-E
                      PACKAGE-A))))))
    (then reject operator UNLOAD-TRUCK-CD))

```

Figure 4.37: Reject invalid Truck-CD – Package-C matches.

```

(if (and
    (current-goal (at-obj <objd> <loc>))
    (type-of-object <objd> PACKAGE-D)
    (or
     (~ (greater-than (TRUCK-BC TRUCK-CD)
                      (PACKAGE-C)))
      (~ (greater-than (TRUCK-AB TRUCK-BC TRUCK-CD)
                      (PACKAGE-B PACKAGE-C)))
      (~ (greater-than (TRUCK-EA TRUCK-AB
                      TRUCK-BC TRUCK-CD)
                      (PACKAGE-A PACKAGE-B
                      PACKAGE-C))))))
    (then reject operator UNLOAD-TRUCK-CD))

```

Figure 4.38: Reject invalid Truck-CD – Package-D matches.

```

(if (and
    (current-goal (at-obj <objd> <loc>))
    (type-of-object <objd> PACKAGE-D)
    (or
     (~ (greater-than (TRUCK-DE TRUCK-EA)
                      (PACKAGE-E)))
      (~ (greater-than (TRUCK-DE TRUCK-EA TRUCK-AB)
                      (PACKAGE-E PACKAGE-A)))
      (~ (greater-than (TRUCK-DE TRUCK-EA
                      TRUCK-AB TRUCK-BC)
                      (PACKAGE-E PACKAGE-A
                      PACKAGE-B))))))
    (then reject operator UNLOAD-TRUCK-DE))

```

Figure 4.39: Reject invalid Truck-DE – Package-D matches.

```

(if (and
    (current-goal (at-obj <obje> <loc>))
    (type-of-object <obje> PACKAGE-E)
    (or
     (~ (greater-than (TRUCK-CD TRUCK-DE)
                      (PACKAGE-D)))
      (~ (greater-than (TRUCK-BC TRUCK-CD TRUCK-DE)
                      (PACKAGE-C PACKAGE-D)))
      (~ (greater-than (TRUCK-AB TRUCK-BC
                      TRUCK-CD TRUCK-DE)
                      (PACKAGE-B PACKAGE-C
                      PACKAGE-D))))))
    (then reject operator UNLOAD-TRUCK-DE))

```

Figure 4.40: Reject invalid Truck-DE – Package-E matches.

```

(if (and
    (current-goal (at-obj <obje> <loc>))
    (type-of-object <obje> PACKAGE-E)
    (or
     (~ (greater-than (TRUCK-EA TRUCK-AB)
                      (PACKAGE-A)))
      (~ (greater-than (TRUCK-EA TRUCK-AB TRUCK-BC)
                      (PACKAGE-A PACKAGE-B)))
      (~ (greater-than (TRUCK-EA TRUCK-AB
                      TRUCK-BC TRUCK-CD)
                      (PACKAGE-A PACKAGE-B
                      PACKAGE-C))))))
    (then reject operator UNLOAD-TRUCK-EA))

```

Figure 4.41: Reject invalid Truck-EA – Package-E matches.

```

(if (and
    (current-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A)
    (or
     (~ (greater-than (TRUCK-DE TRUCK-EA)
                      (PACKAGE-E)))
     (~ (greater-than (TRUCK-CD TRUCK-DE TRUCK-EA)
                      (PACKAGE-D PACKAGE-E)))
     (~ (greater-than (TRUCK-BC TRUCK-CD
                      TRUCK-DE TRUCK-EA)
                      (PACKAGE-C PACKAGE-D
                      PACKAGE-E))))))
    (then reject operator UNLOAD-TRUCK-EA))

```

Figure 4.42: Reject invalid Truck-EA – Package-A matches.

This set of control rules amounts to one control rule per graph edge, where a graph edge is an allowable connection between object subtypes. For example, a Truck-AB – Package-A connection is allowable, and so the resulting edge requires a control rule (Figure 4.33). The domains’ operators disallow a Truck-AB – Package-C connection, and so there is no resulting graph edge and no need for a control rule.

The strategy of only employing linear inequality reject control rules is extremely useful in situations where it is difficult or inappropriate to impose an inherent priority on goal/binding selections. For example, in the Full-CR 5-Pack domain above it would be relatively simple for a user to insist that all Package-C objects be moved first, whereas this would not be possible in the Part-CR 5-Pack domain without completely reworking the domain’s control rules.

The primary disadvantages of the Full-CR domain are that the planner will not be able to find partial solutions to an unsolvable problem without using a goal transformation (Cox

and Veloso, 1998), and that linear inequality control rules are more complicated and therefore less intuitive than FOL control rules. The first disadvantage is actually a mixed blessing, for the planner will be able to immediately recognize an unsolvable problem and reject all possible operators in a Full-CR domain, bringing about an immediate halt to planning. In contrast, a domain using FOL control rules would need to search through the entire allowable search space before being able to determine that the plan is unsolvable, and this search process is prohibitively expensive. On the other hand, the planner will be able to return the best available partial solution if it uses FOL search controls, but it will have no solution data if it uses a Full-CR set of control rules. We can do little about the second disadvantage associated with linear inequality control rules.

In terms of expected performance, there is no difference between the Part-CR domain and the Full-CR domain. Both domains forego the need to backtrack, and so both domains should expand an identical number of search nodes.

The third domain copy, the FOL-CR domain, uses many of the same control rules as the Part-CR domain. The difference is that the FOL-CR domain does not employ linear inequality control rules to decouple the domain.

Thus, in addition to the goal-ordering control rules (Figure 4.16, Figure 4.17, Figure 4.18, Figure 4.19, Figure 4.20) and operator precedence control rules (Figure 4.23, Figure 4.24, Figure 4.25, Figure 4.26) used in the Part-CR domain, the FOL-CR domain uses one

additional operator precedence control rule (Figure 4.43) to replace the linear inequality control rules used in the Part-CR domain (Figure 4.21, Figure 4.22).

```
(if (and
    (current-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A))
    (then prefer operator UNLOAD-TRUCK-AB UNLOAD-TRUCK-EA))
```

Figure 4.43: FOL-CR operator precedence control rule.

This set of control rules allows the FOL-CR domain to find an efficient solution to *some* problems, but not *all* problems. If the problem is one such that the operator precedence control rule in Figure 4.43 will not violate the decoupled linear inequalities (Problem A, Figure 4.44, Figure 4.45), then the FOL-CR domain will exhibit performance that is essentially equivalent to that of the Part-CR and Full-CR domains. However, if the control rule in Figure 4.43 will lead to a violation of the domain's decoupled linear inequalities (Problem B, Figure 4.46, Figure 4.47), then the planner will need to employ costly backtracking.

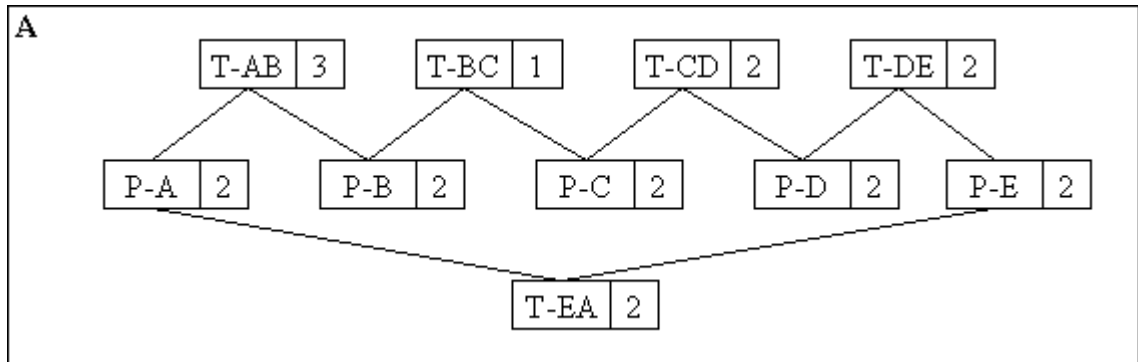


Figure 4.44: Initial state of sample problem A.

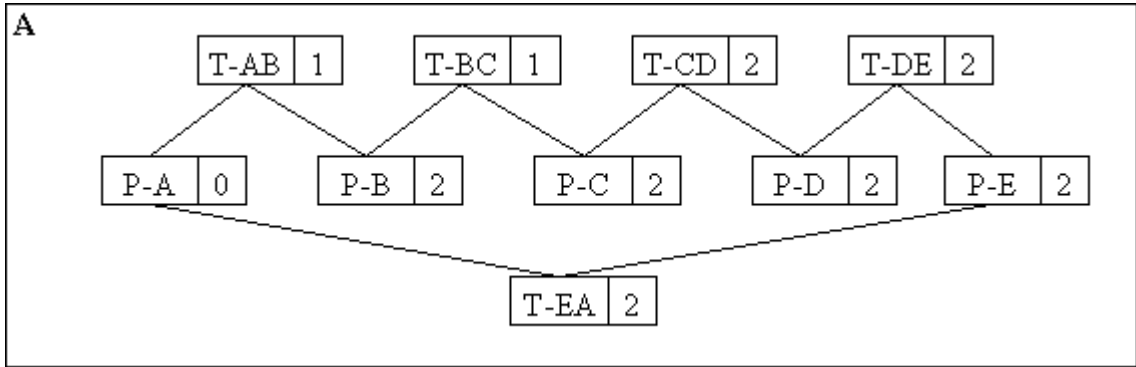


Figure 4.45: The FOL-CR search controls have maintained the linear inequality set.

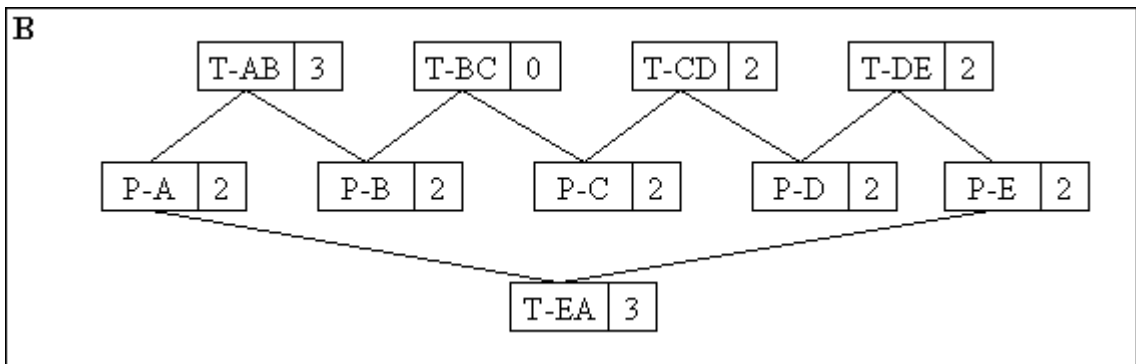


Figure 4.46: Initial state of sample problem B.

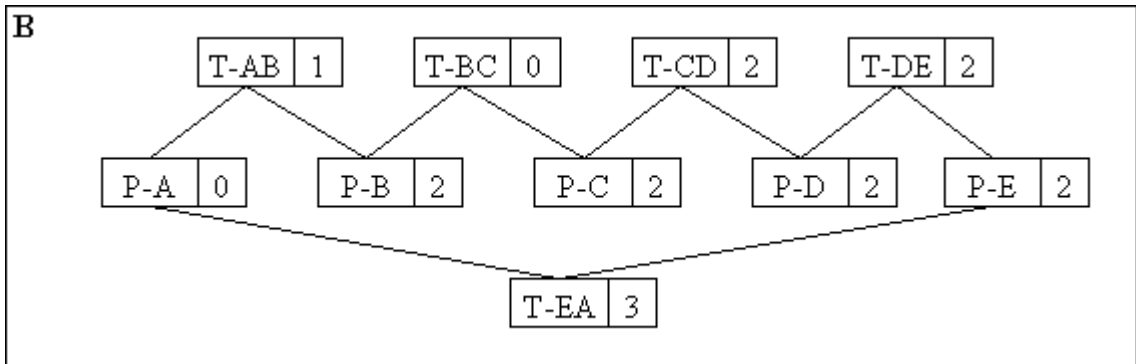


Figure 4.47: The FOL-CR search controls have violated the linear inequality set.

In contrast to the Part-CR, Full-CR, and FOL-CR domains, the No-CR domain uses no search controls whatsoever, and thus we expect it to run in exponential time.

4.2 Experimental Results

We ran experimental tests on all four domain copies for the 3-Pack, 4-Pack, and 5-Pack domains, using one hundred randomly generated x_1 , x_2 , x_3 , and x_4 problems for a total of four hundred problems per domain. We randomly generated problems for every copy of every domain, with the restrictions that every x_N problem would contain N Package objects of each subtype, every problem would contain an equal number of Truck and Package objects, and every problem would possess a solution. The FOL-CR and No-CR tests did not always complete within a reasonable amount of time, and so every problem had a 20,000 maximum node limit set. That is to say, once the planner expanded 20,000 nodes, the search would terminate, and the problem would have “no solution” for purposes of our tests. The Part-CR and Full-CR domains solved all problems within the node limit.

Figure 4.48, Figure 4.49, Figure 4.50, and Figure 4.51 demonstrate the relation between problem/domain complexity and planning complexity. One expects that the number of nodes expanded by the planner will increase as the number of objects within the problem increases, and the data confirm this hypothesis. In addition, one would expect that the number of nodes expanded by the planner will increase as the number of subtypes in the domain increases, and again the data confirm this hypothesis. Note that Figure 4.50 and Figure 4.51 only include data from those problems solved within the node limit, and so they approximately exhibit the same linear trend as Figure 4.48 and Figure 4.49.

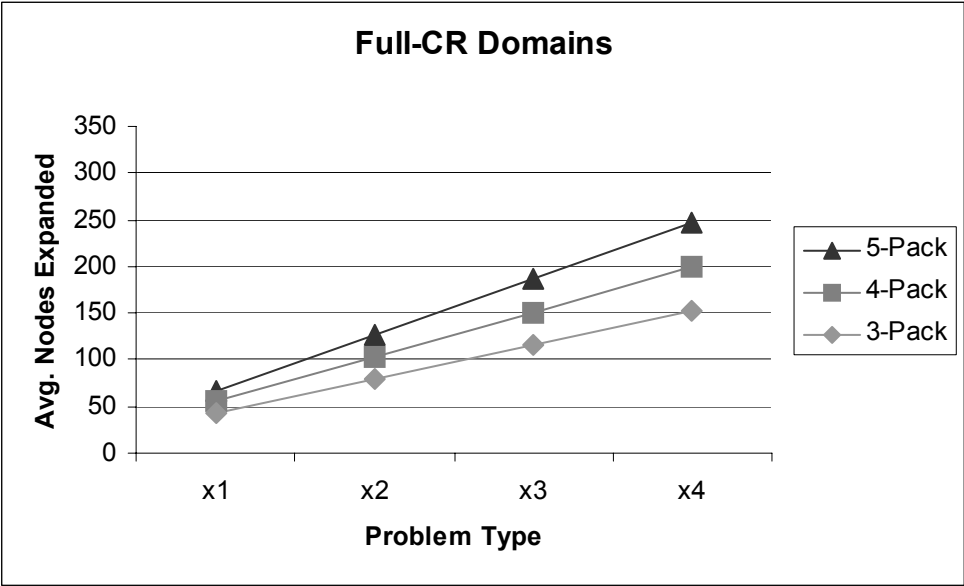


Figure 4.48: Experimental results for the Full-CR domain set.

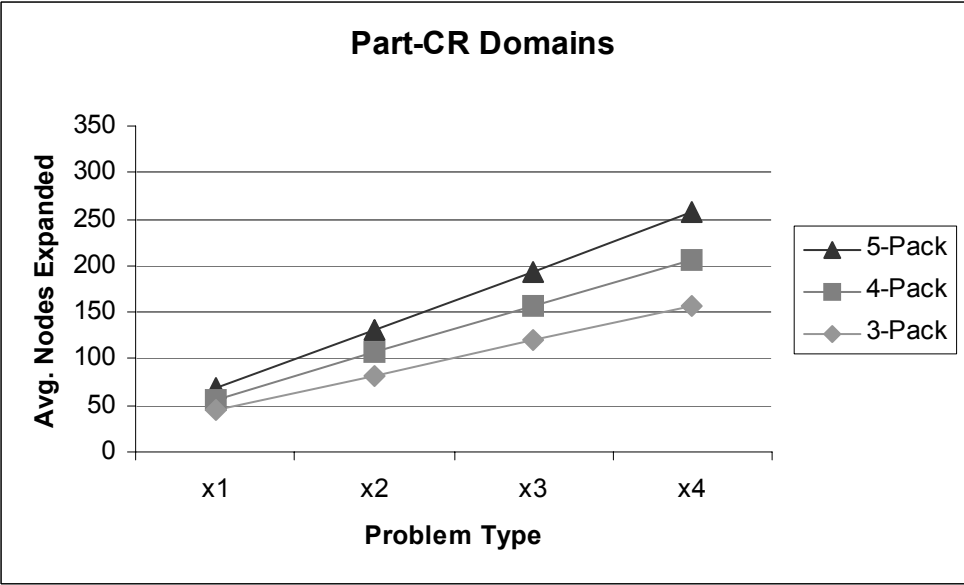


Figure 4.49: Experimental results for the Part-CR domain set.

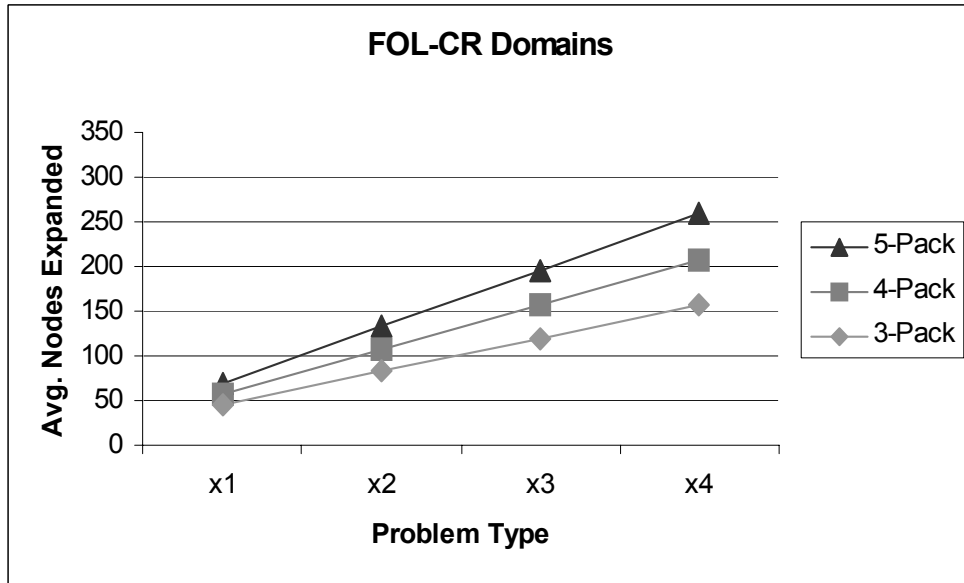


Figure 4.50: Experimental results for the FOL-CR domain set, tracking only those problems solved within the node limit.

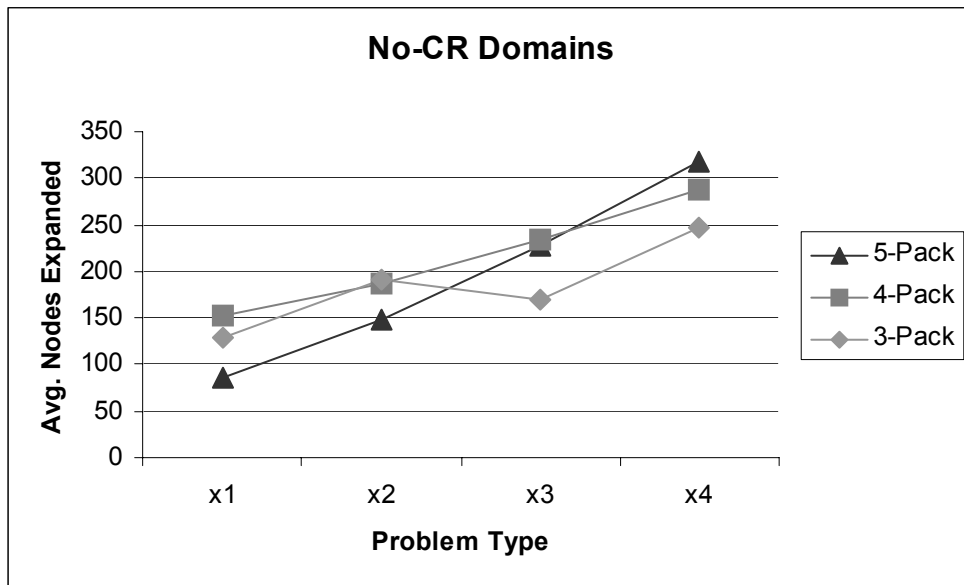


Figure 4.51: Experimental results for the No-CR domain set, tracking only those problems solved within the node limit.

The Full-CR, Part-CR, and FOL-CR domains display the expected trend of increased planning complexity for increased problem/domain complexity, but the No-CR domains do not. This is the result of our only averaging problems that were solved within the 20,000 node limit and ignoring all other problems. As shown in Figure 4.52 and Figure

4.53, this introduces a significant experimental error in the No-CR domains. (For example, the 5-Pack No-CR domain only solved 8% of its x4 problems within the 20,000-node limit). The Full-CR and Part-CR domains solved all problems within the node limit, and so their data are not shown.

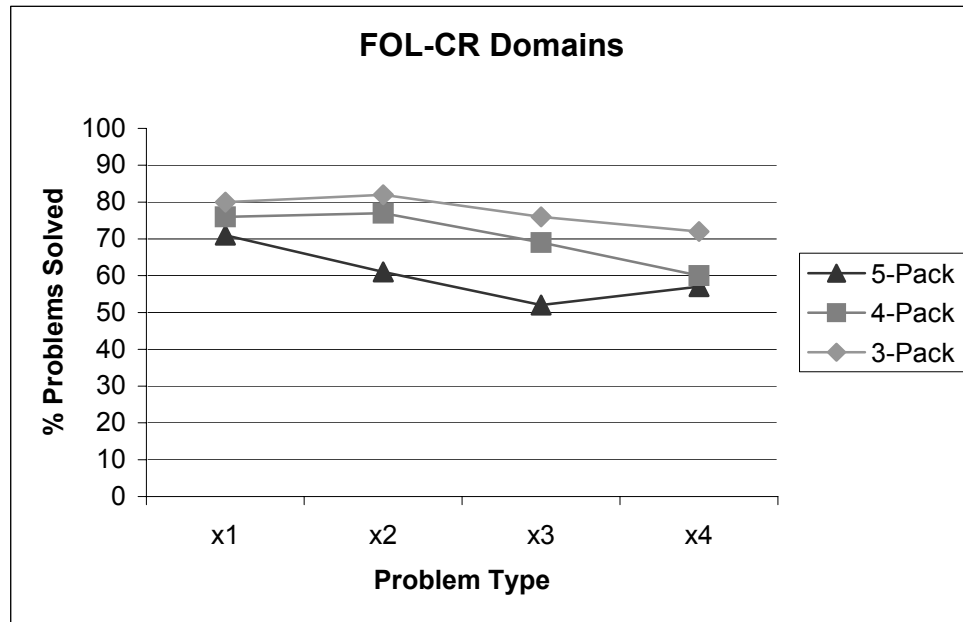


Figure 4.52: Percentage of problems solved in the FOL-CR domain set.

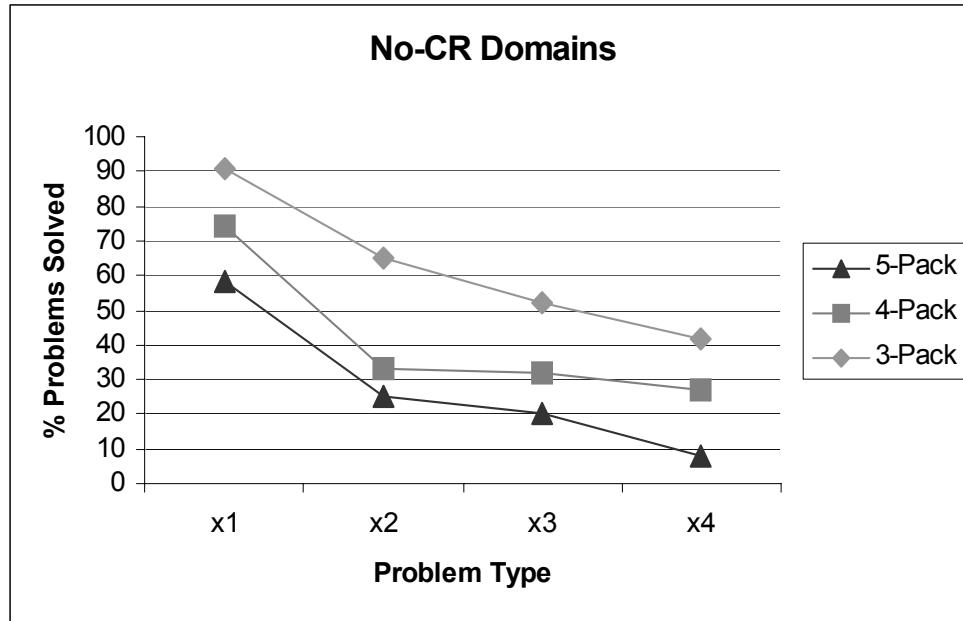


Figure 4.53: Percentage of problems solved in the No-CR domain set.

Figure 4.54 and Figure 4.55 show the number of nodes expanded in the FOL-CR and No-CR domains if we count all unsolved problems as having been solved with exactly 20,000 expanded nodes each. The actual data trends now adhere more closely to the expected data trends, with the only statistical anomaly now occurring between the x3 and x4 problems in the FOL-CR 5-Pack domain. This anomaly may be insignificant, or it may be indicative of the statistical frequency of “hard” problems relative to the size of the planning domain or problem, similar to the statistical frequency of “hard” 3-SAT problems relative to the number of clauses and variables. We leave this question to future research.

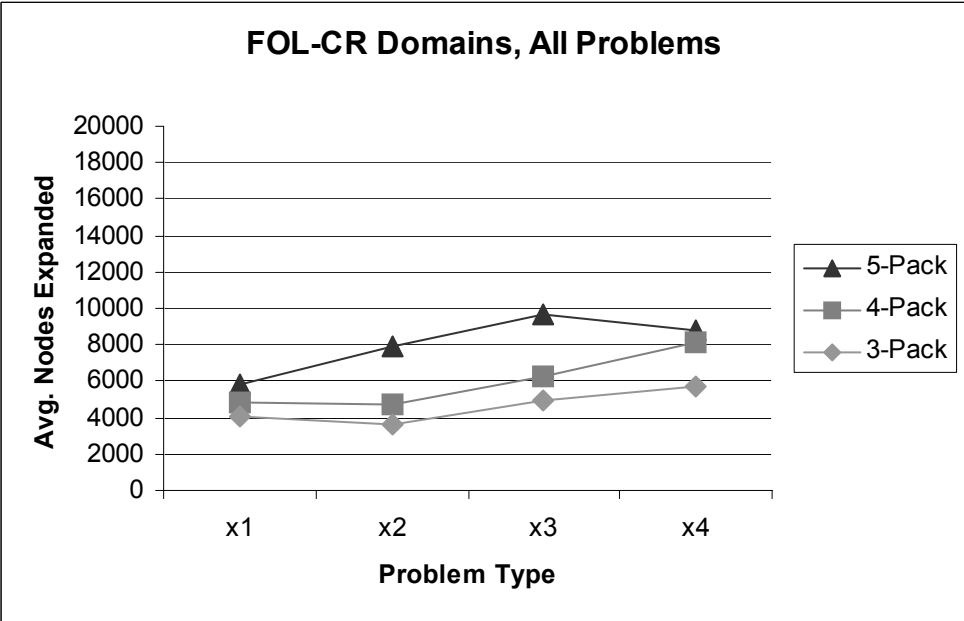


Figure 4.54: Experimental results for all problems from the FOL-CR domain set.

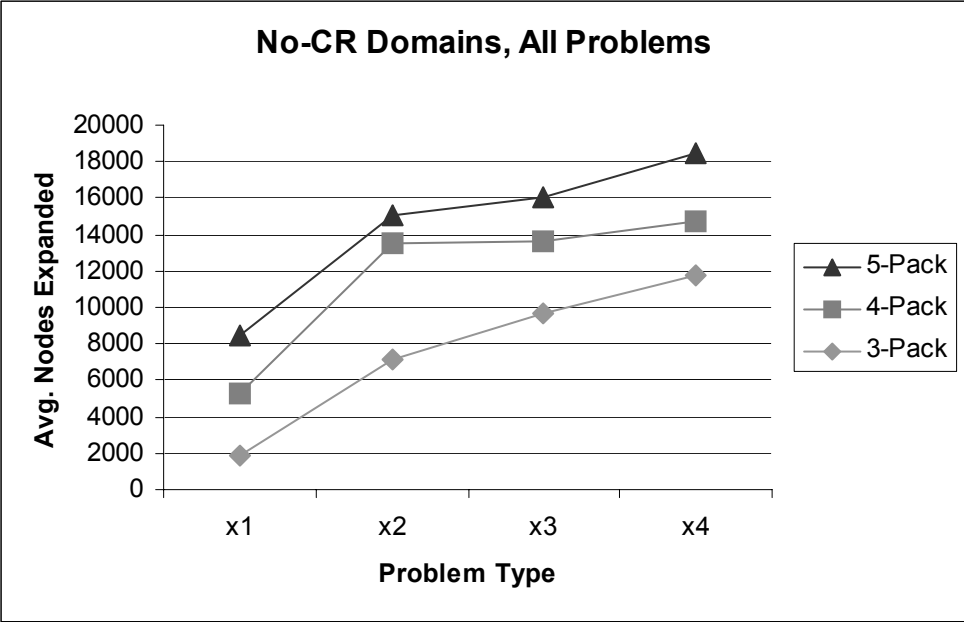


Figure 4.55: Experimental results for all problems from the No-CR domain set.

The FOL-CR domains differ from the No-CR domains in that the FOL-CR domains are essentially equivalent to the Full-CR and Part-CR domains on the problems solved within the 20,000-node limit, whereas the No-CR domains demonstrate more variance within

their results. By virtue of its control rules, a FOL-CR domain either will solve a problem efficiently or will not solve a problem within a reasonable amount of time. This is in contrast to the No-CR domains, which demonstrate a greater deviation in results due to their random nature.

Figure 4.56, Figure 4.57, and Figure 4.58 in section 4.2.1 reiterate the performance similarities between the FOL-CR domain and the Full-CR and Part-CR domains for problems solved within the 20,000-node limit. In contrast, Figure 4.59, Figure 4.60, and Figure 4.61 in section 4.2.2 demonstrate the inferiority of the FOL-CR and No-CR domains as predicted by Hall’s Theorem.

4.2.1 Experimental results, problems solved within the 20,000 node limit

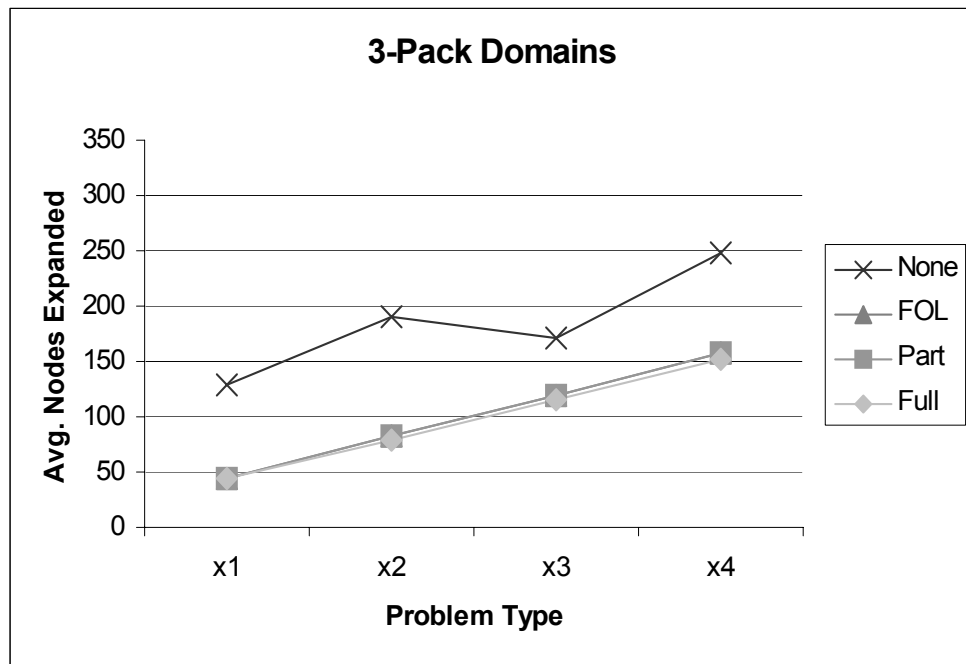


Figure 4.56: Experimental results for problems solved within the 20,000-node limit in the 3-Pack domain set.

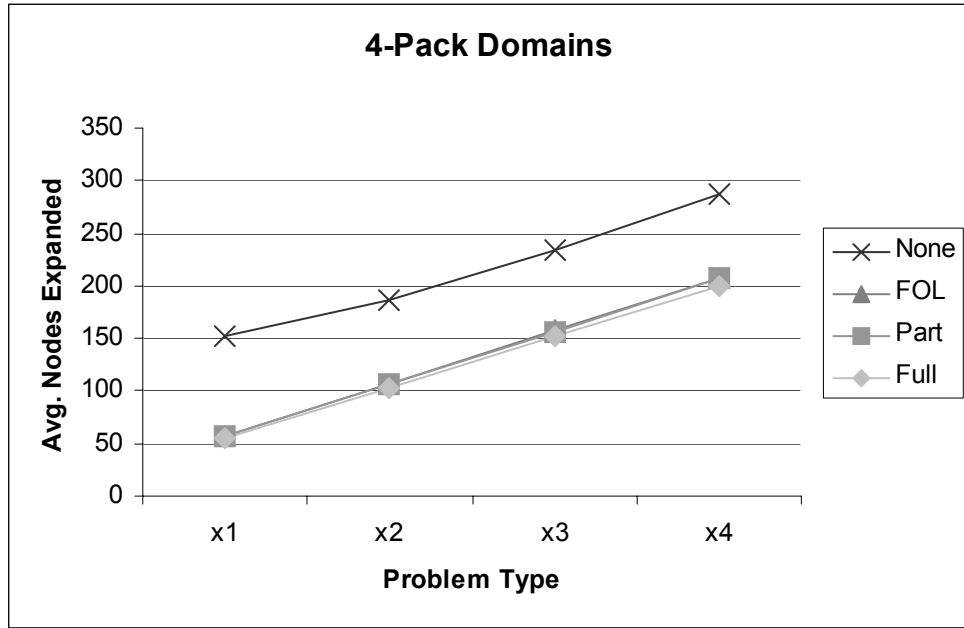


Figure 4.57: Experimental results for problems solved within the 20,000-node limit in the 4-Pack domain set.

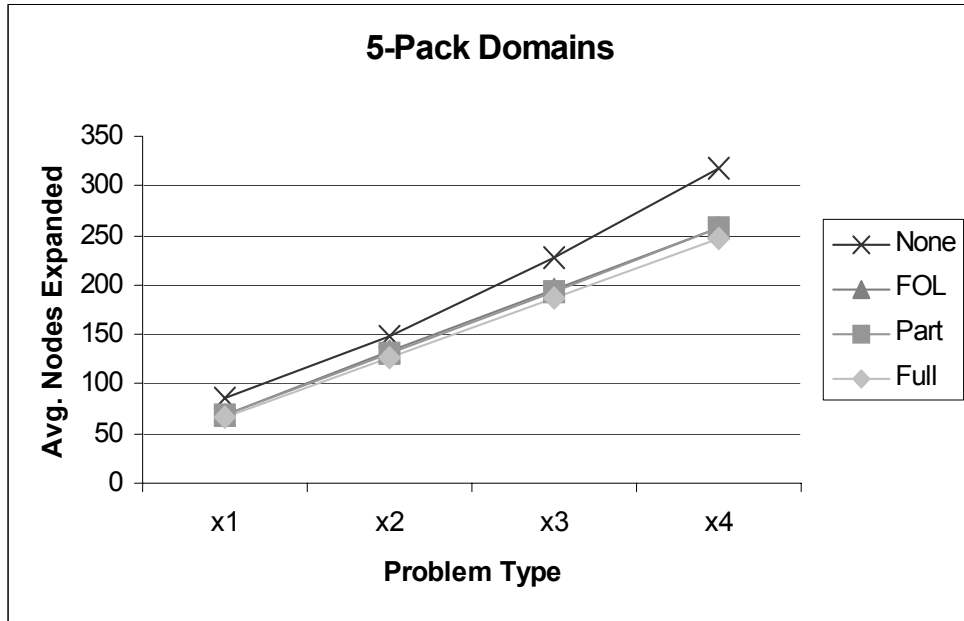


Figure 4.58: Experimental results for problems solved within the 20,000-node limit in the 5-Pack domain set.

4.2.2 Experimental results, all problems

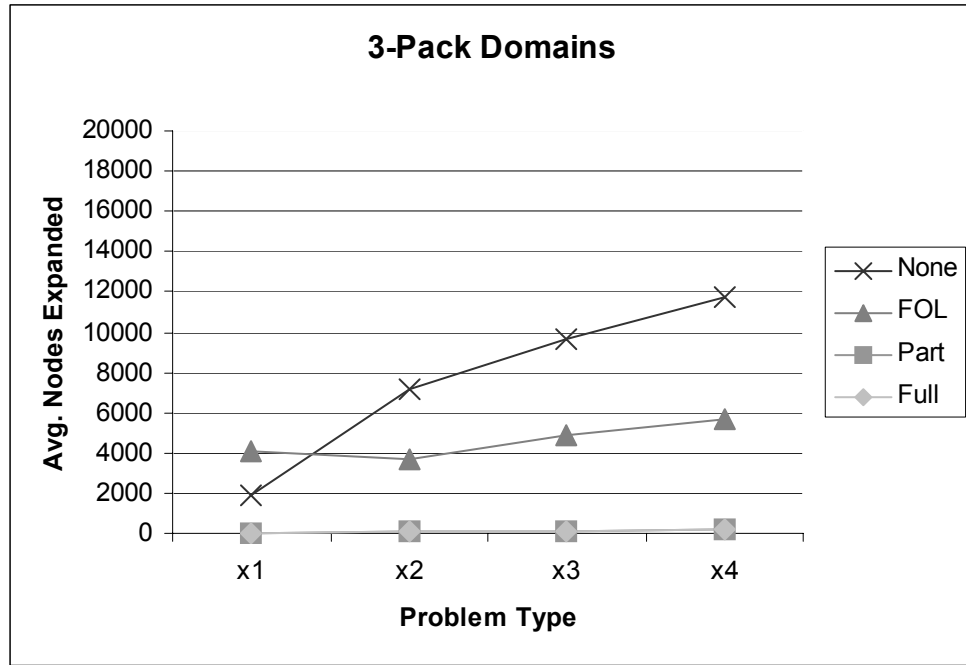


Figure 4.59: Experimental results for all problems in the 3-Pack domain set.

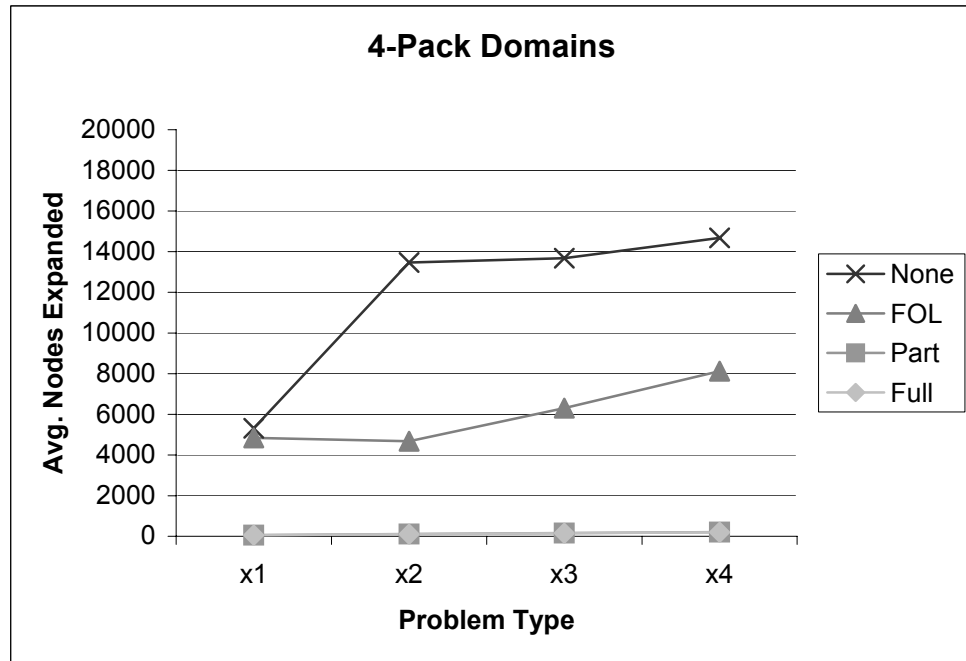


Figure 4.60: Experimental results for all problems in the 4-Pack domain set.

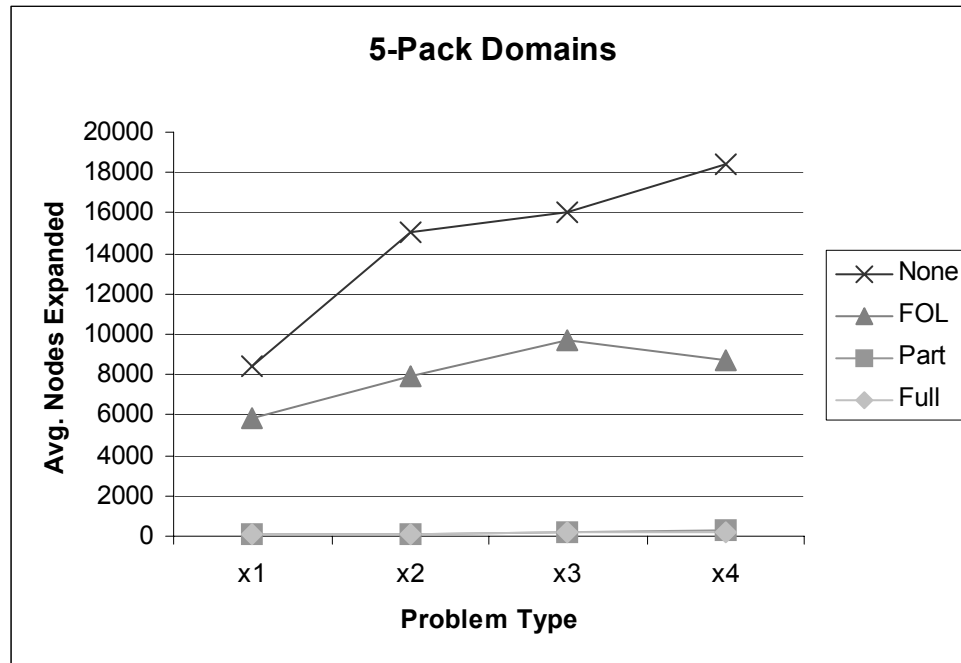


Figure 4.61: Experimental results for all problems in the 5-Pack domain set.

As shown above, the FOL-CR domains have results that are essentially equivalent to the Full-CR and Part-Cr domains for those problems that are solved within a reasonable amount of time, but that the FOL-CR results more closely resemble the No-CR results on average. The use of FOL control rules is certainly an improvement over the use of no control rules at all, but the use of linear inequality control rules (either in addition to or in lieu of FOL control rules) is obviously superior to the use of FOL control rules.

It is evident that the Part-CR and Full-CR domains exhibit nearly identical performance, as predicted. The FOL-CR domains exhibit similar performance in terms of nodes expanded on those problems solved within a reasonable amount of time, but these domains do not exhibit similar performance in terms of the number of problems solved.

The No-CR domains exhibit far worse performance on the problems solved within the node limit, and in addition, these domains solve far fewer problems within the node limit.

5 Applications and Future Research

In Chapter 0, we proved that linear inequality control rules are effective in solving maximum-matching problems in pathological domains. In this chapter, we will demonstrate their effectiveness in solving stable-matching problems, in inter-agent negotiation for parallel planning, and in automated and mixed-initiative control rules used for goal erosion.

5.1 Stable-Matching

Thus far, we have only covered the use of linear inequality control rules in maximum-matching problems, but stable-matching problems will also benefit from their use. A stable-matching problem is one in which the utility of the graph edges possess varying weights, whereas a maximum-matching problem assumes that all graph edges possess the same weight. Figure 5.1 gives a sample 3-Pack Stable-Matching domain.

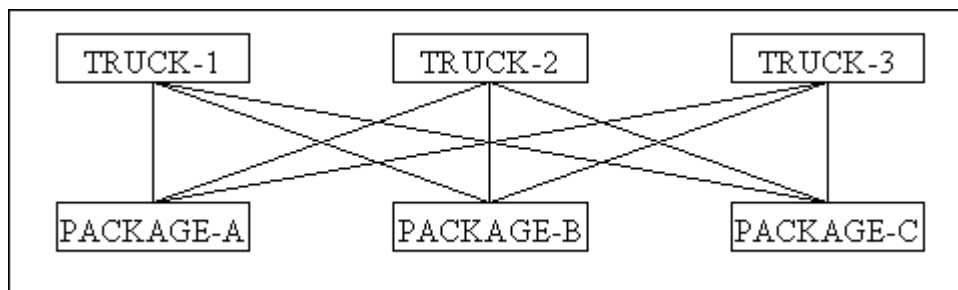


Figure 5.1: 3-Pack Stable-Matching domain.

Note that this is a complete graph, and so it is a trivial maximum-matching problem. However, if the graph edges have varying weights (Table 1), then this is a stable-matching problem. The purpose of a stable-matching problem is to maximize the utility of all matches, where the weight of the graph edge used in a match denotes the utility of the match.

	PACKAGE-A	PACKAGE-B	PACKAGE-C
TRUCK-1	$x1A$	$x1B$	$x1C$
TRUCK-2	$x2A$	$x2B$	$x2C$
TRUCK-3	$x3A$	$x3B$	$x3C$

Table 1: Edge weights for the 3-Pack Stable-Matching domain.

The parameters in Table 1 specify the edge weights for the given matches. For example, the parameter $x1B$ denotes the edge weight for a Truck-1 – Package-B match.

While state-space planners can efficiently solve some stable-matching domains using FOL control rules (Table 2), the general stable-matching problem carries with it the same problem as the general maximum-matching problem in that pathological dependency cycles may be contained within the edge weight matrix (Table 3).

	PACKAGE-A	PACKAGE-B	PACKAGE-C
TRUCK-1	1	0	0
TRUCK-2	0	1	0
TRUCK-3	0	0	1

Table 2: A set of sample edge weights that permit an efficient solution using FOL control rules.

	PACKAGE-A	PACKAGE-B	PACKAGE-C
TRUCK-1	1	1	0
TRUCK-2	1	0	1
TRUCK-3	0	1	1

Table 3: A set of sample edge weights that do *not* permit an efficient solution using FOL control rules.

Table 2 gives the edge weights of a trivial 3-Pack Stable-Matching domain, for which the planner can, e.g., use FOL control rules to always match Truck-1 objects to Package-A objects. However, Table 3 gives the edge weights of a 3-Pack Stable-Matching domain that is essentially equivalent to the 3-Pack Pathological domain (Figure 2.7), and so FOL control rules cannot efficiently locate a solution to all problems in this domain.

We can apply a combination of FOL and linear inequality control rules to a stable-matching domain to maximize utility. Given an arbitrary 2x2 edge matrix (Table 4), we can enumerate the possible orderings that result from the matrix (Figure 5.2).

	PACKAGE-A	PACKAGE-B
TRUCK-1	$x1A$	$x1B$
TRUCK-2	$x2A$	$x2B$

Table 4

1. $x1A = x2A$
 - a. $x1B = x2B$
 - b. $x1B > x2B$
 - c. $x1B < x2B$
2. $x1A > x2A$
 - a. $x1B = x2B$
 - b. $x1B < x2B$
 - c. $x1B > x2B$
 - i. $x1A - x2A = x1B - x2B$
 - ii. $x1A - x2A < x1B - x2B$
 - iii. $x1A - x2A > x1B - x2B$
3. $x1A < x2A$
 - a. $x1B = x2B$
 - b. $x1B < x2B$
 - i. $x1A - x2A = x1B - x2B$
 - ii. $x1A - x2A < x1B - x2B$
 - iii. $x1A - x2A > x1B - x2B$
 - c. $x1B > x2B$

Figure 5.2: Parameter enumeration for the 2-Pack Stable-Matching domain.

FOL control rules can encode all of these precedence orderings (Figure 5.3). $\text{Pre}(X, Y)$ denotes that X takes higher precedence than Y.

1a.	No precedence.
1b.	Pre(Package-B, Package-A). Pre(x1B, x2B).
1c.	Pre(Package-B, Package-A). Pre(x2B, x1B).
2a.	Pre(Package-A, Package-B). Pre(x1A, x2A).
2b.	Pre(x1A, x2A). Pre(x2b, x1b).
2c.	Pre(x1A, x2A). Pre(x1B, x2B).
	i. No additional precedence.
	ii. Pre(Package-B, Package-A).
	iii. Pre(Package-A, Package-B).
3a.	Pre(Package-A, Package-B). Pre(x2A, x1A).
3b.	Pre(x2A, x1A). Pre(x2B, x1B).
	i. No additional precedence.
	ii. Pre(Package-B, Package-A).
	iii. Pre(Package-A, Package-B).
3c.	Pre(x2A, x1A). Pre(x1B, x2B).

Figure 5.3: Precedence orderings for the 2-Pack Stable-Matching domain.

The planner can encode the precedence orderings in Figure 5.3 with Goal, Operator, or Binding preference control rules using FOL. 2-Pack domains cannot be pathological, so the domain cannot benefit from linear inequality control rules.

Therefore, given the 3-Pack domain in Table 1, we can split this domain into nine 2-Pack sub-domains (Figure 5.4). In general, we can decompose an i -Pack domain into $(i \ C \ 2)$ 2-Pack sub-domains.

i.	PA	PB	ii.	PA	PC	iii.	PB	PC
T1	x_{11}	x_{12}	T1	x_{11}	x_{13}	T1	x_{12}	x_{13}
T2	x_{21}	x_{22}	T2	x_{21}	x_{23}	T2	x_{22}	x_{23}
iv.	PA	PB	v.	PA	PC	vi.	PB	PC
T1	x_{11}	x_{12}	T1	x_{11}	x_{13}	T1	x_{12}	x_{13}
T3	x_{31}	x_{32}	T3	x_{31}	x_{33}	T3	x_{32}	x_{33}
vii.	PA	PB	viii.	PA	PC	ix.	PB	PC
T2	x_{21}	x_{22}	T2	x_{21}	x_{23}	T2	x_{22}	x_{23}
T3	x_{31}	x_{32}	T3	x_{31}	x_{33}	T3	x_{32}	x_{33}

Figure 5.4: Decomposed 3-Pack Stable-Matching domain.

The 3-Pack domain's control rule set is the union of all control rules generated by its sub-domains. If the resulting set of FOL control rules is consistent, then the domain is non-pathological. If it is inconsistent, then the domain is pathological, and the planner must supplement FOL control rules with linear inequality control rules. For example, if we split the domain in Table 3 into its component sub-domains and generate FOL control rules from these using the rules in Figure 5.2 and Figure 5.3, then the decomposition gives us:

- I. From the first sub-matrix:
 1. Truck-1 – Package-B matches have precedence over Truck-1 – Package-A matches.
 2. Truck-1 – Package-B matches have precedence over Truck-2 – Package-B matches.

3. Truck-1 – Package-A matches and Truck-2 – Package-A matches are unordered.
- II. From the second sub-matrix:
1. Truck-2 – Package-C matches have precedence over Truck-2 – Package-A matches.
 2. Truck-2 – Package-C matches have precedence over Truck-1 – Package-C matches.
 3. Truck-1 – Package-A matches and Truck-2 – Package-A matches are unordered.
- III. From the third sub-matrix:
1. Truck-2 – Package-C matches have precedence over Truck-1 – Package-C matches.
 2. Truck-1 – Package-B matches have precedence over Truck-2 – Package-B matches.
 3. Truck-1 – Package-B matches and Truck-2 – Package-C matches are unordered.
- IV. From the fourth sub-matrix:
1. Truck-1 – Package-A matches have precedence over Truck-1 – Package-B matches.
 2. Truck-1 – Package-A matches have precedence over Truck-3 – Package-A matches.
 3. Truck-1 – Package-B matches and Truck-3 – Package-B matches are unordered.
- V. From the fifth sub-matrix:
1. Truck-1 – Package-A matches have precedence over Truck-3 – Package-A matches.
 2. Truck-3 – Package-C matches have precedence over Truck-1 – Package-C matches.
 3. Truck-1 – Package-A matches and Truck-3 – Package-C matches are unordered.
- VI. From the sixth sub-matrix:
1. Truck-3 – Package-C matches have precedence over Truck-1 – Package-C matches.
 2. Truck-3 – Package-C matches have precedence over Truck-3 – Package-B matches.
 3. Truck-1 – Package-B matches and Truck-3 – Package-B matches are unordered.

VII. From the seventh sub-matrix:

1. Truck-2 – Package-A matches have precedence over Truck-3 – Package-A matches.
2. Truck-3 – Package-B matches have precedence over Truck-2 – Package-B matches.
3. Truck-2 – Package-A matches and Truck-3 – Package-B matches are unordered.

VIII. From the eighth sub-matrix:

1. Truck-2 – Package-A matches have precedence over Truck-3 – Package-A matches.
2. Truck-2 – Package-A matches have precedence over Truck-2 – Package-C matches.
3. Truck-2 – Package-C matches and Truck-3 – Package-C matches are unordered.

IX. From the ninth and final sub-matrix:

1. Truck-3 – Package-B matches have precedence over Truck-2 – Package-B matches.
2. Truck-3 – Package-B matches have precedence over Truck-3 – Package-C matches.
3. Truck-2 – Package-C matches and Truck-3 – Package-C matches are unordered.

After checking these FOL control rules for consistency, the remaining (i.e., consistent) precedence orderings are:

1. Truck-1 – Package-A matches have precedence over Truck-3 – Package-A matches.
2. Truck-1 – Package-B matches have precedence over Truck-2 – Package-B matches.
3. Truck-2 – Package-A matches have precedence over Truck-3 – Package-A matches.
4. Truck-2 – Package-C matches have precedence over Truck-1 – Package-C matches.

5. Truck-3 – Package-B matches have precedence over Truck-2 – Package-B matches.
6. Truck-3 – Package-C matches have precedence over Truck-1 – Package-C matches.

Inconsistencies in the FOL rules also provide a set of unresolved precedence orderings:

1. Truck-1 – Package-A matches and Truck-2 – Package-A matches have indeterminate ordering.
2. Truck-1 – Package-B matches and Truck-3 – Package-B matches have indeterminate ordering.
3. Truck-2 – Package-C matches and Truck-3 – Package-C matches have indeterminate ordering.

Not surprisingly, these three indeterminate orderings correspond exactly to the first three linear inequality control rules for the 3-Pack domain (Figure 3.2) after making the appropriate variable substitutions. We have thereby reduced the set of Stable-Matching domains to the set of Maximum-Matching domains, expanding the role of linear inequality control rules in the process.

5.2 Parallel Planning and Multiagent Negotiation

Linear inequality control rules are necessary for efficient planning in a pathological domain, but they are also useful in a non-pathological domain when we have the potential to split the domain into pathological sub-domains. For example, recall that our 3-Pack Non-Pathological domain (Figure 2.7) had a linear inequality set (Figure 2.8) that allowed FOL control rules to provide efficient search control.

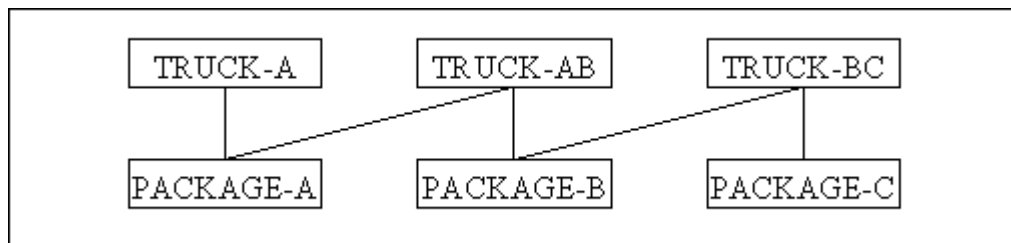


Figure 2.7:

1. $t_a + t_{ab} \geq p_a$
2. $t_{ab} + t_{bc} \geq p_b + p_c$
3. $t_{bc} \geq p_c$
4. $t_a + t_{ab} + t_{bc} \geq p_a + p_b + p_c$

Figure 2.8: Linear inequality set for the 3-Pack Non-Pathological domain.

However, linear inequality control rules are still useful for guiding search in non-pathological domains. It may be inappropriate to encode the goal/operator/binding precedence that would allow FOL control rules to efficiently guide search. For example, if Truck-AB – Package-B matches had a much higher utility than Truck-BC – Package-B matches, and if Truck-AB – Package-A matches had a much higher utility than Truck-A – Package-A matches, then it would be inappropriate to employ an FOL precedence that yields a lower utility, and so the number of Truck-AB matches must be maximized.

Unfortunately, any Truck-AB – Package-A match that occurs while there are still Truck-A matches may violate equation 2, and any Truck-AB – Package-B match that occurs while there are still Truck-BC matches may violate equation 1. The solution is to use a linear inequality control rule (Figure 5.5). Note that this rule assumes that Package-A goals have precedence over Package-B goals.

```
(if (and
    (current-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A)
    (~ (greater-than (TRUCK-AB TRUCK-BC) PACKAGE-B)))
    (then reject operator Unload-Truck-AB))
```

Figure 5.5: Linear inequality control rule used for multiagent coordination in the 3-Pack Non-Pathological domain.

This same principle applies to using multiagent systems to solve a problem in parallel. If we split the 3-Pack Non-Pathological domain among three agents such that: Agent-A controls the Truck-A subtype (Figure 5.6), Agent-AB controls the Truck-AB subtype (Figure 5.7), and Agent-BC controls the Truck-BC subtype (Figure 5.8), then Agent-AB would run the risk of violating one of the linear inequalities if it took any action before either Agent-A or Agent-BC had exhausted its supply of Truck objects.

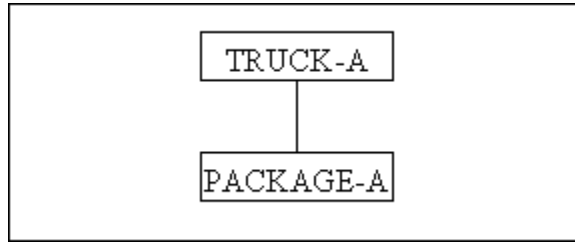


Figure 5.6: Agent-A's sub-domain.

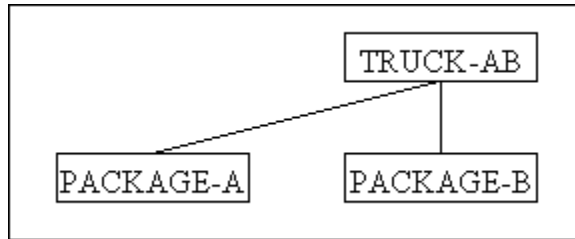


Figure 5.7: Agent-AB's sub-domain.

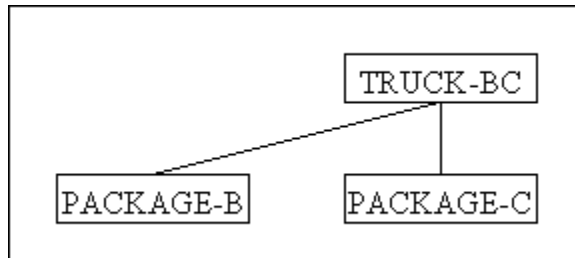


Figure 5.8: Agent-BC's sub-domain.

However, if Agent-AB uses the linear inequality control rule in Figure 5.5, then it can commit to Truck-AB – Package-A matches before either Agent-A or Agent-BC have exhausted their supplies of Truck objects. Thus, the planner benefits from parallelism by using linear inequality control rules in non-pathological domains, whereas strict use of FOL control rules would force Agent-AB to remain idle at the start of planning.

The issue of multiagent negotiation comes into play for larger domains, such as the 5-Pack Pathological domain (Figure 3.23). If we subdivide this domain as we did the 3-Pack Non-Pathological domain such that each of five agents controls one of the five

Truck subtypes, then the domain's linear inequality set (Figure 4.14) is also subdivided as appropriate (Figure 5.9).

1.	$tea + tab \geq pa$
2.	$tab + tbc \geq pb$
3.	$tea + tab + tbc \geq pa + pb$
4.	$tab + tbc + tcd \geq pb + pc$
5.	$tde + tea + tab \geq pe + pa$
6.	$tea + tab + tbc + tcd \geq pa + pb + pc$
7.	$tab + tbc + tcd + tde \geq pb + pc + pd$
8.	$tcd + tde + tea + tab \geq pd + pe + pa$
9.	$tde + tea + tab + tbc \geq pe + pa + pb$

Figure 5.9: Full linear inequality set for Agent-AB's sub-domain from the 5-Pack Pathological domain.

Agent-BC, Agent-CD, Agent-DE, and Agent-EA are isomorphic to Agent-AB, and so we have only shown the linear inequality set for Agent-AB. These five agents are then given linear inequality reject rules similar to those used in the Full-CR set of control rules, thereby allowing parallelism without running the risk of needing to backtrack.

However, such a setup can become computationally expensive for a larger domain. If the values of the Truck variables are stored locally (e.g., Agent-BC stores and modifies the value of the *tbc* variable), then this setup requires that every agent be capable of communicating with every other agent, leading to a large communications overhead. If instead the values are stored in shared memory, then the administrative overhead can become prohibitive. These unsatisfactory solutions do not even begin to address the issues of communicating/storing the Package variables, whose values must be somehow locked by the agent that is currently modifying them.

We can improve network efficiency by integrating FOL control rules with the linear inequality control rule set. As was noted in the discussion of the Part-CR control rule set in Chapter 0, we can use decoupling to reduce the number of linear inequality control rules. For example, we can choose to give a higher precedence to Package-A objects than to Package-B objects in Agent-AB to reduce the control rule set (Figure 5.10).

- | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none">1. $tab + tbc \geq pb$2. $tab + tbc + tcd \geq pb + pc$3. $tab + tbc + tcd + tde \geq pb + pc + pd$ |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Figure 5.10: Reduced linear inequality set for Agent-AB's sub-domain from the 5-Pack Pathological domain.

We significantly reduce the complexity of the linear inequality control rule set through the addition of one FOL control rule. Agent-AB matches its Truck-AB objects to Package-A objects until doing so would violate one of the linear inequalities in the reduced set, after which the agent matches its remaining Truck-AB objects to Package-B objects. Note that this effectively solves the problem of determining the value of the Package variables, for Truck-AB has ability to store and modify the Package-A variable locally until one of the inequalities is violated, after which Agent-AB passes control of the Package-A variable to Agent-EA and Agent-AB then assumes control of the Package-B variable.

Unfortunately, this setup still requires that each agent be able to communicate with all but one of the other agents in the system, assuming that Truck variables are stored locally.

An alternative would be for every agent to be given the ability to communicate with its two adjacent agents and no others if we stipulate that all agents be given the same left-to-right goal precedence (e.g., Agent-AB follows the left-to-right precedence rule $\text{Pre}(\text{Package-A}, \text{Package-B})$, and Agent-BC follows the left-to-right precedence rule $\text{Pre}(\text{Package-B}, \text{Package-C})$), for in this way the agents' control rules will have maximum overlap. For example, Agent-AB needs access to the variables tbc , tcd , and tde for the third linear inequality in the reduced set, but Agent-BC also needs access to tcd and tde , so Agent-AB can find the values of all relevant Truck variables from its adjacent agent.

This alternative has the unfortunate side effect of introducing a delay in updating the variables for the agents' reduced inequality sets, for every agent must wait for the updated values to propagate before taking another action. This delay is easily circumvented by stipulating that the agents act synchronously, which in effect has already been stipulated by requiring them to propagate the values of the Truck variables. With this stipulation in effect, every agent can update the values of other agents' Truck variables by assuming that their values are decremented in synch with all of the other agents. Thus, whenever an agent decrements its own Truck variable it also decrements the locally stored copies of the other agents' Truck variables. The agents follow the same synchronous decrementing practice with Package variables.

This demonstrates the flexibility of linear inequality control rules in multiagent systems. The programmer has access to a wide range of system architectures, from the fully parallel asynchronous architecture afforded by use of a Full-CR set of control rules, to the

reduced communication requirements afforded by use of the synchronous Part-CR set of control rules.

5.3 Automatic and Mixed-Initiative Goal Transformations

We noted previously that use of linear inequality reject rules is a mixed blessing when attempting to solve problems that do not have a solution. On the positive side, search will immediately terminate when a problem has no solution, rather than first expanding large sections of the search tree. On the negative side, the planner will not be able to find a partial solution to the problem when reject rules terminate search in this manner. However, when the search process uses eroding goal transformations (Cox and Veloso, 1998; Cox, Elahi, and Cleereman, 2003), then there are no negative side effects to the use of linear inequality control rules.

A goal transformation is a process that changes a goal during the planning process, often either eroding the goal or enhancing the goal. We erode a goal when there are not sufficient resources to accomplish the original goal, but when there are sufficient resources to accomplish a weakened version of the original goal. For example, in a problem containing six Packages to be moved, an erosion will change the goal “move all packages” to the goal “move five packages” if there are only five Truck objects available. We enhance a goal when additional resources become available, allowing the planner to accomplish more than it would otherwise be able to. For example, the planner may enhance the goal “assign one truck per package” to the goal “assign two trucks per package” in a problem wherein there are many more Truck objects than Package objects.

We will examine two kinds of eroding goal transformation: automatic and mixed-initiative. The planner performs automatic goal transformations with no additional user input needed, while the user and planner coordinate to perform mixed-initiative goal transformations (Cox, 2003).

Consider, for example, the No-Soln (No Solution) problem in Figure 5.11.

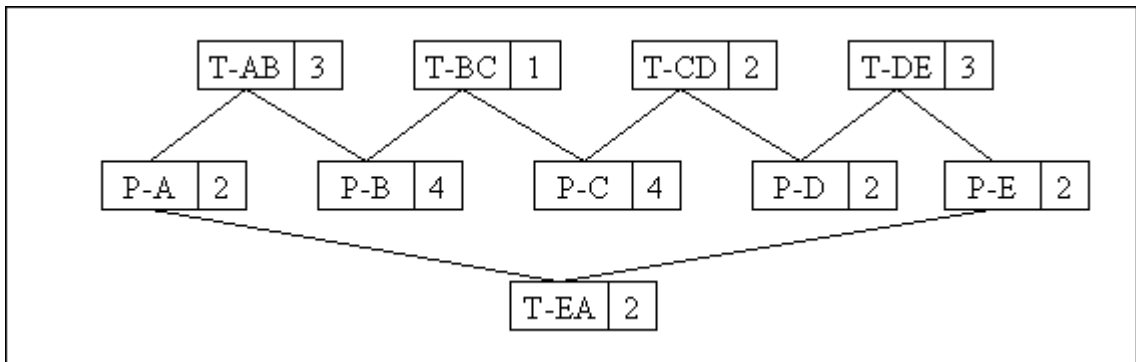


Figure 5.11: No-Soln problem for the 5-Pack Pathological domain.

There are 14 Package objects and 11 Truck objects in this problem, so there are insufficient Truck resources to move all Package objects.

5.3.1 Automatic Goal Transformations

We use automatic goal transformations (Figure 5.12, Figure 5.13, Figure 5.14, Figure 5.15, Figure 5.16) to erode goals that the planner cannot achieve *regardless* of how the user distributes Truck resources. These goal transformations encode the unary (one Package variable on the right hand side) linear inequalities 1-5 (Figure 4.14) by deleting goals until the inequalities are satisfied. Note that a “delete goal” control rule is fundamentally different from a “reject goal” control rule. A reject rule leaves open the

possibility that the planner will expand a goal elsewhere on the search tree. A delete rule removes the goal from the problem³.

```
(if (and
    (candidate-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A)
    (~ (greater-than (TRUCK-EA TRUCK-AB) PACKAGE-A)))
    (then delete goal (at-obj <obja> <loc>)))
```

Figure 5.12: Automatic goal transformation for equation 1 (Figure 4.14).

```
(if (and
    (candidate-goal (at-obj <objb> <loc>))
    (type-of-object <objb> PACKAGE-B)
    (~ (greater-than (TRUCK-AB TRUCK-BC) PACKAGE-B)))
    (then delete goal (at-obj <objb> <loc>)))
```

Figure 5.13: Automatic goal transformation for equation 2 (Figure 4.14).

```
(if (and
    (candidate-goal (at-obj <objc> <loc>))
    (type-of-object <objc> PACKAGE-C)
    (~ (greater-than (TRUCK-BC TRUCK-CD) PACKAGE-C)))
    (then delete goal (at-obj <objc> <loc>)))
```

Figure 5.14: Automatic goal transformation for equation 3 (Figure 4.14).

```
(if (and
    (candidate-goal (at-obj <objd> <loc>))
    (type-of-object <objd> PACKAGE-D)
    (~ (greater-than (TRUCK-CD TRUCK-DE) PACKAGE-D)))
    (then delete goal (at-obj <objd> <loc>)))
```

Figure 5.15: Automatic goal transformation for equation 4 (Figure 4.14).

³ PRODIGY does not support delete rules. We use them here for pedagogical reasons.

```

(if (and
    (candidate-goal (at-obj <obje> <loc>))
    (type-of-object <obje> PACKAGE-E)
    (~ (greater-than (TRUCK-DE TRUCK-EA) PACKAGE-E)))
    (then delete goal (at-obj <obje> <loc>)))

```

Figure 5.16: Automatic goal transformation for equation 5 (Figure 4.14).

The automatic goal transformation in Figure 5.14 fires once, deleting one (at-obj Package-C <loc>) goal from the problem (Figure 5.17).

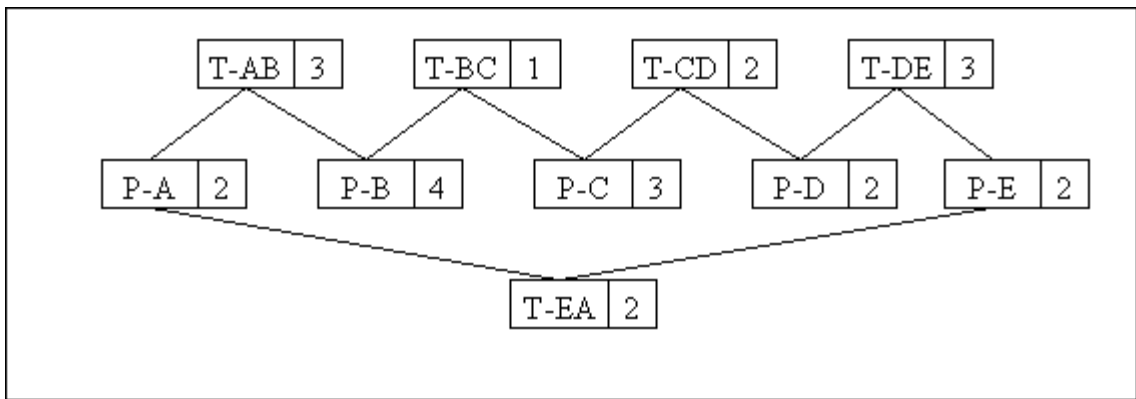


Figure 5.17: An automatic goal transformation (Figure 5.14) has deleted one Package-C object.

This goal erosion still leaves us with 13 Package objects and 11 Truck objects. We must remove two more Package objects. While we could use automatic goal transformations for this purpose, this would require that we impose an arbitrary precedence on the Package objects. We therefore use mixed-initiative goal transformations instead, so that we can give the user a choice of which goals to delete from the problem.

5.3.2 Mixed-Initiative Goal Transformations

These binary (two Package variables on the right hand side of the linear inequality) mixed-initiative goal transformations (Figure 5.18, Figure 5.19, Figure 5.20, Figure 5.21,

Figure 5.22) employ a (user-selection g1 g2 &rest) function. This function asks the user which of two or more goals should be deleted from the problem, and returns that goal.

```
(if (and
    (candidate-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A)
    (candidate-goal (at-obj <objb> <loc>))
    (type-of-object <objb> PACKAGE-B)
    (~ (greater-than (TRUCK-EA TRUCK-AB TRUCK-BC)
                     (PACKAGE-A PACKAGE-B))))
    (then delete goal
      (user-selection (at-obj <obja> <loc>)
                     (at-obj <objb> <loc>))))
```

Figure 5.18: Mixed-initiative goal transformation for equation 6 (Figure 4.14).

```
(if (and
    (candidate-goal (at-obj <objb> <loc>))
    (type-of-object <objb> PACKAGE-B)
    (candidate-goal (at-obj <objc> <loc>))
    (type-of-object <objc> PACKAGE-C)
    (~ (greater-than (TRUCK-AB TRUCK-BC TRUCK-CD)
                     (PACKAGE-B PACKAGE-C))))
    (then delete goal
      (user-selection (at-obj <objb> <loc>)
                     (at-obj <objc> <loc>))))
```

Figure 5.19: Mixed-initiative goal transformation for equation 7 (Figure 4.14).

```
(if (and
    (candidate-goal (at-obj <objc> <loc>))
    (type-of-object <objc> PACKAGE-C)
    (candidate-goal (at-obj <objd> <loc>))
    (type-of-object <objd> PACKAGE-D)
    (~ (greater-than (TRUCK-BC TRUCK-CD TRUCK-DE)
                     (PACKAGE-C PACKAGE-D))))
    (then delete goal
      (user-selection (at-obj <objc> <loc>)
                     (at-obj <objd> <loc>))))
```

Figure 5.20: Mixed-initiative goal transformation for equation 8 (Figure 4.14).

```

(if (and
    (candidate-goal (at-obj <objd> <loc>))
    (type-of-object <objd> PACKAGE-D)
    (candidate-goal (at-obj <obje> <loc>))
    (type-of-object <obje> PACKAGE-E)
    (~ (greater-than (TRUCK-CD TRUCK-DE TRUCK-EA)
                     (PACKAGE-D PACKAGE-E))))
    (then delete goal
      (user-selection (at-obj <objd> <loc>)
                     (at-obj <obje> <loc>))))

```

Figure 5.21: Mixed-initiative goal transformation for equation 9 (Figure 4.14).

```

(if (and
    (candidate-goal (at-obj <obje> <loc>))
    (type-of-object <obje> PACKAGE-E)
    (candidate-goal (at-obj <obja> <loc>))
    (type-of-object <obja> PACKAGE-A)
    (~ (greater-than (TRUCK-DE TRUCK-EA TRUCK-AB)
                     (PACKAGE-E PACKAGE-A))))
    (then delete goal
      (user-selection (at-obj <obje> <loc>)
                     (at-obj <obja> <loc>))))

```

Figure 5.22: Mixed-initiative goal transformation for equation 10 (Figure 4.14).

The goal transformation in Figure 5.19 either decrements the number of Package-B objects or decrements the number of Package-C objects, bringing the total number of Package objects down to 12. We will assume that the user has deleted a Package-B object (Figure 5.23).

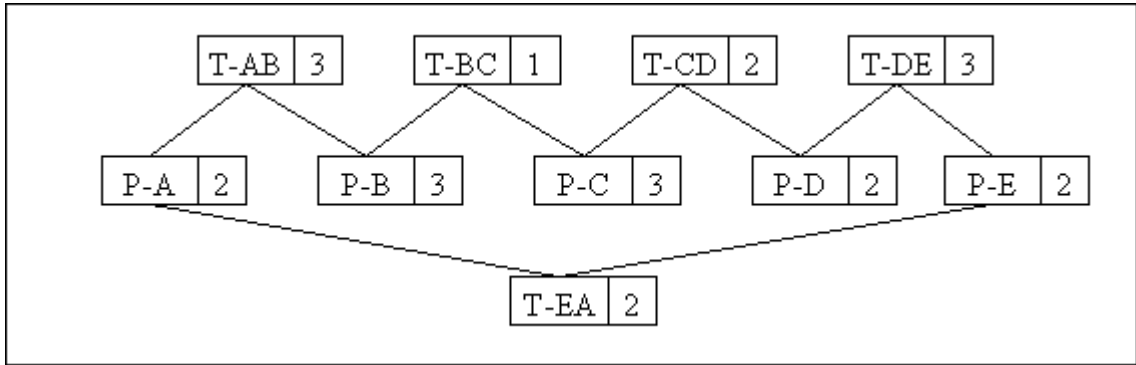


Figure 5.23: The user has removed one Package-B object.

The planner would next move on to the ternary (three Package variables on the right hand side of the equation) mixed-initiative goal transformations, but in the No-Soln problem the ternary goal transformations will not be able to remove the extra Package object, and so these control rules are not shown. We therefore move on to the final n-ary (all Package variables on the right hand side of the equation) linear inequality (Figure 5.24).

$$16. \quad tab + tbc + tcd + tde + tea \geq pa + pb + pc + pd + pe$$

Figure 5.24: N-ary linear inequality for the 5-Pack Pathological domain.

As indicated in this equation, the user may remove *any* one Package object to erode the goals for the No-Soln problem (Figure 5.25).

```

(if (and
  (candidate-goal (at-obj <obja> <loc>))
  (type-of-object <obja> PACKAGE-A)
  (candidate-goal (at-obj <objb> <loc>))
  (type-of-object <objb> PACKAGE-B)
  (candidate-goal (at-obj <objc> <loc>))
  (type-of-object <objc> PACKAGE-C)
  (candidate-goal (at-obj <objd> <loc>))
  (type-of-object <objd> PACKAGE-D)
  (candidate-goal (at-obj <obje> <loc>))
  (type-of-object <obje> PACKAGE-E)
  (~
    (greater-than (TRUCK-AB TRUCK-BC TRUCK-CD
                  TRUCK-DE TRUCK-EA)
                  (PACKAGE-A PACKAGE-B PACKAGE-C
                  PACKAGE-D PACKAGE-E))))
  (then delete goal
    (user-selection (at-obj <obja> <loc>)
                   (at-obj <objb> <loc>)
                   (at-obj <objc> <loc>)
                   (at-obj <objd> <loc>)
                   (at-obj <obje> <loc>))))

```

Figure 5.25: Mixed-initiative control rule for the 5-Pack Pathological domain's n-ary linear inequality.

We will assume that the user has deleted a Package-E object (Figure 5.26).

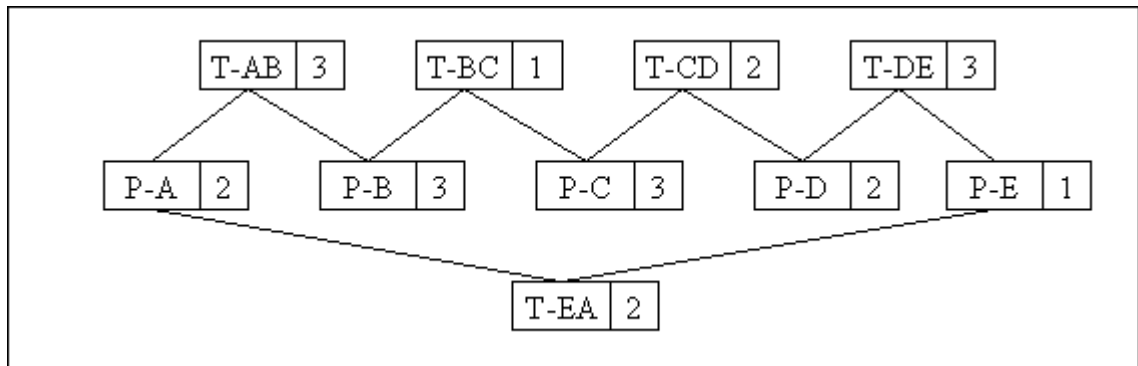


Figure 5.26: The user has removed one Package-E object.

This final goal erosion brings the total number of Package objects down to 11, equal to the total number of Truck objects. All linear inequalities are satisfied, and so the No-

Soln problem now has a solution. The planner may now use any of the search control strategies discussed earlier to find an exact solution to the problem.

6 Conclusion

The arguments presented by this thesis along with the theoretical results given in Hall's Theorem point to the need for state-space planning to move beyond first order logic. The main reason is that it is impossible for FOL to effectively guide search in a pathological domain. The experimental results prove that even a simple pathological planning domain defined using FOL may require search controls with power beyond that of FOL, as evidenced by the fact that first order logic search controls resulted in at least exponential-time planning performance, while linear inequality search controls resulted in linear-time planning performance. In addition to these results on pathological domains, Chapter 5 demonstrates that first order logic's lack of efficiency is a problem in nearly all multiagent planning domains, both pathological and non-pathological. Thus, FOL may be succinct, but it is too inefficient.

A communication protocol for multiagent planning systems must be capable of expressing meta-domain and meta-problem information, and we have proven that this meta-information may require a protocol with an expressive power beyond that of the protocol used to encode the base information. Our development of a communication protocol must therefore reflect this expressive requirement, although in the ideal case the language used to encode a planner's domain and problem information will also be

capable of encoding meta-domain and meta-problem information. Gödel's incompleteness theorem (Gödel, 1992) may prove that no such ideal domain language exists, but we leave this question for the future.

Linear inequality control rules provide necessary and sufficient meta-information for a multiagent planner to effectively solve problems in bipartite matching domains. These control rules also provide necessary (though insufficient) meta-information for a multiagent planner to effectively solve problems in tripartite matching domains. While they are effective in allowing PRODIGY to solve bipartite matching problems in linear time instead of in exponential time, we cannot expect them to be equally effective in allowing PRODIGY to solve NP-Complete problems (such as tripartite matching problem) in polynomial time instead of exponential time. Thus, they perform as well as they can be theoretically expected to perform in achieving optimal solutions to planning problems and have proven themselves as a necessary addition to the inter-agent communication protocol for the PRODIGY planner. Given the expressive shortcomings of other planners' domain and search control languages, we conclude that linear inequality control rules are also a necessary addition to classical planners other than PRODIGY.

The next step in our research is to develop a communication protocol that will enable multiagent planners to achieve imperfect solutions to NP-Complete planning problems, such as the tripartite graph-matching problem described in Chapter 3. Ideally, we would be able to employ an information protocol that is also capable of functioning as a meta-

information protocol, thereby eliminating the need to use separate protocols for domain encoding, search control, and inter-agent communication. This would require the development of a new planning system, a project that will fit in well with a doctoral dissertation. An alternative is to develop a communication protocol that is capable of efficiently representing Prodigy 4.0's search tree thereby allowing multiple agents to better coordinate their efforts without the need for inflexible mathematical constraints. Our concern is that a domain language capable of expressing all necessary information will begin to suffer the same problems encountered by formal specification languages. It may be better to continue to limit the expressiveness of domain languages rather than search for a panacea.

Regardless of the step we take next, the addition of linear inequality control structures to state-space planning clearly represents a solid first step in the right direction towards our goal of developing a general inter-agent communication protocol. Linear inequality control rules allow multiagent systems to solve problems that they could not otherwise solve in a reasonable amount of time, and they permit greater flexibility in designing conversations between agents. Even in a single agent system, linear inequality control rules demonstrate far greater efficiency than first order logic control rules.

7 References

- Asratian, A. S., Denley, T., and Häggkvist, R. (1998). *Bipartite Graphs and Their Applications*. Cambridge University Press: Cambridge.
- Bacchus, F., and Kabanza, F. (2000). Using Temporal Logics to Express Search Control Knowledge for Planning. In *Artificial Intelligence*, vol 116.
- Blum, A., and Furst, M. (1997). Fast Planning Through Planning Graph Analysis. In *Artificial Intelligence*, 90 (pp. 281-300).
- 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 (pp. 333-377).
- Cleereman, K., and Cox, M.T. (2004a). Linear Inequality Control Rules in State-Space Planning: Beyond the First Order Predicate Calculus. In Berkowitz (Ed.), *Proceedings of the 15th Midwest Artificial Intelligence and Cognitive Science Conference* (pp. 148-153).
- Cleereman, K., and Cox, M. T. (2004b). 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: 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.

Fischer, M.J., and Rabin, M.O. (1974). Super-Exponential Complexity of Presburger Arithmetic. In *Proceedings of the SIAM-AMS Symposium in Applied Mathematics*, vol. 7 (pp. 27-41).

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

Gödel, K. (1992). *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*. New York: Dover.

Gusfield, D., and Irving, R.W. (1989). *The Stable Marriage Problem: Structure and Algorithms*. Cambridge, Mass.: MIT Press.

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

Long, D., and Fox, M. (1999). Efficient Implementation of the Plan Graph in STAN. In *Journal of Artificial Intelligence Research*, vol. 10, (pp. 87-115).

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

Penberthy, J.S., and Weld, D. (1992). UCPOP: A Sound, Complete, Partial-Order Planner for ADL. In *Proceedings of KR-92*, (pp. 103-114). Cambridge, MA.

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.

Smith, D.E., Frank, J., and Jónsson, A.K. (2000). Bridging the Gap Between Planning and Scheduling. In *The Knowledge Engineering Review*, Vol. 15:1, (pp. 47-83). United Kingdom: Cambridge University Press.

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.