(2)

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

COMMUNICATIONS-EQUIPMENT DISTRIBUTION
PLANNING USING SEARCH TECHNIQUES

by

Emil K. Velez

March 1989

Thesis Advisor: Neil C. Rowe

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) Code 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)
COMMUNICATIONS_EQUIPMENT DISTRIBUTION PLANNING USING SEARCH TECHNIQUES

12. PERSONAL AUTHOR(S)
Velez, Emil K.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____	1989 March	78

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Communications Equipment Planning, Planning, Search Allocation,

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

This research developed a prototype planning program to plan the distribution of communications equipment within an army organization, using artificial-intelligence search strategies. The system does two searches: it assembles the maximum number of sets of equipment from component equipment procurements, then plans how to assign these sets to organizational units. The program was done on a Sun workstation with a Quintus Prolog compiler. Due to classification requirements, fictitious but typical data was used to test the system. The system can find near-optimal ways to modernize the communications equipment of up to ten battalion-sized signal units, and has been tested through the procurement of three generations of equipment. There were twenty sets of equipment to allocate to the ten units in the final test case. This program can also develop a new distribution plan if unexpected events require changes to the current plan.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Professor Neil C. Rowe	22b. TELEPHONE (*Include Area Code*) (408) 646-2462 22c. OFFICE SYMBOL Code 52Rp

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

* U.S. Government Printing Office: 1986—606-24.

Unclassified

Approved for public release; distribution is unlimited.

COMMUNICATIONS-EQUIPMENT
DISTRIBUTION PLANNING
USING SEARCH TECHNIQUES

by

Emil K. Velez
Captain, United States Army
B.S., Loyola University of Chicago, 1983

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
1989

Author: Emil K. Velez

Approved by: Neil C. Rowe
Neil C. Rowe, Thesis Advisor

Sehung Kwak
Sehung Kwak, Second Reader

Robert McGhee
Robert McGhee, Chairman
Department of Computer Science

K. T. Marshall
Kneale T. Marshall
Dean of Information and Policy Sciences

ABSTRACT

This research developed a prototype program to plan the distribution of communications equipment within an army organization, using artificial-intelligence search strategies. The system does two searches: it assembles the maximum number of sets of equipment from component equipment procurements, then plans how to assign these sets to organizational units. The program was done on a Sun workstation with a Quintus Prolog compiler. Due to classification requirements, fictitious but typical data was used to test the system. The system can find near-optimal ways to modernize the communications equipment of up to ten battalion-sized signal units, and has been tested through the procurement of three generations of equipment. There were twenty sets of equipment to allocate to the ten units in the final test case. This program can also develop a new distribution plan if unexpected events require changes to the current plan.

THESIS DISCLAIMER

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

The reader is also warned that the programs contained in this research have not been tested in many cases. This research sought to develop a methodology to automate the decisions that result in the planning that has an impact on critical resources. Every effort was made to ensure correct and clear program development but the use of the programs without additional research and modification is at the risk of the user.

TABLE OF CONTENTS

I. INTRODUCTION

A. THE COMMUNICATIONS SYSTEM PLANNING PROBLEM

Military organizations must satisfy many constraints. The most pressing is the budget. Unfortunately, budgets for military hardware are always in a state of flux. A hardware procurement planned and contracted for in one fiscal year can be cut drastically the next, requiring renegotiation with contractors, slowdown of production, and possibly higher cost per item; so planning and replanning of the allocation of current and future resources is necessary. This is often true within the U.S. Army Communications Community.

It is generally impossible to fully modernize all military units at one time. Usually equipment procurements are staggered so that some units can be modernized while other units at lesser levels of modernization receive the equipment given up by the first group of units. Planning of communications-equipment allocation is complicated by the possibility that items distributed to a unit may or may not be technically compatible with already acquired equipment; such items are then useless to the unit. The goal of the distribution plan is to allocate necessary amounts of new equipment and redistribute older equipment in an intelligent way. Without a total equipment distribution plan in the past, many items of equipment were distributed on individual timetables and many units received only a subset of the equipment necessary for a set.

B. SCOPE

This thesis developed a prototype planning program that performs basic distribution and redistribution planning for a set of U.S. Army Signal Units, given a particular set of procurements of equipment. The prototype planning program handles up to ten units modernizing each unit a maximum of three times based on the equipment available and modernization level of the units. This plan was developed in a few days or less on a Sun workstation, which is considered a reasonable amount of time.

C. OVERVIEW

Chapter II outlines the different methods of planning, gives an introduction to the search being used, and discusses previous research in this area. Chapter III describes the planning problem with communications equipment and motivates the need for an automated system. Chapter IV describes the two phases to the search to allocate equipment to Army Communications Units. Chapter V describes the technical requirements of the system and the performance of the programs developed in this application. Chapter VI gives conclusions and recommendations for further work in this area. The detailed results from each of the runs are contained in Appendix C.

II. PLANNING AND SEARCH TECHNIQUES

A. SEARCH

Search techniques in artificial intelligence can be used for allocation problems as an alternative to computing all possible methods of allocation and then determining which are optimal. A detailed description of search techniques including the A* search and search engines are contained in *Artificial Intelligence Through Prolog* by Rowe [Ref 1]. The search goes through intermediate steps involving small changes to the working plan that make it gradually more comprehensive, when searching for a solution to a problem. Usually there are many possible intermediate steps in the search; these are called successors or partial solutions. Different search algorithms may take different paths to a solution, and generate different intermediate steps. The solution developed by a search may not be optimal, but usually will be valid and reasonable based on the rules that the search has for progressing toward its solution. These rules are called state-transition or successor rules. A good search technique only develops the most promising alternatives (best partial solutions) during the search. These partial solutions must be evaluated and ranked. The criteria for ranking depend on the type of search and will be explained later.

A solution for the problem of communications-equipment distribution is a plan that allocates all of the available equipment to units based on their needs and priorities. This can be found by a search. The starting state of the problem is a description of the units, the equipment, and the expected procurements. The final plan generated by the program will show when units will be allocated the procured equipment as well as how to reallocate equipment given up by units modernized. In communications equipment distribution, the plan with the most units modernized in the shortest amount of time using the equipment already planned for procurement has the least overall cost and best ranking. An additional cost could be based on the amount of idle equipment, equipment procured before actual use by a unit.

Three important search techniques are A* search, branch-and-bound, and "nopathsearch". The A* search technique develops a plan by making transitions based on the rules that define the finished plan. The new states developed are then added to a storage area called the agenda. Each state in the agenda is given a

numeric rating. This rating is based on two considerations: the cost of this state and a guess of how much additional cost will be necessary to reach a goal from the state. These two factors are added and used to compare the state to other states. The program then chooses the lowest-sum state to find successors of next. This is repeated until either a valid goal is reached or all of the states are gone from the agenda. States are removed from the agenda when all their possible successor states are found. This allows concurrent processing of states.

The branch-and-bound search technique is similar to the A* search. The difference is that each state is evaluated by its cost without adding the additional cost that would be incurred to reach a goal state. We use branch-and-bound in Phase I of our program.

The search used in the Phase II of our program is called "nopathsearch". It is an A* search that does not store information about what steps it took to get to a state. For this search to be used properly, the cost of the state must be able to be evaluated without need for information about how the search got there, as the allocation problem examined in this research. Hutson utilized the "nopathsearch" technique in a thesis that developed training plans for tactical air wing squadrons [Ref 2]. The problem was to schedule training and required inspections during times that squadrons were not deployed or otherwise committed. The inputs to the scheduler application were the required training and deployment timeframes. The scheduler developed a plan for multiple squadrons given the requirements and constraints of each.

B. OTHER PLANNING METHODS

One of the earliest applications of planning was the R1 system developed by Digital Equipment Corporation (DEC) for planning the configuration of Digital Computer Equipment for its customers [Ref 3]. Digital Equipment Corporation manufactures computers that can be configured in many ways. It was a very complex job to configure their computer systems because of all the auxiliary parts needed by different components and configurations of the systems. The system used rules that stated relationships between the many different components produced by DEC, and performed forward chaining. This ad hoc approach works well for similar problems which are rich in constraints.

A .gnificant amount of work in scheduling focused on manufacturing systems [Ref 4]. Production scheduling must work within resource constraints, capacity constraints, and production-time requirements. Savings realized in these environments are easily measured. But these techniques require highly restricted problems.

A different approach to scheduling has been tried by Goehring who developed plans for training using stochastic methods that pseudo-randomly made changes to a suboptimal but reasonable plan [Ref 5]. Stochastic approaches depend on the planner to evaluate a comparative cost to the utility of solution. We did not use this approach since there seems no easy way to get an initial plan for our problem. This approach also generates unnecessary suboptimal possibilities.

Relaxation is a planning technique where all of the possible alternatives for parts of the problem are enumerated in advance and the analysis progressively eliminates them. This technique was considered, but the number of alternatives would be very large for our problem. Each distribution to a unit has a strong possibility of affecting other units. Relaxation is useful when one or more sets of possibilities can be reduced to a single or small set features not characteristic of our problem.

III. DESCRIPTION OF THE APPLICATION

A. PRESENT SITUATION

The US Army Signal Center recently developed a strategy to address the problem of communications-equipment distribution. The Battlefield Communications Review II (BCR II) task force was formed to control all communications equipment procurement, distribution, redistribution, and disposal. The main problem was that many different items of equipment were being procured under separate contracts and timeframes. If an item of equipment was received and could not be used by a unit, the item was usually left idle until a complete set of compatible equipment was received. Furthermore, the unit could not give up the older equipment until they had a complete set of new equipment. Any problem that delays even a single piece of equipment requires a change in the distribution plan that affects many units.

The BCR II distribution plan is complicated by many factors. Budgetary factors limit the amount of equipment that can be purchased, as well as the need for a unit to keep equipment long enough to fully use it; these will not be addressed in this research since they cannot be easily modified due to contractual obligations. Units stationed in different geographic areas have different levels of military risk and value, summarized by a priorities among units. Units must be modernized a single level at a time by doctrine and by the ability of the personnel (in many cases personnel are trained on the next generation of equipment beyond what is present in the units). In the examples used in this research, three levels of modernization were chosen based on the three in the BCR II distribution plan. Once a unit gets new equipment, the old equipment can be redistributed or disposed of. A decision of what unit will receive it must be made again for the old equipment.

The BCR II distribution plan was developed during a conference with up to twenty people working with a large set of blackboards, manually balancing requirements with the planned procurement and unit priorities. When there is a later problem with some equipment, this analytical method is again used to a lesser extent and the plan is revised. This method is time-intensive and manpower-intensive, though computers assist in the tracking the equipment. The premise of this research is that computers can assist much more by proposing necessary changes to the plan.

B. COMMUNICATIONS EQUIPMENT OVERVIEW

Communications Equipment has four basic components: switching equipment, transmission equipment, terminal equipment, and networking equipment. Switching equipment performs local telephone operations supporting local communications users. Terminal equipment is used and managed locally to transmit information through the network, and includes message centers and FAX equipment. Transmission equipment transmits raw communications data from one communications site to another. Networking equipment is the interface between the switching equipment and the transmission equipment, connecting channels from transmission units to various local switches. Each of these components of a communications network must be compatible. Compatibility depends on technical characteristics of the equipment including capacity, size, channel numbers, and communication protocols.

The major problem that the BCR II addressed was that these four types of communications equipment types cannot usually be modernized separately. For example, a new type of transmission equipment cannot always work with an older type of networking equipment. The current BCR II planners constantly have shortages of equipment needed to modernize units. Another problem that must be addressed is compatibility between units. This can be partially resolved in this planning program with the priorities of the units by ensuring that units that are part of the same network have the same priority for equipment allocation.

C. EXAMPLES

Appendix A shows the components and amounts needed in different unit sets of equipment and how a unit may receive single pieces of equipment according to present procurement distribution practices.

IV. DESCRIPTION OF THE PROGRAM

A. PHASE ONE: CONSTRUCTION OF EQUIPMENT SETS

The first phase of our search program develops the sets of equipment that can be made from individual equipment procurements already planned and contracted. The search state has two parts: the individual pieces of equipment that need to be put in sets and the sets of equipment generated to up to the current point. The search continues until no further sets can be made. Two approaches were taken to Phase I.

The first approach generated all possible partitions of the procured equipment into sets such that no further sets could be created. The program output the partition scheme with the least number of pieces of equipment not in a set. The results are shown in Chapter V. This method required a significant amount of recursion, memory storage, and CPU time. As the number of pieces of equipment procured in the program was increased, these factors increased exponentially.

The second approach which we now employ uses a branch-and-bound search technique similar to the second phase described later. This search technique generates only a few possible solutions for a list of equipment, one at a time. It takes a state on the agenda with the lowest cost value and computes its successors; this state is called the most promising intermediate state. The successors differ from the original state by the inclusion of one additional set of equipment that can be made from unassigned procurements in the state. The procurement totals are then correspondingly decremented. Items of equipment are assigned to sets until no further sets can be made. The cost function counts the equipment that is not in sets. Figure 1 shows the Prolog state-succession rule for this phase.

B. PHASE TWO: ALLOCATION OF EQUIPMENT SETS TO UNITS

The distribution plan is generated by a search which looks for an allocation of an equipment set to a unit of appropriate type and priority, giving the older set of equipment to another unit or disposing of it. The starting state contains a list of units, their types, and what equipment they currently have. States also hold a list of sets of equipment not yet allocated and when the sets will be available. A state transition means a set of equipment has been allocated to a unit.

```
successor([List_of_equipment,Setsofar],
          [Out_list_of_equipment,Newsetsofar]):-
    make_up_one_set(List_of_equipment,
          Out_list_of_equipment,Set,Year),
    appendd(procurement(Year,Set,1),Setsofar,Newsetsofar).
```

This successor rule for the unit set development phase takes components of the correct type and number from the procurement to make up one set. The set is then appended to the list of sets.

Figure 1

Successor Rule For Unit Set Development

Appendix B shows the state-succession rule for this phase. It shows how the unit information, units equipment status, and the projected procurements are used to make an equipment-allocation decision in the plan. To generate the successor states as fast as possible, predicates in the successor rule are ordered so that possible choices required are few after the first predicates succeed. So the successor rule first determines what type of set is next for the unit, and then if that set is available. It then determines if the equipment should be given to the unit based on the other equipment in other units, the priority of the unit, and the modernization level of the equipment.

Figure 2 shows the cost and evaluation function used by "nopathsearch" for this phase. The cost function sums up the times when each of the units were given their last set of equipment. Since the state-successor rule ensures that the equipment set is given to units of highest priority among those that can accept it, the state with the lowest total cost has used the equipment in the most efficient way among all valid possibilities. The evaluation function counts the equipment left to distribute.

We also explored a refinement of this phase that added an extra number to the cost. This number measured the time that equipment was available and not assigned to a unit. This measures how well the budget is used in relation to the actual needs of the units, a "delay factor" measurement.

```
cost(List,Amount):-
        bagof(U,unitstatus_look_year(List,U),Givelist),
        addit(Givelist,Total),member(delay(Time),List),
        dfactor(Weight), Amount is Total+Time*Weight.
```

The cost of a partial allocation is the sum of the last years that the equipment was given to a each unit. The best distribution plan gives equipment to the units at the earliest opportunity. The function "unitstatus_look_year" returns the last year that a unit was given equipment. The code above includes the delay factor that considered when equipment was available for the unit but no unit was able to receive it.

```
eval(List,Amount):-
        bagof(U,priorities(List,U),Prioritylist),
        minimum(Prioritylist,Amount).
```

The evaluation amount for a state is based on the priority of the equipment left to distribute to a unit. This predicate collects all of the modernization levels of equipment left to allocate and then returns the minimum. This underestimates the amount of additional time necessary until all of the equipment will be allocated.

Figure 2

Cost And Evaluation Function From Phase II

V. RESULTS

A. SYSTEM REQUIREMENTS

The Prolog program was developed with a C-PROLOG interpreter. To speed it up, the program was modified to be compiled with Quintus Prolog. Quintus requires certain declarations of predicates in a program to compile it. The Prolog software and the program required 2.5 megabytes. The program alone required 1.5 megabytes. All Prolog source code will run on a C-PROLOG interpreter once these declarations are removed.

B. TEST RUNS OF THE PROGRAMS

Table 1 shows some statistics on Phase I (developing unit sets from separate pieces of equipment). This phase developed unit sets within a small time compared to the search in Phase II.

Phase II was tested with various kinds of units and equipment. Procured equipment was chosen for each run so it would not modernize all units fully since this is usually the case with real budgeted assets. And also like real units, some units were at the intermediate level of modernization to start, but none were at the most modern level.

Table 2 summarizes the time used to produce plans for specific numbers of units, and it compares compiled and interpreted code with different delay factors. The time to develop a plan increased exponentially as the number of units and procurements increased.

TABLE 1

STATISTICS ON PHASE 1 OF THE PROGRAM

Number of Equipment Procurement Transactions	All-Partitons Method CPU Time (seconds)	Search Method CPU Time (seconds)
6	5.116	7.050
7	42.433	42.733
15	out of memory	57.166
22	out of memory	387.773

TABLE 2

STATISTICS ON PHASE 2 OF THE PROGRAM

CPU Time in Seconds

Number of Units	Interpreted Code	Compiled Code	Compiled Code Delay Factor=1	Compiled Code Delay Factor=2
2	52	57	53	53
3	172	116	85	128
4	1586	1152	1053	867
5	3434	2473	2340	2140
6	40072	30422	24126	20775
7	26793	20607	18086	16678
8	96625	77488	73615	9654
9	133018	114670	107911	98552
10	104830	83748	80199	78809

TABLE 3

COMPARISON OF CPU TIMES USING DIFFERENT DELAY FACTORS

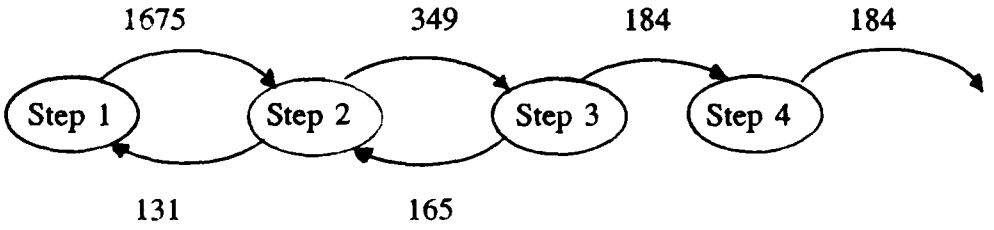
Delay Factor	Total CPU Time for Program (Seconds)
1	2340.733
2	2140.566
5	2125.316
10	2173.850

(for distributing equipment to five units)

Appendix C summarizes each of the test runs. The input was a list of units, their current equipment, and a list of procured equipment. The procured equipment was defined by a type, amount, and year procured.

Table 3 summarizes the time required with different delay weights, with five units and a limited amount of equipment. The inclusion of the delay factor in the cost function allowed only a savings of ten to twenty percent in the execution time from the original run of the program. The distribution plan found did not change.

Another analysis was performed in this thesis was to find bottlenecks in the state calculation in Phase II. A special version of the program was developed to trace this and is included in the Appendix C. Figure 3 shows a transition frequency diagram for the major parts of the successor rule. Table 4 shows the distribution of time it took to compute a successor.



Explanation of Each Step in the Transition Diagram

Step 1: Retrieve a Unit from the current state.

Step 2: Check that the amount of equipment in the procurement is enough to meet the needs of the Unit.

Step 3: Check if the proposed allocation violates the prioritization of the units.

Step 4: Give the set of equipment that the unit previously held to PDO or Procurement.

The upper numbers represent the frequency that a test succeeded; the lower numbers represent the frequency a test failed and backtracking was necessary.

Figure 3

State Movement Within The Successor Rule

TABLE 4

SUCCESSOR-COMPUTATION-TIME DISTRIBUTION

CPU Time To Compute Successor (1/100 of a second)	Frequency
300 - 500	18
501 - 1000	87
1001 - 1500	38
1501 - 2000	13
2001 - 2500	44
2501 - 3000	75
3001 - 3500	53
3501 - 4000	30
4001 - 4500	38
4501 - 5000	24
5001 - 5500	31
5501 - 6000	14
6001 - 6500	8
6501 - 7000	11
7001 - 7500	6
7501 - 8000	7
8001 - 8500	3
8501 - 9000	0
9001 - 9500	2
9501 -10000	1

This table is based on generating a plan for five units and a delay factor of two. The average time to compute successors was 2.981 seconds. The above numbers represent raw numbers of time a successor was generated within the program run.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. STRENGTHS AND WEAKNESSES

The Prolog programming language was a good choice for this research since it is widely available and does not constrain the programmer to a specific way of programming. The performance of the system improved with the compiler. The backtracking of the search allowed the system to consider a significant number of good alternatives and produce plans which were valid and reasonable. The components file in Appendix D showing the relationships of the equipment and modernization levels can be adapted to other types of problems.

Speed of the program does not seem a significant problem, anticipating forthcoming computer software and hardware improvements. The plans developed by the program are reasonable, timely, and correct while not guaranteed to be fully optimized.

A main weakness of this program is its criteria for choosing the state to generate a successor for, a function of the cost and successor functions. The width of the agenda could not be narrowed any further in this program since the system needed to be able to consider many options in order to get a good plan. But there is a critical tradeoff between the number of options considered and the speed of the final plan.

This prototype does not consider the distribution of equipment where the number of sets of equipment held in a unit is changed. This decision was made based on the short-term need to divide the system into separate phases. Otherwise, the system would constantly have to work with changing amounts of unit sets and compute unit sets prior to each allocation.

B. RECOMMENDATIONS

This system could be used as a reactionary or analytical tool for a planning agency. If the current plan needs to be changed due to equipment availability or unit realignment, the program could be rerun and a new plan could be put in place in only a few hours or days. A strength of computer-generated plans is that all decisions are based on well-documented information formalized in a program. No unknown biases or undocumented decisions can be introduced into the plan.

A enhanced prototype could be developed to work with the classified actual data. An additional feature that is necessary to better model the real world would be to require that sets of units that operate together use the same equipment. The planning of the amounts of equipment to procure was not addressed in this thesis. All these would be worthwhile areas of future study.

APPENDIX A: EQUIPMENT-SET EXAMPLES

The following examples are sets of equipment that actual units might have. Each example shows the items in the particular set, and the amount and type of equipment when appropriate. These sets are the standard ones.

11035J5 Division Signal Battalion with Atacs/Iatacs equipment

ITEM	AMOUNT	TYPE OF EQUIPMENT
AN/GRC-142	14	
AN/GSQ-80	4	
AN/MSC-31	6	
AN/TCC-65	4	
AN/TRC-113	12	Transmission
AN/TRC-145	42	Transmission
AN/TSC-58	2	
AN/TSC-76	7	Networking
AN/TSC-85	9	Transmission
AN/TSC-93	3	Transmission
AN/TTC-41	9	Switching

11045L0 Light Infantry Division Signal Battalion (specialized Equipment)

ITEM	AMOUNT	TYPE OF EQUIPMENT
SB-3614	1	Switching
AN/GRC-142	14	
AN/TTC-41	6	Switching
AN/TGC-30	2	
AN/TRC-145	31	Transmission
AN/TRC-113	6	Transmission
AN/TSC-76	4	Networking

11065L0 Heavy Division Signal Battalion
(Mobile Subscriber Equipment)

ITEM	AMOUNT	TYPE OF EQUIPMENT
AN/MSC-31	3	
AN/TRC-190	38	TRANSMISSION
AN/TRC-191	9	TRANSMISSION
AN/TTC-46	1	
AN/TTC-48	16	
AN/TTC-47	4	

11425H7 Corps Level Signal Battalion (Atacs)

ITEM	AMOUNT	TYPE OF EQUIPMENT
AN/GRC-142	2	
AN/GRC-122	31	
AN/MSC-25	1	
AN/MSC-31	3	
AN/TCC-72	5	SWITCHING
AN/TCC-73	8	SWITCHING
AN/TRC-112	5	TRANSMISSION
AN/TRC-121	5	TRANSMISSION
AN/TRC-151	20	TRANSMISSION
AN/TRC-152	34	TRANSMISSION
AN/TRC-170	6	TRANSMISSION
AN/TSC-76	1	NETWORKING
AN/TSC-85	3	TRANSMISSION
AN/TSC-93	1	TRANSMISSION

11435L0 Corps Level Signal Battalion (Iatacs)

ITEM	AMOUNT	TYPE OF EQUIPMENT
AN/MSC-31	1	
AN/TTC-46	1	
AN/TRC-190	72	TRANSMISSION
AN/TTC-47	6	
AN/TRC-191	13	TRANSMISSION
AN/TTC-48	40	
AN/TSQ-154	7	

EXAMPLE UNIT ALLOCATION

This allocation changes the unit into one that supports tactical satellite communications channels.

3rd Sig Bde

Date	Equipment	Amount
2 Qtr Fy 90	AN/GSQ-80	3
2 Qtr Fy 90	AN/MSC-32	1
1 Qtr Fy 90	AN/TYQ-35	2
3 Qtr Fy 88	AN/TSC-85	3
3 Qtr Fy 88	AN/TSC-93	3

APPENDIX B

TOP LEVEL RULE FOR DISTRIBUTION OF UNIT SETS TO UNITS

```
allocated(State,Newstate):-
 1. unit(NameU,PriorityU,TypeU),
 2. member(procurement(YearP,EquipP,AmountP),State),
 3. Amountp > 0,
 4. unitstatus_look_unit(State,NameU,YearU,EquipU),
 5. forward(EquipU,EquipP),
 6. amount(EquipP,TypeU,Required),
 7. AmountP>=Required,
 8. Temp is YearU+3,
 9. bigyear(YearP,Temp,Giveyear),
10. deleteone(procurement(YearP,EquipP,AmountP),State,X2),
11. NewamountP is AmountP-Required,
12. append_if_necessary(X2,X3,YearP,EquipP,NewamountP),
13. appendd(given(NameU,EquipP,Giveyear,Required),X3,X4),
14. not(any_wrong_priority(X4)),
15. member(given(NameU,EquipU,YearU,Oldamount),X4),
16. YY is Giveyear+1,
17. give_to_PDO_or_procurement(X4,NameU,EquipU,YY,Oldamount,X5),
18. get_rid_of_extraneous_proc(EquipU,X5,Newstate).
```

For clarity the time units were called years in the program. Capitalized items are variables in Prolog.

Variable	Explanation
State	Current state being examined
Newstate	Modified state after change has been made
NameU	Name of the unit being examined
PriorityU	Priority of UnitU
TypeU	Type of unit UnitU is
YearP	Year that the procurement will be made
EquipP	Name of the set of equipment in the procurement
AmountP	Amount of equipment being procured
EquipU	The current set of equipment UnitU has
Required	The amount of EquipP that a unit of TypeU needs
X2,X3,X4,X5	Temporary changes to the state in the rule

The successor rule states that an allocation of equipment can be made to a unit if the following conditions can be satisfied in this order.

1. There is a unit available (call it NameU).
2. There is still equipment to distribute.
3. The amount to distribute is > 0.
4. The set of equipment NameU currently has is EquipU and the time it received it is YearU.
5. The next modernization step after EquipU is EquipP.
6. The amount needed of EquipP for a unit of type TypeU is computed and called the variable Required.
7. The amount needed to fully modernize the unit NameU is sufficient.
8. A unit must keep an equipment set for at least 3 time periods, called years in the program for clarity; calculate the end of this period from when the old equipment was issued to the unit.
9. The unit can receive an equipment set only at or after the 3-year point.
10. Delete the equipment set from the list of available sets.
11. If there are still sets of EquipP left after modernizing NameU, calculate the amount.
12. Put the additional amount if any of EquipP back in the state for later allocation.
13. Add the allocation record to the state for unit NameU.
14. Determine if the allocation violates the priority for equipment allocation.
15. Determine the equipment the unit had prior to allocation.
16. The unit NameU only gives back equipment after it has both sets. This overlap of at least one time unit is necessary to continue the readiness posture of the unit.
17. Decide what to do with the old equipment. If a unit needs the equipment that unit NameU is giving up, the equipment is put in a procurement record. Otherwise it is given to Property Disposal (PDO).
18. Take care of equipment which is no longer needed.

There is an additional rule that combines procurements of the same type.

APPENDIX C
QUERIES AND RESULTS

This appendix gives Prolog top-level queries and results from the different runs of the Phase II, the allocation-of-equipment phase. The queries describe the units present and their status. The units each have a type of unit and a priority for new equipment. The state description has "given records" that describe the latest level of equipment modernization, the amount received, and the date it was given to the unit. The state also has "procurement records" which describe when equipment is available for distribution to a unit, the amount available, and the type of equipment.

The final state computed by Phase II used all the equipment available and disposed of equipment not needed. The latter is done with the "pdo record" of the final state, which indicates at the time specified all units had no use for the equipment at that time or any time in the future.

TWO UNITS

```
go_2c:- compile(nopathsearch),
        compile(components),
        compile(planner),
        assert(unit(sig5,1,s415)),
        assert(unit(sig3,3,s415)),
         done_searching_file([
                        given(sig5,iatacs,2,2),
                        given(sig3,atacs,3,1),
                        procurement(6,iatacs,1),
                        procurement(5,iatacs,1),
                        procurement(7,mse,2)],
           outplanner2c),halt.

[given(sig5,iatacs,2,2),given(sig5,mse,7,1),
given(sig3,atacs,3,1),given(sig3,iatacs,6,2),given(sig3,mse,9,1),

pdo(sig3,7,atacs,1),pdo(procurement,8,iatacs,2), pdo(sig3,10,iatacs,2)]
```

THREE UNITS

```
go_3c:- compile(nopathsearch),
        compile(components),
        compile(planner),
        assert(unit(sig5,1,s415)),
        assert(unit(sig2,2,s415)),
        assert(unit(sig3,3,s415)),
        done_searching_file([
                        given(sig5,iatacs,2,2),
                        given(sig2,atacs,3,1),
                        given(sig3,atacs,5,1),
                        procurement(5,iatacs,2),
                        procurement(7,mse,2),
                        procurement(9,mse,1)
                                        ],
         outplanner3c),halt.

[given(sig5,iatacs,2,2),given(sig5,mse,7,1),
given(sig2,atacs,3,1),given(sig2,iatacs,6,2),given(sig2,mse,9,1),
given(sig3,atacs,5,1),given(sig3,iatacs,8,2),given(sig3,mse,11,1),

pdo(sig2,7,atacs,1),pdo(sig3,9,atacs,1),pdo(sig2,10,iatacs,2), pdo(sig3,12,iatacs,2)]
```

FOUR UNITS

```
go_4c:- compile(nopathsearch),
        compile(components),
        compile(planner),
        assert(unit(sig5,1,s415)),
        assert(unit(sig4,2,s415)),
        assert(unit(sig2,4,s415)),
        assert(unit(sig3,3,s415)),
        done_searching_file([
                        given(sig5,iatacs,2,2),
                        given(sig3,atacs,3,1),
                        given(sig2,atacs,5,1),
                        given(sig4,atacs,3,1),
                        procurement(5,iatacs,3),
                        procurement(7,mse,2),
                        procurement(9,mse,2)
                            ],
          outplanner4c),halt.

[given(sig5,iatacs,2,2),given(sig5,mse,7,1),
given(sig4,atacs,3,1),given(sig4,iatacs,6,2),given(sig4,mse,9,1),
given(sig3,atacs,3,1),given(sig3,iatacs,8,2),given(sig3,mse,11,1),
given(sig2,atacs,5,1),given(sig2,iatacs,10,2),given(sig2,mse,13,1),

pdo(sig4,7,atacs,1),pdo(procurement,8,iatacs,1),
pdo(sig3,9,atacs,1),pdo(sig2,11,atacs,1),
pdo(procurement,12,iatacs,2),pdo(sig2,14,iatacs,2)]
```

FIVE UNITS

```
go_5c:- compile(nopathsearch),
        compile(components),
        compile(planner),
        assert(unit(sig5,1,s415)),
        assert(unit(sig4,2,s415)),
        assert(unit(sig2,4,s415)),
        assert(unit(sig3,3,s415)),
        assert(unit(sig1,5,s415)),
        done_searching_file([
                        given(sig5,iatacs,2,2),
                        given(sig3,atacs,4,1),
                        given(sig2,atacs,5,1),
                        given(sig4,atacs,4,1),
                        given(sig1,atacs,5,1),
                        procurement(5,iatacs,3),
                        procurement(7,mse,2),
                        procurement(9,mse,2)
                                ],
        outplanner5c),halt.


[given(sig5,iatacs,2,2),given(sig5,mse,7,1),
given(sig4,atacs,4,1),given(sig4,iatacs,7,2),given(sig4,mse,10,1),
given(sig3,atacs,4,1),given(sig3,iatacs,8,2),given(sig3,mse,11,1),
given(sig2,atacs,5,1),given(sig2,iatacs,11,2),given(sig2,mse,14,1),
given(sig1,atacs,5,1),given(sig1,iatacs,12,2),

pdo(procurement,5,iatacs,1),
pdo(sig4,8,atacs,1),pdo(sig3,9,atacs,1),pdo(sig2,12,atacs,1),
pdo(sig1,13,atacs,1),pdo(sig2,15,iatacs,2)]
```

SIX UNITS

```
go_6c:- compile(nopathsearch),
        compile(components),
        compile(planner),
        assert(unit(sig5,1,s415)),
        assert(unit(sig4,2,s415)),
        assert(unit(sig2,4,s415)),
        assert(unit(sig3,3,s415)),
        assert(unit(sig1,5,s415)),
        assert(unit(sig0,5,s415)),
        done_searching_file([
                        given(sig5,iatacs,2,2),
                        given(sig3,atacs,4,1),
                        given(sig2,atacs,5,1),
                        given(sig4,atacs,4,1),
                        given(sig1,atacs,5,1),
                        given(sig0,atacs,5,1),
                        procurement(5,iatacs,3),
                        procurement(7,mse,2),
                        procurement(7,iatacs,3),
                        procurement(9,mse,2)
                                ],
        outplanner6c),halt.
```

PLAN GENERATED

```
[given(sig5,iatacs,2,2),given(sig5,mse,7,1),
given(sig4,atacs,4,1),given(sig4,iatacs,7,2),given(sig4,mse,10,1),
given(sig3,atacs,4,1),given(sig3,iatacs,7,2),given(sig3,mse,10,1),
given(sig2,atacs,5,1),given(sig2,iatacs,8,2),given(sig2,mse,11,1),
given(sig0,atacs,5,1),given(sig1,atacs,5,1), given(sig0,iatacs,8,2),
given(sig1,iatacs,11,2),

pdo(sig4,8,atacs,1),pdo(sig3,8,atacs,1), pdo(sig2,9,atacs,1),
pdo(sig0,9,atacs,1),
pdo(procurement,11,iatacs,2), pdo(sig1,12,atacs,1),pdo(sig2,12,iatacs,2)]
```

SEVEN UNITS

```
go_7c:- compile(nopathsearch),
        compile(components),
        compile(planner),
        assert(unit(sig5,1,s415)),
        assert(unit(sig4,2,s415)),
        assert(unit(sig3,3,s415)),
        assert(unit(sig2,4,s415)),
        assert(unit(sig1,5,s415)),
        assert(unit(sig0,5,s415)),
        assert(unit(siga,6,s415)),
        done_searching_file([
                        given(sig5,iatacs,2,2),
                        given(sig4,atacs,2,1),
                        given(sig3,atacs,3,1),
                        given(sig2,atacs,5,1),
                        given(sig1,atacs,5,1),
                        given(sig0,atacs,5,1),
                        given(siga,atacs,5,1),
                        procurement(5,iatacs,4),
                        procurement(7,mse,2),
                        procurement(9,mse,2)
                                ],
        outplanner7c),halt.
```

PLAN GENERATED

```
given(sig5,iatacs,2,2),given(sig5,mse,7,1),
given(sig4,atacs,2,1),given(sig4,iatacs,5,2),given(sig4,mse,8,1),
given(sig3,atacs,3,1),given(sig3,iatacs,6,2),given(sig3,mse,9,1),
given(sig2,atacs,5,1),given(sig2,iatacs,8,2),given(sig2,mse,11,1),
given(sig0,atacs,5,1),given(sig1,atacs,5,1), given(sig0,iatacs,9,2),
given(sig1,iatacs,10,2),given(siga,atacs,5,1),given(siga,iatacs,12,2),

pdo(sig4,6,atacs,1),pdo(sig3,7,atacs,1),pdo(sig2,9,atacs,1),
pdo(sig0,10,atacs,1),pdo(sig1,11,atacs,1),
pdo(siga,13,atacs,1)]
```

EIGHT UNITS

```
go_8c:- compile(nopathsearch),
        compile(components),
        compile(planner),
        assert(unit(sig8,3,s415)),
        assert(unit(sig7,4,s415)),
        assert(unit(sig6,5,s415)),
        assert(unit(sig5,6,s415)),
        assert(unit(sig4,7,s415)),
        assert(unit(sig3,8,s415)),
        assert(unit(sig2,9,s415)),
        assert(unit(sig1,10,s415)),
        done_searching_file([
                        given(sig8,iatacs,3,2),
                        given(sig7,atacs,2,1),
                        given(sig6,atacs,2,1),
                        given(sig5,atacs,2,1),
                        given(sig4,atacs,4,1),
                        given(sig3,atacs,4,1),
                        given(sig2,atacs,5,1),
                        given(sig1,atacs,5,1),
                        procurement(8,iatacs,3), procurement(5,iatacs,3),
                        procurement(7,mse,2),    procurement(9,mse,2)
                            ],outplanner8c).
```

PLAN GENERATED

```
[given(sig8,iatacs,3,2),given(sig8,mse,7,1),
given(sig7,atacs,2,1),given(sig7,iatacs,5,2),given(sig7,mse,8,1),
given(sig6,atacs,2,1),given(sig6,iatacs,8,2),given(sig6,mse,11,1),
given(sig5,atacs,2,1),given(sig5,iatacs,8,2),given(sig5,mse,11,1),
given(sig4,atacs,4,1),given(sig4,iatacs,8,2),
given(sig3,atacs,4,1),given(sig3,iatacs,9,2),
given(sig2,atacs,5,1),given(sig2,iatacs,12,2),
given(sig1,atacs,5,1),given(sig1,iatacs,12,2),

pdo(sig7,6,atacs,1),pdo(sig6,9,atacs,1),pdo(sig5,9,atacs,1),
pdo(sig4,9,atacs,1),pdo(sig3,10,atacs,1),pdo(sig2,13,atacs,1), pdo(sig1,13,atacs,1)]
```

NINE UNITS

```
go_9c:- compile(nopathsearch),
        compile(components),
        compile(planner),
        assert(unit(sig9,2,s415)), assert(unit(sig8,3,s415)),
        assert(unit(sig7,4,s415)), assert(unit(sig6,5,s415)),
        assert(unit(sig5,6,s415)), assert(unit(sig4,7,s415)),
        assert(unit(sig3,8,s415)), assert(unit(sig2,9,s415)),
        assert(unit(sig1,10,s415)),
        done_searching_file([
                    given(sig9,iatacs,2,2), given(sig8,iatacs,3,2),
                    given(sig7,atacs,2,1),  given(sig6,atacs,2,1),
                    given(sig5,atacs,2,1),  given(sig4,atacs,4,1),
                    given(sig3,atacs,4,1),  given(sig2,atacs,5,1),
                    given(sig1,atacs,5,1),
                    procurement(8,iatacs,3),  procurement(5,iatacs,3),
                    procurement(7,mse,2),     procurement(9,mse,2)
                      ],outplanner9c).
```

PLAN GENERATED

```
[given(sig9,iatacs,2,2),given(sig9,mse,7,1),
given(sig8,iatacs,3,2),given(sig8,mse,7,1),
given(sig7,atacs,2,1),given(sig7,iatacs,5,2),given(sig7,mse,9,1),
given(sig6,atacs,2,1),given(sig6,iatacs,8,2),given(sig6,mse,11,1),
given(sig5,atacs,2,1),given(sig5,iatacs,8,2),
given(sig4,atacs,4,1),given(sig4,iatacs,8,2),
given(sig3,atacs,4,1),given(sig3,iatacs,8,2),
given(sig2,atacs,5,1),given(sig2,iatacs,10,2),
given(sig1,atacs,5,1),given(sig1,iatacs,12,2),

pdo(sig7,6,atacs,1),pdo(sig6,9,atacs,1),
pdo(sig5,9,atacs,1),pdo(sig4,9,atacs,1),
pdo(sig3,9,atacs,1),pdo(sig2,11,atacs,1),pdo(sig1,13,atacs,1)]
```

TEN UNITS

```
go_10c:- compile(nopathsearch),
        compile(components),
        compile(planner),
        assert(unit(sig10,1,s415)),  assert(unit(sig9,2,s415)),
        assert(unit(sig8,3,s415)),   assert(unit(sig7,4,s415)),
        assert(unit(sig6,5,s415)),   assert(unit(sig5,6,s415)),
        assert(unit(sig4,7,s415)),   assert(unit(sig3,8,s415)),
        assert(unit(sig2,9,s415)),   assert(unit(sig1,10,s415)),
        done_searching_file([given(sig10,iatacs,2,2),  given(sig9,iatacs,2,2),
                             given(sig8,iatacs,3,2),   given(sig7,atacs,2,1),
                             given(sig6,atacs,2,1),    given(sig5,atacs,2,1),
                             given(sig4,atacs,4,1),    given(sig3,atacs,4,1),
                             given(sig2,atacs,5,1),    given(sig1,atacs,5,1),
                             procurement(8,iatacs,3),  procurement(5,iatacs,3),
                             procurement(7,mse,2),     procurement(9,mse,2)
                             ],outplanner10c).
```

PLAN GENERATED

```
[given(sig10,iatacs,2,2),given(sig10,mse,7,1),
given(sig9,iatacs,2,2),given(sig9,mse,7,1),
given(sig8,iatacs,3,2),given(sig8,mse,9,1),
given(sig7,atacs,2,1),given(sig7,iatacs,5,2),given(sig7,mse,9,1),
given(sig6,atacs,2,1),given(sig6,iatacs,8,2),
given(sig5,atacs,2,1),given(sig5,iatacs,8,2),
given(sig4,atacs,4,1),given(sig4,iatacs,8,2),
given(sig3,atacs,4,1),given(sig3,iatacs,8,2),
given(sig2,atacs,5,1),given(sig2,iatacs,10,2),
given(sig1,atacs,5,1),given(sig1,iatacs,10,2),

pdo(sig7,6,atacs,1),pdo(sig6,9,atacs,1),pdo(sig5,9,atacs,1),
pdo(sig4,9,atacs,1),pdo(sig3,9,atacs,1),pdo(sig2,11,atacs,1),
pdo(sig1,11,atacs,1)]
```

APPENDIX D: SOURCE CODE

```
/*   module procsearch */

/*   This is the module that generates unit sets of equipment  */

/*   from individual procurements of equipment                 */

/*   The goal of the search is a state where no other sets of
equipment  */
/*  can be made from the components can be made
      */

goalreached([[],X]):-write(X),nl,statistics.
goalreached([X,Y]):-not(make_up_one_set(X,Output,Set,Year)),
     write('-----------'),nl,write(X),nl,write(Y),nl,statistics.

/* the cost of a state is the amount of equipment not yet in sets
 */
cost([X,Y],Cost):-countit(X,Cost).

countit([],0).
countit([X|L],Y):-countit(L,Y2),Y is Y2+1.

eval(X,1).

/*  this is the top level call for this module which calls the
search function*/
done_searching_file(X,Fname):-save(coresearcher),
     write('core image saved'),nl,
    tell(Fname),search([X,[]],Y),told,halt.

successor([List_comp_proc,Setsofar],
     [Out_list_comp_proc,Newsetsofar]):-

make_up_one_set(List_comp_proc,Out_list_comp_proc,Set,Year),
      appendd(procurement(Year,Set,1),Setsofar,Newsetsofar).


/* no more sets of possibilities to make up  */

make_up_one_set(List_of_components,Rest_of_components,
     NameG,Year):-
     generation(NameG),
     bagof([Cname,Camount],component(NameG,Cname,Camount),Clist),
     makeit(List_of_components,Rest_of_components,Clist,0,Year).

makeit(Procured_components_left,Procured_components_left,
     [],Year,Year).
```

```
makeit(Procured_components,Rest_of_components_not_used,
     Needed_components, Latest_year,New_latest_year):-
          firstmember(Needed_components,[NameC,AmountC]),
   member(procurement(Year,NameC,AmountP),Procured_components),
take_care_of_lists(Year,AmountP,AmountC,NameC,Needed_components,
New_needed,Procured_components,New_rest_of_components),
          bigyear(Year,Latest_year,Tyear),
makeit(New_rest_of_components,Rest_of_components_not_used,
                New_needed,Tyear,New_latest_year).

take_care_of_lists(Year,AmountP,AmountC,NameC,Needed_components,

New_needed,Procured_components,New_rest_of_components):-
          AmountP>AmountC,
          deleteone([NameC,AmountC],Needed_components,New_needed),
          T is AmountP-AmountC,

deleteone(procurement(Year,NameC,AmountP),Procured_components,
              New_procured),
          appendd(procurement(Year,NameC,T),New_procured,
              New_rest_of_components).

take_care_of_lists(Year,AmountP,AmountC,NameC,Needed_components,

New_needed,Procured_components,New_rest_of_components):-
          AmountP<AmountC,

deleteone([NameC,AmountC],Needed_components,New_needed1),
          T is AmountC-AmountP,

deleteone(procurement(Year,NameC,AmountP),Procured_components,
              New_procured),
          appendd([NameC,T],New_needed1,New_needed).


take_care_of_lists(Year,AmountP,AmountC,NameC,Needed_components,

New_needed,Procured_components,New_rest_of_components):-
          AmountP=AmountC,
          deleteone([NameC,AmountC],Needed_components,New_needed),


deleteone(procurement(Year,NameC,AmountP),Procured_components,
              New_rest_of_components).
```

```
/* the following code does a sort of the list for comparisons and
output   */

sortpplist([],[]).
sortpplist([X|L1],L2):-
     sortpplist(L1,L3),
     insert_item_for_sort(X,L3,L2).

insert_item_for_sort(X,[],[X]).
insert_item_for_sort(procurement(XYear,XName,XAmount),
      [procurement(YYear,YName,YAmount)|L],
     [procurement(XYear,XName,XAmount),
     procurement(YYear,YName,YAmount)|L]):-
           priority(XName,X),
           priority(YName,Y),
           XX is X+XYear*1000,
           YY is Y+YYear*1000,
           XX<YY.
insert_item_for_sort(procurement(XYear,XName,XAmount),
      [procurement(YYear,YName,YAmount)|L1],
      [procurement(YYear,YName,YAmount)|L2]):-
           priority(XName,X),
           priority(YName,Y),
           XX is X+XYear*1000,
           YY is Y+YYear*1000,
           XX>=YY,

insert_item_for_sort(procurement(XYear,XName,XAmount),L1,L2).
```

```
/*  general purpose utilities needed in the procsearc module   */

prunable(State,D,BestState,Dbest):-
      check_permutation(State,BestState),!.

firstmember([X|L],X).

bigyear(Inyear,Year,Outyear):- Year >=Inyear, Outyear is Year.
bigyear(Inyear,Year,Outyear):- Year <Inyear, Outyear is Inyear.

minimum([X],X).
minimum([X|L],X):-minimum(L,X2),X<X2.
minimum([X|L],X2):-minimum(L,X2),not(X<X2).

max([X],X).
max([X|L],X):-max(L,X2),X>X2.
max([X|L],X2):-max(L,X2),not(X>X2).

member(X,[X|L]).
member(X,[Y|L]):- member(X,L).

deleteone(X,[X|L],L).
deleteone(X,[Y|L],[Y|Z]):- deleteone(X,L,Z).

appendd(Element,List,[Element|List]).
```

```
/*   module proc-gen */

/*  this module was the first alternative method of generating
unit sets*/
/*  of equipment.  This module enumerates all of the combinations
of unit */
/*  sets of equipment and then returns the one with the most sets
of equipment */

make_up_possibilities_no_dups_file(Input,Fname):-
    save(coreprocgen),
    write('core saved ctl c now'),nl,nl,
    setof(U,multiple_sets_of_proc(Input,U),X),
    remove_sublists(X,Z),
    tell(Fname),
    write(Z),statistics,told.

multiple_sets_of_proc(Input,X):-
          have_them_all(Input,[],Sets),
          tell(workproc),write(Sets),told,
          sortpplist(Sets,X).

have_them_all([],X,X).

have_them_all(List_comp_proc,Setsofar,Newsets):-

make_up_one_set(List_comp_proc,Out_list_comp_proc,Set,Year),
      appendd(procurement(Year,Set,1),Setsofar,Newsetsofar),
      have_them_all(Out_list_comp_proc,Newsetsofar,Newsets).

have_them_all(Y,X,X).

make_up_one_set(List_of_components,Rest_of_components,
     NameG,Year):-
     generation(NameG),
     bagof([Cname,Camount],component(NameG,Cname,Camount),Clist),
     makeit(List_of_components,Rest_of_components,Clist,0,Year).

makeit(Procured_components_left,Procured_components_left,
     [],Year,Year).
```

```
makeit(Procured_components,Rest_of_components_not_used,
     Needed_components,
       Latest_year,New_latest_year):-
         firstmember(Needed_components,[NameC,AmountC]),
     member(procurement(Year,NameC,AmountP),
     Procured_components),
     take_care_of_lists(Year,AmountP,AmountC,NameC,
     Needed_components,
     New_needed,Procured_components,New_rest_of_components),
         bigyear(Year,Latest_year,Tyear),
     makeit(New_rest_of_components,Rest_of_components_not_used,
                New_needed,Tyear,New_latest_year).

take_care_of_lists(Year,AmountP,AmountC,NameC,Needed_components,

New_needed,Procured_components,New_rest_of_components):-
          AmountP>AmountC,
          deleteone([NameC,AmountC],Needed_components,New_needed),
          T is AmountP-AmountC,
           deleteone(procurement(Year,NameC,AmountP),
           Procured_components,
              New_procured),
          appendd(procurement(Year,NameC,T),New_procured,
              New_rest_of_components).

take_care_of_lists(Year,AmountP,AmountC,NameC,Needed_components,
           New_needed,Procured_components,
           New_rest_of_components):-
           AmountP<AmountC,
         deleteone([NameC,AmountC],Needed_components,New_needed1),
          T is AmountC-AmountP,
           deleteone(procurement(Year,NameC,AmountP),
           Procured_components,
              New_procured),
          appendd([NameC,T],New_needed1,New_needed).


take_care_of_lists(Year,AmountP,AmountC,NameC,Needed_components,
     New_needed,Procured_components,New_rest_of_components):-
          AmountP=AmountC,
          deleteone([NameC,AmountC],Needed_components,New_needed),
         deleteone(procurement(Year,NameC,AmountP),
           Procured_components,
              New_rest_of_components).
```

```
sortpplist([],[]).
sortpplist([X|L1],L2):-
     sortpplist(L1,L3),
     insert_item_for_sort(X,L3,L2).

insert_item_for_sort(X,[],[X]).
insert_item_for_sort(procurement(XYear,XName,XAmount),
     [procurement(YYear,YName,YAmount)|L],
     [procurement(XYear,XName,XAmount),
          procurement(YYear,YName,YAmount)|L]):-
           priority(XName,X),
           priority(YName,Y),
           XX is X+XYear*1000,
           YY is Y+YYear*1000,
           XX<YY.
insert_item_for_sort(procurement(XYear,XName,XAmount),
      [procurement(YYear,YName,YAmount)|L1],
      [procurement(YYear,YName,YAmount)|L2]):-
           priority(XName,X),
           priority(YName,Y),
           XX is X+XYear*1000,
           YY is Y+YYear*1000,
           XX>=YY,

insert_item_for_sort(procurement(XYear,XName,XAmount),L1,L2).
```

```
remove_sublists(Y,Z):- get_rid_of_sublist(Y,Z).

get_rid_of_sublist(Y,Y):- not(delete_sublist(Y,Y1)).

get_rid_of_sublist(Y,Z):-    delete_sublist(Y,Y1),
get_rid_of_sublist(Y1,Z).

delete_sublist(Y,Ynew):-
       member(X,Y),
       member(X1,Y),
       not(X=X1),
       length_of_list(X,Lx),
       length_of_list(X1,Lx1),
       Lx<Lx1,
       sublist(X,X1),
       deleteone(X,Y,Ynew).

sublist([],X).
sublist(X,X1):-  member(E,X),member(E,X1), deleteone(E,X,Xnew),
      sublist(Xnew,X1).

 /*   General purpose utilities used in this module */

subset([],L).
subset([X|L],L2):-singlemember(X,L2),subset(L,L2).

firstmember([X|L],X).

bigyear(Inyear,Year,Outyear):- Year >=Inyear, Outyear is Year.
bigyear(Inyear,Year,Outyear):- Year <Inyear, Outyear is Inyear.

minyear([procurement(Year,Equip,Amount)],Year,Amount).
minyear([procurement(Year,Equip,Amount)|L],Year,Amount):-
           minyear(L,Y,A),Year<Y.
minyear([procurement(Year,Equip,Amount)|L],Y,A):-
           minyear(L,Y,A),not(Year<Y).

minimum([X],X).
minimum([X|L],X):-minimum(L,X2),X<X2.
minimum([X|L],X2):-minimum(L,X2),not(X<X2).

member(X,[X|L]).
member(X,[Y|L]):- member(X,L).

deleteone(X,[X|L],L).
deleteone(X,[Y|L],[Y|Z]):- deleteone(X,L,Z).

appendd(Element,List,[Element|List]).

length_of_list([X],1).
length_of_list([X|L],I):-length_of_list(L,I1),I is I1+1.
```

```
/*   module components */
/*   This module defines the parameters for the program.
/*     It defines the following */
/*          1.  Equipment in a set of equipment    */
/*          2.  The modernization path for types of units  */
/*           3.  A numeric sequence for equipment used in sorted
output  */


priority(msediv,1).                priority(iatacsdiv,2).
priority(atacsdiv,3).

forward(atacs,iatacs). forward(iatacs,mse).

generation(atacs).   generation(iatacs).
generation(mse).

amount(mse,s615,1).
amount(iatacs,s615,1).
amount(atacs,s615,1).

/* priority of components for sorts  */
priority(trc145,100).
priority(trc113,101).
priority(trc115,102).
priority(ttc29,103).
priority(ttc39,104).
priority(ttc41,105).
priority(trc152,106).
priority(trc153,107).
priority(trc154,108).

/* list of components    */

component(atacs,trc145,1).
component(atacs,ttc29,1).
component(atacs,trc113,1).
component(iatacs,trc115,1).
component(iatacs,ttc39,1).
component(iatacs,trc152,1).
component(mse,ttc41,1).
component(mse,trc153,1).
component(mse,trc154,1).
```

```
/* module nopathsearch     */
/* This implements search with an evaluation and cost function,
*/
/* but without keeping of path lists.  Answer returned is the */
/*  final state.   This can save much space, but the solutions
found */
/*  are not guaranteed optimal (nor will typing semicolons
necessarily */
/* help).  For an application, you must define 6 predicates:   */

/*    (a) successor(State,Newstate)  (gives state transitions) */
/*    (b) goalreached(State)  (defines when goal achieved)  */
/*    (c) eval(State,Evaluation) (estimates cost to the goal) */
/*     (d) cost(State,Cost) (computes cost to a state from its
description)   */
/*            (NOTE PRECEDING IS DIFFERENT FROM A*--PATHLIST NOT
USED)   */
/*      (e)  prunable(State,DState,Beststate,DBeststate)  (defines
conditions  */
/*         for pruning State from the agenda, given Beststate is
the best */
/*       state on the agenda according to D=cost+eval (optional))
*/
/*    (f) a top-level predicate that initializes things if needed
and */
/*        then calls astarsearch with two arguments, the starting
*/
/*       state and variable which will be the solution path. */
/* Note: "cost" must be nonnegative.  The "eval" should be a
lower bound */
/* on cost in order for the first answer found to be guaranteed
optimal. */


/* this progrm has the declarations for quintus prolog   */

  :- dynamic beststate/2, agenda/3, usedstate/2, counter/1.

search(Start,Ans) :- nopathsearch(Start,Ans).

nopathsearch(Start,State) :- cleandatabase, add_state(Start),
  repeatifagenda, pick_best_state(State),
  add_successors(State), agenda(State,C,D),
  retract(agenda(State,C,D)), measurework.

pick_best_state(State) :- asserta(beststate(dummy,dummy)),
  agenda(S,C,D), beststate(S2,D2), special_less_than(D,D2),
  retract(beststate(S2,D2)), asserta(beststate(S,D)), fail.
pick_best_state(State)   :-   beststate(S,D),   agenda(S2,C2,D2),
not(S=S2),
  prunable(S2,D2,S,D), retract(agenda(S2,C2,D2)), fail.
```

```
pick_best_state(State) :- beststate(State,D),
  retract(beststate(State,D)), not(D=dummy), !.

add_successors(State) :- goalreached(State), !.
add_successors(State)    :-    successor(State,Newstate),
add_state(Newstate), fail.
add_successors(State) :- retract(agenda(State,C,D)),
  asserta(usedstate(State,C)), fail.

add_state(Newstate)    :-    cost(Newstate,Cnew),    !,
agenda_check(Newstate,Cnew),
  !, usedstate_check(Newstate,Cnew), !, eval(Newstate,Enew), D is
Enew + Cnew,
  asserta(agenda(Newstate,Cnew,D)), !.
add_state(Newstate) :- not(cost(Newstate,Cnew)),
  write('Warning: your cost function failed on path list '), nl,
!.
add_state(Newstate) :- not(eval(Newstate,Enew)),
  write('Warning: your evaluation function failed on state '),
  write(Newstate), nl, !.

agenda_check(S,C)    :-    agenda(S,C2,D2),    C<C2,
retract(agenda(S,C2,D2)), !.
agenda_check(S,C) :- agenda(S,C2,D2), !, fail.
agenda_check(S,C).

usedstate_check(S,C)    :-    usedstate(S,C2),    C<C2,
retract(usedstate(S,C2)),
  asserta(usedstate(S,C)), !.
usedstate_check(S,C).

repeatifagenda.
repeatifagenda :- agenda(X,Z,W), repeatifagenda.

special_less_than(X,dummy) :- !.
special_less_than(X,Y) :- X<Y.

cleandatabase    :-    checkabolish(agenda,3),
checkabolish(usedstate,2),
  checkabolish(beststate,2), checkabolish(counter,1).

checkabolish(P,N) :- abolish(P,N), !.
checkabolish(P,N).

measurework    :-    countup(agenda(X,C,D),NA),
countup(usedstate(S,C),NB),
  tell(outstates),write(NA),write('    incompletely    examined
state(s) and '),
    write(NB),write('    examined
state(s)'),nl,print_agenda,print_usedstate,
   told,!.
```

```
countup(P,N)   :-   asserta(counter(0)),   call(P),   counter(K),
retract(counter(K)),
  K2 is K+1, asserta(counter(K2)), fail.
countup(P,N) :- counter(N), retract(counter(N)), !.

append([],L,L).
append([I|L1],L2,[I|L3]) :- append(L1,L2,L3).

print_agenda:-
       write('agenda'),nl,agenda(X,Y,Z),
       write(X),write(' '),write(Y),write('  '),write(Z),nl,fail.

print_agenda.

print_usedstate:-
       write('usedstates'),nl,usedstate(X,Y),
       write(X),write(Y),nl,fail.
print_usedstate.
```

```
/*   module planner */

/* this is a protytype plannng program   */
/* it comes up with a distribution but does not consider the  */
/* priority of the units or the fact that distribution must   */
/* modernize the units as soon as possible                    */
/*  This version of the planning program does not have the delay
factor  */
/*  being used in it   */

:-dynamic mytrace/3.
:-dynamic variable/1.

mytrace(allocated,0,0).
variable(1).

/* this traces how long a successor takes  if the tracer flag is
asserted */

successor(X,Y):-tracer,statistics(runtime,CG),
                allocated(X,Y),

statistics(runtime,Time),retract(variable(Count)),
                Counter is Count+1, asserta(variable(Counter)),
                asserta(mytrace(allocated,Counter,Time)).

 successor(X,Y):- not(tracer),allocated(X,Y).

/* allocate for procurement after unit is available */
allocated(X1,XNew2) :-
        unit(NameU,PriorityU,TypeU),
        member(procurement(YearP,EquipP,AmountP),X1),
        AmountP>0,
        unitstatus_look_unit(X1,NameU,YearU,EquipU),
        forward(EquipU,EquipP),
        amount(EquipP,TypeU,Required),
        AmountP >= Required,
        Temp is YearU+3,
        bigyear(YearP,Temp,Giveyear),
        deleteone(procurement(YearP,EquipP,AmountP),X1,X2),
        NewamountP is AmountP - Required,
        append_if_necessary(X2,X3,YearP,EquipP,NewamountP),
        appendd(given(NameU,EquipP,Giveyear,Required),X3,X4),
        not(any_wrong_priority(X4)),
        member(given(NameU,EquipU,YearU,Oldamount),X4),
        YY is Giveyear +1,

give_to_PDO_or_procurement(X4,NameU,EquipU,YY,Oldamount,X5),
        get_rid_of_extraneous_proc(EquipU,X5,XNew2).
```

```
/* when more than one procurement is necessary for the unit  */

allocated(X1,XNew2) :-
        unit(NameU,PriorityU,TypeU),
        unitstatus_look_unit(X1,NameU,YearU,EquipU),
        forward(EquipU,EquipP),
        amount(EquipP,TypeU,Required),
        bagof(P,member(procurement(_,EquipP,P),X1),Plist),
        addit(Plist,AmountP),
        AmountP >= Required,
        bagof(Q,proc_for_equip(EquipP,X1,Q),Pelist),
        delete_all_e(EquipP,X1,X2),
        get_enough(Pelist,Required,Outlist,Tyear),
        Temp is YearU+3,  bigyear(Tyear,Temp,Outyear),
        concat(Outlist,X2,X3),
        appendd(given(NameU,EquipP,Outyear,Required),X3,X4),
        not(any_wrong_priority(X4)),
        member(given(NameU,EquipU,YearU,Oldamount),X4),
        YY is Outyear +1,

give_to_PDO_or_procurement(X4,NameU,EquipU,YY,Oldamount,X7),
        get_rid_of_0_procurements(X7,X8),
        get_rid_of_extraneous_proc(EquipU,X8,XNew2).

get_rid_of_extraneous_proc(Equip,X,Y):-
                member(procurement(Year,Equip,Amount),X),
                not(anybody_at_lower_levels(X,Equip)),
                deleteone(procurement(Year,Equip,Amount),X,X1),
                appendd(pdo(procurement,Year,Equip,Amount),X1,Y).


get_rid_of_extraneous_proc(Equip,X,X).

anybody_at_lower_levels(List,Equip):-
                forward(X,Equip),
                unitstatus_look_equip(List,X).

anybody_at_lower_levels(List,Equip):-
                forward(X,Equip),
                anybody_at_lower_levels(List,X).

append_if_necessary(List,Newlist,YearP,EquipP,NewamountP):-
      NewamountP>0,

appendd(procurement(YearP,EquipP,NewamountP),List,Newlist).

append_if_necessary(List,List,YearP,EquipP,NewamountP):-
      NewamountP=0.
```

```
unitstatus_look_year(List,Year):- unit(Name,_,_),
        unitstatus_look_unit(List,Name,Year,E1).

unitstatus_look_unit(List,Name,Year,Equip):-
        bagof([YY,EE],look_for_given(List,Name,YY,EE),U),
        maxequip(U,Year,Equip).

look_for_given(List,Name,Year,Equip):-
      member(given(Name,Equip,Year,Amount),List).
     unitstatus_look_equip(List,Equip):- unit(Name,_,_),
     unitstatus_look_unit(List,Name,Year,E1),Equip=E1.

maxequip([[YY,EE]],YY,EE).
maxequip([[YY,EE]|L],YY,EE):-maxequip(L,Y1,E1),YY>Y1.
maxequip([[YY,EE]|L],Y1,E1):-maxequip(L,Y1,E1),not(YY>Y1).

proc_for_equip(EquipP,X1,procurement(Year,EquipP,Amount)):-
      member(procurement(Year,EquipP,Amount),X1).

/* when the amount in the minimum procurement is the correct
amount */
get_enough(List,Required,Outlist,Year):-
     Required>0, minyear(List,Tyear,Amount),
     T is Required - Amount,
     T=0, deleteone(procurement(Tyear,X,Amount),List,Outlist),
     Year is Tyear.

/* when the amount in min procurement is not enough   */
get_enough(List,Required,Outlist,Year):-
     Required>0,  minyear(List,Tyear,Amount),
     T is Required - Amount, T>0,
     deleteone(procurement(Tyear,X,Amount),List,Toutlist),
     get_enough(Toutlist,T,Outlist,Year).

/* when the amount in the min procurement is more than enough */
get_enough(List,Required,Outlist,Year):-
     Required>0, minyear(List,Year,Amount),
     T is Required - Amount,
     T<0, deleteone(procurement(Year,X,Amount),List,Toutlist),
     Excess is Amount- Required,
     appendd(procurement(Year,X,Excess),Toutlist,Outlist).
```

```
 /* this rule makes sure units with different priority */
/* have different priority equipment                    */

any_wrong_priority(List):-
     unit(N1,P1,_),
     unit(N2,P2,_),
     not(N1=N2),
     unitstatus_look_unit(List,N1,Y1,E1),
     unitstatus_look_unit(List,N2,Y2,E2),
     priority(E1,PE1),
     priority(E2,PE2),
     priority_is_bad(E1,E2,Y1,Y2,P1,P2,PE1,PE2).

/* these are the two cases of non valid priorities  */
priority_is_bad(E1,E2,Y1,Y2,P1,P2,PE1,PE2):-
     E1=E2,
     P1<P2,
     Y1>Y2.

priority_is_bad(E1,E2,Y1,Y2,P1,P2,PE1,PE2):-
     not(E1=E2),
     P1<P2,
     PE1>PE2.

give_to_PDO_or_procurement(O_state,Name,Equip,Year,
     Amount,N_state):-
            check_for_a_unit(O_state,Equip),

appendd(procurement(Year,Equip,Amount),O_state,N_state).

check_for_a_unit(State,Equip):- forward(O_equip,Equip),
            unitstatus_look_equip(State,O_equip).

check_for_a_unit(State,Equip):- forward(O_equip,Equip),
            check_for_a_unit(State,O_equip).

 /* this predicate works when a unit doesnt need the equipment
you are giving */
/*    and there is a previous generation   */

give_to_PDO_or_procurement(Old_state,Name,Equip,Year,
     Amount,New_State):-
            forward(Old_equip,Equip),
            not(unitstatus_look_equip(Old_state,Old_equip)),

get_rid_of_remaining_procurements(Equip,Old_state,Old_state1),

appendd(pdo(Name,Year,Equip,Amount),Old_state1,New_State).
```

```
/* this works when a unit does not need the equipment you are
giving  */

give_to_PDO_or_procurement(Old_state,Name,Equip,Year,
     Amount,New_State):-
            not(forward(Old_equip,Equip)),

get_rid_of_remaining_procurements(Equip,Old_state,Old_state1),

appendd(pdo(Name,Year,Equip,Amount),Old_state1,New_State).

get_rid_of_remaining_procurements(X,[],[]).
get_rid_of_remaining_procurements(X,
          [procurement(Year,X,Amount)|L],
          [pdo(procurement,Year,X,Amount)|LL]):-
         get_rid_of_remaining_procurements(X,L,LL).
get_rid_of_remaining_procurements(X,
          [Transaction|L],[Transaction|LL]):-
         get_rid_of_remaining_procurements(X,L,LL)  .

get_rid_of_0_procurements(Old_state, New_state):-
    deleteall(Old_state,New_state).

/* the evaluation function calculates the least number of */
/* givens needed to get a unit to the highest procurement  */

eval(List,Z):-bagof(U,priorities(List,U),Prioritylist),
               minimum(Prioritylist,Z).

priorities(List,Return):- unit(Name,_,_),
unitstatus_look_unit(List,Name,_,Equip),
priority(Equip,Return).

/* the cost function is the number of time periods the plan lasts
*/

cost(List,Z):-bagof(U,unitstatus_look_year(List,U),Givelist),
                addit(Givelist,Z).

/* define the goal of the program     */

goalreached(X):-  get_rid_of_0_procurements(X, New_state),
                  subgoal(New_state).
subgoal(New_state):- not(member(procurement(_,_,_),New_state)).
subgoal(New_state):- not(allocated(New_state,X)).
```

```
/* this is a format for the program   */
done_searching_file(List,Fname):-any_wrong_priority(List),
   tell(Fname),write('it cannot be done'),nl,
        write('there is a problem with the units data'),nl,
        write('equipoment already out of priority sequence'),nl,
               told,halt.

done_searching_file(List,Fname):-
    not(tracer),
    save(coreplanner),
     write('core image saved you can ctl c now'),nl,
    search(List,X),sortplan(X,Y),tell(Fname),write(Y),statistics,
    countup(agenda(A,B,C),NA),write(NA),write(':incomplete '),
    countup(usedstate(S,SS),NB),write(NB),write(':examined '),
    told,halt.

done_searching_file(List,Fname):-
    tracer,
    save(coreplanner),
     write('core image saved you can ctl c now'),nl,
    search(List,X),sortplan(X,Y),
     tell(Fname),write(Y),statistics,
    countup(agenda(A,B,C),NA),write(NA),write(':incomplete '),
    countup(usedstate(S,SS),NB),write(NB),write(':examined '),nl,
    print_mytrace, told,halt.

print_mytrace:-nl,
     write("trace of execution times of successor calculations"),
     nl,mytrace(A,B,C),printone(A,B,C),nl,fail.
print_mytrace.

printone(A,B,C):-write(A),write("   "),
     write(B),write("  "),write(C).

/* this is the fall through if it cannot do the search */
done_searching_file(List,Fname):-  tell(Fname),
            write('it cannot be done'),told,halt.

sortplan([],[]).
sortplan([X|L1],L2):-sortplan(L1,L3),insert_in_plan(X,L3,L2).

insert_in_plan(X,[],[X]).
insert_in_plan(X,[Y|L],[X,Y|L]):- cost_in_plan(X,Xcost),
            cost_in_plan(Y,Ycost), Xcost<Ycost.
insert_in_plan(X,[Y|L1],[Y|L2]):- cost_in_plan(X,Xcost),
            cost_in_plan(Y,Ycost), Xcost>=Ycost,
            insert_in_plan(X,L1,L2).

cost_in_plan(given(Unit,_,Year,_),Cost):- unit(Unit,Priority,_),
          Cost is Priority*100+Year.
cost_in_plan(pdo(Unit,Year,_,_),Cost):- Cost is 1000+Year*10.
```

```
/*  General Purpose Utilities for the module */

prunable(State,D,BestState,Dbest):-
      check_permutation(State,BestState),!.

check_permutation(State1,State2):-
      subset(State1,State2),
      subset(State2,State1),!.

subset([],L).
subset([X|L],L2):-singlemember(X,L2),subset(L,L2).

singlemember(X,[X|L]):-!.
singlemember(X,[Y|L]):-singlemember(X,L).

 firstmember([X|L],X).

addit([X],X).
addit([X|L],X1):- addit(L,X2),X1 is X+X2.

bigyear(Inyear,Year,Outyear):- Year >=Inyear, Outyear is Year.
bigyear(Inyear,Year,Outyear):- Year <Inyear, Outyear is Inyear.

concat([],L,L).
concat([X|L1],L2,[X|L3]):-concat(L1,L2,L3).

minyear([procurement(Year,Equip,Amount)],Year,Amount).
minyear([procurement(Year,Equip,Amount)|L],Year,Amount):-
           minyear(L,Y,A),Year<Y.
minyear([procurement(Year,Equip,Amount)|L],Y,A):-
           minyear(L,Y,A),not(Year<Y).

minimum([X],X).
minimum([X|L],X):-minimum(L,X2),X<X2.
minimum([X|L],X2):-minimum(L,X2),not(X<X2).

max([X],X).
max([X|L],X):-max(L,X2),X>X2.
max([X|L],X2):-max(L,X2),not(X>X2).


member(X,[X|L]).
member(X,[Y|L]):- member(X,L).

deleteone(X,[X|L],L).
deleteone(X,[Y|L],[Y|Z]):- deleteone(X,L,Z).

deleteall([],[]).
deleteall([procurement(X,Y,0)|L],L2):- deleteall(L,L2).
deleteall([Y|L],[Y|L2]):-not(Y=procurement(X,XX,0)),
deleteall(L,L2).
```

```
delete_all_e(Equip,[],[]).
delete_all_e(Equip,[procurement(X,Equip,Y)|L],L2):-
delete_all_e(Equip,L,L2).
delete_all_e(Equip,[Y|L],[Y|L2]):-not(Y=procurement(X,Equip,Z)),
          delete_all_e(Equip,L,L2).

appendd(Element,List,[Element|List]).

duplicate_in_list(List):-
            member(X,List),deleteone(X,List,Newlist),
            member(X,Newlist).

remove_duplicates(X,X):-not(duplicate_in_list(X)).
remove_duplicates(Listin,[Z|Listout]):-
            member(Z,Listin),
            deleteall_gp(Z,Listin,Tlist),
            remove_duplicates(Tlist,Listout).

deleteall_gp(X,[],[]).
deleteall_gp(X,[X|L],L2):-deleteall_gp(X,L,L2).
deleteall_gp(X,[Y|L],[Y|L2]):-not(Y=X),deleteall_gp(X,L,L2).

length_of_list([X],1).
length_of_list([X|L],I):-length_of_list(L,I1),I is I1+1.
```

```
/*   module planner6 */


/* this is a protytype plannng program    */
/* it comes up with a distribution but does not consider the  */
/* priority of the units or the fact that distribution must   */
/* modernizse the units as soon as possible                   */

/* this version of the planner program utilizes a delay factor as
part of */
/* the cost and evaluation of the soution.
  */

:-dynamic mytrace/3.
:-dynamic variable/1.

mytrace(allocated,0,0).
variable(1).

/* this traces how long a successor takes  */

successor(X,Y):-tracer,statistics(runtime,CG),
                allocated(X,Y),

statistics(runtime,Time),retract(variable(Count)),
                Counter is Count+1, asserta(variable(Counter)),
                asserta(mytrace(allocated,Counter,Time)).

successor(X,Y):- not(tracer),allocated(X,Y).

    /* allocate for procurement after equipment is available */

allocated(X1,XNew2) :-
        unit(NameU,PriorityU,TypeU),
        member(procurement(YearP,EquipP,AmountP),X1),
        AmountP>0,
        unitstatus_look_unit(X1,NameU,YearU,EquipU),
        forward(EquipU,EquipP),
        amount(EquipP,TypeU,Required),
        AmountP >= Required,
        Temp is YearU+3,
        bigyear(YearP,Temp,Giveyear),
         deleteone(procurement(YearP,EquipP,AmountP),X1,X2),
         NewamountP is AmountP - Required,
         append_if_necessary(X2,X3,YearP,EquipP,NewamountP),
        appendd(given(NameU,EquipP,Giveyear,Required),X3,X4),
        not(any_wrong_priority(X4)),
        member(given(NameU,EquipU,YearU,Oldamount),X4),
        YY is Giveyear +1,
```

```
give_to_PDO_or_procurement(X4,NameU,EquipU,YY,Oldamount,X5),
        modify_delay_if_necessary(X5,X7,Temp,Giveyear),
        get_rid_of_extraneous_proc(EquipU,X7,XNew2).


/* when more than one procurement is necessary for the unit  */

allocated(X1,XNew2) :-
        unit(NameU,PriorityU,TypeU),
        unitstatus_look_unit(X1,NameU,YearU,EquipU),
        forward(EquipU,EquipP),
        amount(EquipP,TypeU,Required),
        bagof(P,member(procurement(_,EquipP,P),X1),Plist),
        addit(Plist,AmountP),
        AmountP >= Required,
         bagof(Q,proc_for_equip(EquipP,X1,Q),Pelist),
        delete_all_e(EquipP,X1,X2),
        get_enough(Pelist,Required,Outlist,Tyear),
        Temp is YearU+3,
        bigyear(Tyear,Temp,Outyear),
        concat(Outlist,X2,X3),
        appendd(given(NameU,EquipP,Outyear,Required),X3,X4),
        not(any_wrong_priority(X4)),
        member(given(NameU,EquipU,YearU,Oldamount),X4),
        YY is Outyear +1,

give_to_PDO_or_procurement(X4,NameU,EquipU,YY,Oldamount,X6),
        modify_delay_if_necessary(X6,X7,Temp,Outyear),
        get_rid_of_0_procurements(X7,X8),
        get_rid_of_extraneous_proc(EquipU,X8,XNew2).


modify_delay_if_necessary(Oldlist,Oldlist,Year,Year).
/* unit given year as soon as the equipment is ready */
modify_delay_if_necessary(Oldlist,Newlist,Unityear,Giveyear):-
    Giveyear > Unityear,
    Diff is Giveyear - Unityear,
    deleteone(delay(Oldamount),Oldlist,Templist),
    Newdiff is Oldamount + Diff,
    appendd(delay(Newdiff),Templist,Newlist).

modify_delay_if_necessary(Oldlist,Oldlist,Unityear,Giveyear):-
     Giveyear<Unityear.
```

```
get_rid_of_extraneous_proc(Equip,X,Y):-
                member(procurement(Year,Equip,Amount),X),
                not(anybody_at_lower_levels(X,Equip)),
                deleteone(procurement(Year,Equip,Amount),X,X1),
                appendd(pdo(procurement,Year,Equip,Amount),X1,Y).


get_rid_of_extraneous_proc(Equip,X,X).
anybody_at_lower_levels(List,Equip):-
                forward(X,Equip),
                unitstatus_look_equip(List,X).

 anybody_at_lower_levels(List,Equip):-
                forward(X,Equip),
anybody_at_lower_levels(List,X).


append_if_necessary(List,Newlist,YearP,EquipP,NewamountP):-
                NewamountP>0,

appendd(procurement(YearP,EquipP,NewamountP),List,Newlist).

append_if_necessary(List,List,YearP,EquipP,NewamountP):-
                NewamountP=0.

unitstatus_look_year(List,Year):- unit(Name,_,_),
        unitstatus_look_unit(List,Name,Year,E1).

unitstatus_look_unit(List,Name,Year,Equip):-
        bagof([YY,EE],look_for_given(List,Name,YY,EE),U),
        maxequip(U,Year,Equip).

look_for_given(List,Name,Year,Equip):-
         member(given(Name,Equip,Year,Amount),List).

unitstatus_look_equip(List,Equip):- unit(Name,_,_),
            unitstatus_look_unit(List,Name,Year,E1),
            Equip=E1.

maxequip([[YY,EE]],YY,EE).
maxequip([[YY,EE]|L],YY,EE):-maxequip(L,Y1,E1),YY>Y1.
maxequip([[YY,EE]|L],Y1,E1):-maxequip(L,Y1,E1),not(YY>Y1).

proc_for_equip(EquipP,X1,procurement(Year,EquipP,Amount)):-
      member(procurement(Year,EquipP,Amount),X1).
```

```
get_enough(List,Required,Outlist,Year):-
     Required>0, minyear(List,Tyear,Amount),
     T is Required - Amount,
     T=0, deleteone(procurement(Tyear,X,Amount),List,Outlist),
     Year is Tyear.

/* when the amount in min procurement is not enough   */
get_enough(List,Required,Outlist,Year):-
     Required>0, minyear(List,Tyear,Amount),
     T is Required - Amount,  T>0,
     deleteone(procurement(Tyear,X,Amount),List,Toutlist),
     get_enough(Toutlist,T,Outlist,Year).

/* when the amount in the min procurement is more than enough */
get_enough(List,Required,Outlist,Year):-
     Required>0,
     minyear(List,Year,Amount),
     T is Required - Amount,
     T<0,
     deleteone(procurement(Year,X,Amount),List,Toutlist),
     Excess is Amount- Required,
     appendd(procurement(Year,X,Excess),Toutlist,Outlist).

/* this rule makes sure units with different priority */
/* have different priority equipment                  */

any_wrong_priority(List):- unit(N1,P1,_), unit(N2,P2,_),
     not(N1=N2), unitstatus_look_unit(List,N1,Y1,E1),
     unitstatus_look_unit(List,N2,Y2,E2),
     priority(E1,PE1),  priority(E2,PE2),
     priority_is_bad(E1,E2,Y1,Y2,P1,P2,PE1,PE2).

/* these are the two cases of non valid priorities  */
priority_is_bad(E1,E2,Y1,Y2,P1,P2,PE1,PE2):-
     E1=E2,  P1<P2,  Y1>Y2.

priority_is_bad(E1,E2,Y1,Y2,P1,P2,PE1,PE2):-
     not(E1=E2), P1<P2, PE1>PE2.

give_to_PDO_or_procurement(O_state,Name,Equip,Year,Amount,
     N_state):-
          check_for_a_unit(O_state,Equip),

appendd(procurement(Year,Equip,Amount),O_state,N_state).

check_for_a_unit(State,Equip):- forward(O_equip,Equip),
          unitstatus_look_equip(State,O_equip).

check_for_a_unit(State,Equip):- forward(O_equip,Equip),
          check_for_a_unit(State,O_equip).
```

```
/* this works when a unit does not need the equipment you are
giving */
/*    and there is a previous generation   */

give_to_PDO_or_procurement(Old_state,Name,Equip,Year,Amount,
     New_State):-
            forward(Old_equip,Equip),
            not(unitstatus_look_equip(Old_state,Old_equip)),

get_rid_of_remaining_procurements(Equip,Old_state,Old_state1),

appendd(pdo(Name,Year,Equip,Amount),Old_state1,New_State).


/* this works when a unit does not need the equipment you are
giving  */

give_to_PDO_or_procurement(Old_state,Name,Equip,Year,
     Amount,New_State):-
            not(forward(Old_equip,Equip)),

get_rid_of_remaining_procurements(Equip,Old_state,Old_state1),

appendd(pdo(Name,Year,Equip,Amount),Old_state1,New_State).


get_rid_of_remaining_procurements(X,[],[]).
get_rid_of_remaining_procurements(X,
          [procurement(Year,X,Amount)|L],
          [pdo(procurement,Year,X,Amount)|LL]):-
         get_rid_of_remaining_procurements(X,L,LL).
get_rid_of_remaining_procurements(X,[Transaction|L],
          [Transaction|LL]):-
         get_rid_of_remaining_procurements(X,L,LL)  .

get_rid_of_0_procurements(Old_state, New_state):-
    deleteall(Old_state,New_state).

/* the evaluation function calculates the least number of */
/* givens needed to get a unit to the highest procurement  */

eval(List,Z):-bagof(U,priorities(List,U),Prioritylist),
               minimum(Prioritylist,Z).


priorities(List,Return):- unit(Name,_,_),
                          unitstatus_look_unit(List,Name,_,Equip),

                          priority(Equip,Return).
```

```
/* the cost function is the number of time periods the plan lasts
plus */
/*  The delays incurred in this plan when equipment was available
and */
/* units were not*/
cost(List,Z):-bagof(U,unitstatus_look_year(List,U),Givelist),
                addit(Givelist,ZZ),member(delay(YY),List),
                dfactor(RR), Z is ZZ+YY*RR.

 /* define the goal of the program     */

goalreached(X):- get_rid_of_0_procurements(X, New_state),
                  subgoal(New_state).
subgoal(New_state):- not(member(procurement(_,_,_),New_state)).
subgoal(New_state):- not(allocated(New_state,X)).

/* this is a format for the program   */
done_searching_file(List,Fname):-any_wrong_priority(List),
   tell(Fname),write('it cannot be done'),nl,
        write('there is a problem with the units data'),nl,
        write('equipoment already out of priority sequence'),nl,
              told,halt.

done_searching_file(List,Fname):-
    not(tracer),
    save(coreplanner),
     write('core image saved you can ctl c now'),nl,
    appendd(delay(0),List,Newlist),write(Newlist),
    search(Newlist,X),sortplan(X,Y),
    tell(Fname),write(Y),statistics,
    countup(agenda(A,B,C),NA),write(NA),write(':incomplete '),
    countup(usedstate(S,SS),NB),write(NB),write(':examined '),
    told,halt.

done_searching_file(List,Fname):-
    tracer,
    save(coreplanner),
     write('core image saved you can ctl c now'),nl,
    appendd(delay(0),List,Newlist),
    search(Newlist,X),sortplan(X,Y),
    tell(Fname),write(Y),statistics,
    countup(agenda(A,B,C),NA),write(NA),write(':incomplete '),
    countup(usedstate(S,SS),NB),write(NB),write(':examined '),nl,

    print_mytrace,
    told,halt.

print_mytrace:-nl,
     write("trace of execution times of successor calculations"),
     nl,mytrace(A,B,C),printone(A,B,C),nl,fail.
print_mytrace.
```

```
printone(A,B,C):-write(A),write("    "),
     write(B),write("  "),write(C).

/* this is the fall through if it cannot do it  */
done_searching_file(List,Fname):-
      tell(Fname),write('it cannot be done'),told,halt.
```

```
/*   module planner7 */

/* this is a protytype plannng program   */
/* it comes up with a distribution but does not consider the  */
/* priority of the units or the fact that distribution must   */
/* modernizse the units as soon as possible                   */

/*  This version of the planning program uses the delay factors
but  */
/* provided information about the number of times a predicate
*/
/*  in the successor rule failed and succeeded.          */

:-dynamic mytrace/3.
:-dynamic variable/1.
:-dynamic tposvariable/2.
:-dynamic tnegvariable/2.
mytrace(allocated,0,0).
variable(1).

/* this traces how long a successor takes  */

successor(X,Y):-tracer,statistics(runtime,CG),
                allocated(X,Y),

statistics(runtime,Time),retract(variable(Count)),
                Counter is Count+1, asserta(variable(Counter)),
                asserta(mytrace(allocated,Counter,Time)).


successor(X,Y):-bigtrace,traceallocated(X,Y).

tcount(P):-call(P),incriment_poscall(P).
tcount(P):-incriment_negcall(P),fail.

incriment_negcall(P):-get_first_pred(P,FIRST),
                      retract_if_there_pos(FIRST,COUNT),
                       COUNTER is COUNT+1,
                      asserta(tnegvariable(FIRST,COUNTER)),!.
incriment_poscall(P):-get_first_pred(P,FIRST),
                      retract_if_there_neg(FIRST,COUNT),
                      COUNTER is COUNT+1,
                      asserta(tposvariable(FIRST,COUNTER)),!.
```

```
get_first_pred(P,FIRST):-P=..[FIRST].
get_first_pred(P,FIRST):-P=..[FIRST,_].
get_first_pred(P,FIRST):-P=..[FIRST,_,_].
get_first_pred(P,FIRST):-P=..[FIRST,_,_,_].
get_first_pred(P,FIRST):-P=..[FIRST,_,_,_,_].
get_first_pred(P,FIRST):-P=..[FIRST,_,_,_,_,_].
get_first_pred(P,FIRST):-P=..[FIRST,_,_,_,_,_,_].
get_first_pred(P,FIRST):-P=..[FIRST,_,_,_,_,_,_,_].


retract_if_there_pos(FIRST,COUNT):-tposvariable(FIRST,COUNT),

retract(tposvariable(FIRST,COUNT)).
retract_if_there_pos(FIRST,0).
retract_if_there_neg(FIRST,COUNT):-tnegvariable(FIRST,COUNT),

retract(tnegvariable(FIRST,COUNT)).
retract_if_there_neg(FIRST,0).

successor(X,Y):- not(tracer),not(bigtrace),allocated(X,Y).

    /* allocate for procurement after unit is available */

traceallocated(X1,XNew2) :-
       unit(NameU,PriorityU,TypeU),
       tcount(member(procurement(YearP,EquipP,AmountP),X1)),
       AmountP>0,
       tcount(unitstatus_look_unit(X1,NameU,YearU,EquipU)),
       forward(EquipU,EquipP),
       tcount(amount(EquipP,TypeU,Required)),
       AmountP >= Required,
       Temp is YearU+3,
       tcount(bigyear(YearP,Temp,Giveyear)),
       deleteone(procurement(YearP,EquipP,AmountP),X1,X2),
       NewamountP is AmountP - Required,
       append_if_necessary(X2,X3,YearP,EquipP,NewamountP),
       appendd(given(NameU,EquipP,Giveyear,Required),X3,X4),
       tcount(not(any_wrong_priority(X4))),
       member(given(NameU,EquipU,YearU,Oldamount),X4),
       YY is Giveyear +1,
       tcount(give_to_PDO_or_procurement(X4,NameU,
           EquipU,YY,Oldamount,X5)),
       modify_delay_if_necessary(X5,X7,Temp,Giveyear),
       get_rid_of_extraneous_proc(EquipU,X7,XNew2).

/* when more than one procurement is necessary for the unit  */
```

```
traceallocated(X1,XNew2) :-
        unit(NameU,PriorityU,TypeU),
        tcount(unitstatus_look_unit(X1,NameU,YearU,EquipU)),
        forward(EquipU,EquipP),
        amount(EquipP,TypeU,Required),
        tcount(bagof(P,member(procurement(_,EquipP,P),X1),Plist)),
        addit(Plist,AmountP),
        AmountP >= Required,
        tcount(bagof(Q,proc_for_equip(EquipP,X1,Q),Pelist)),
        delete_all_e(EquipP,X1,X2),
        get_enough(Pelist,Required,Outlist,Tyear),
        Temp is YearU+3,
        bigyear(Tyear,Temp,Outyear),
        concat(Outlist,X2,X3),
        tcount(appendd(given(NameU,EquipP,
             Outyear,Required),X3,X4)),
        tcount(not(any_wrong_priority(X4))),
        member(given(NameU,EquipU,YearU,Oldamount),X4),
        YY is Outyear +1,
       tcount(give_to_PDO_or_procurement(X4,NameU,EquipU,
                    YY,Oldamount,X6)),
        modify_delay_if_necessary(X6,X7,Temp,Outyear),
        get_rid_of_0_procurements(X7,X8),
        get_rid_of_extraneous_proc(EquipU,X8,XNew2).


allocated(X1,XNew2) :-
        unit(NameU,PriorityU,TypeU),
        member(procurement(YearP,EquipP,AmountP),X1),
        AmountP>0,
        unitstatus_look_unit(X1,NameU,YearU,EquipU),
        forward(EquipU,EquipP),
        amount(EquipP,TypeU,Required),
        AmountP >= Required,
        Temp is YearU+3,
        bigyear(YearP,Temp,Giveyear),
        deleteone(procurement(YearP,EquipP,AmountP),X1,X2),
        NewamountP is AmountP - Required,
        append_if_necessary(X2,X3,YearP,EquipP,NewamountP),
        appendd(given(NameU,EquipP,Giveyear,Required),X3,X4),
        not(any_wrong_priority(X4)),
        member(given(NameU,EquipU,YearU,Oldamount),X4),
        YY is Giveyear +1,

give_to_PDO_or_procurement(X4,NameU,EquipU,YY,Oldamount,X5),
        modify_delay_if_necessary(X5,X7,Temp,Giveyear),
        get_rid_of_extraneous_proc(EquipU,X7,XNew2).
```

```
/* when more than one procurement is necessary for the unit  */

allocated(X1,XNew2) :-
        unit(NameU,PriorityU,TypeU),
        unitstatus_look_unit(X1,NameU,YearU,EquipU),
        forward(EquipU,EquipP),
        amount(EquipP,TypeU,Required),
        bagof(P,member(procurement(_,EquipP,P),X1),Plist),
        addit(Plist,AmountP),
        AmountP >= Required,
         bagof(Q,proc_for_equip(EquipP,X1,Q),Pelist),
        delete_all_e(EquipP,X1,X2),
        get_enough(Pelist,Required,Outlist,Tyear),
        Temp is YearU+3,
        bigyear(Tyear,Temp,Outyear),
        concat(Outlist,X2,X3),
        appendd(given(NameU,EquipP,Outyear,Required),X3,X4),
        not(any_wrong_priority(X4)),
        member(given(NameU,EquipU,YearU,Oldamount),X4),
        YY is Outyear +1,

give_to_PDO_or_procurement(X4,NameU,EquipU,YY,Oldamount,X6),
        modify_delay_if_necessary(X6,X7,Temp,Outyear),
        get_rid_of_0_procurements(X7,X8),
        get_rid_of_extraneous_proc(EquipU,X8,XNew2).


modify_delay_if_necessary(Oldlist,Oldlist,Year,Year).

modify_delay_if_necessary(Oldlist,Newlist,Unityear,Giveyear):-
    Giveyear > Unityear,   Diff is Giveyear - Unityear,
    deleteone(delay(Oldamount),Oldlist,Templist),
    Newdiff is Oldamount + Diff,
    appendd(delay(Newdiff),Templist,Newlist).

modify_delay_if_necessary(Oldlist,Oldlist,Unityear,Giveyear):-
     Giveyear<Unityear.

get_rid_of_extraneous_proc(Equip,X,Y):-
                member(procurement(Year,Equip,Amount),X),
                not(anybody_at_lower_levels(X,Equip)),
                deleteone(procurement(Year,Equip,Amount),X,X1),
                appendd(pdo(procurement,Year,Equip,Amount),X1,Y).


get_rid_of_extraneous_proc(Equip,X,X).
```

```
anybody_at_lower_levels(List,Equip):- forward(X,Equip),
                unitstatus_look_equip(List,X).

anybody_at_lower_levels(List,Equip):- forward(X,Equip),
                anybody_at_lower_levels(List,X).

append_if_necessary(List,Newlist,YearP,EquipP,NewamountP):-
     NewamountP>0,

appendd(procurement(YearP,EquipP,NewamountP),List,Newlist).

append_if_necessary(List,List,YearP,EquipP,NewamountP):-
     NewamountP=0.

unitstatus_look_year(List,Year):-
        unit(Name,_,_), unitstatus_look_unit(List,Name,Year,E1).

unitstatus_look_unit(List,Name,Year,Equip):-
        bagof([YY,EE],look_for_given(List,Name,YY,EE),U),
        maxequip(U,Year,Equip).

look_for_given(List,Name,Year,Equip):-
         member(given(Name,Equip,Year,Amount),List).

unitstatus_look_equip(List,Equip):-  unit(Name,_,_),
            unitstatus_look_unit(List,Name,Year,E1),
            Equip=E1.

maxequip([[YY,EE]],YY,EE).
maxequip([[YY,EE]|L],YY,EE):-maxequip(L,Y1,E1),YY>Y1.
maxequip([[YY,EE]|L],Y1,E1):-maxequip(L,Y1,E1),not(YY>Y1).

proc_for_equip(EquipP,X1,procurement(Year,EquipP,Amount)):-
      member(procurement(Year,EquipP,Amount),X1).

/* when the amount in the minimum procurement is the correct
amount */
get_enough(List,Required,Outlist,Year):-
     Required>0,minyear(List,Tyear,Amount),
     T is Required - Amount,  T=0,
     deleteone(procurement(Tyear,X,Amount),List,Outlist),
     Year is Tyear.

/* when the amount in min procurement is not enough   */
get_enough(List,Required,Outlist,Year):-
     Required>0,  minyear(List,Tyear,Amount),
     T is Required - Amount,  T>0,
     deleteone(procurement(Tyear,X,Amount),List,Toutlist),
     get_enough(Toutlist,T,Outlist,Year).
```

```
/* when the amount in the min procurement is more than enough */
get_enough(List,Required,Outlist,Year):-
     Required>0,  minyear(List,Year,Amount),
     T is Required - Amount, T<0,
     deleteone(procurement(Year,X,Amount),List,Toutlist),
     Excess is Amount- Required,
     appendd(procurement(Year,X,Excess),Toutlist,Outlist).

any_wrong_priority(List):- unit(N1,P1,_), unit(N2,P2,_),
     not(N1=N2),
     unitstatus_look_unit(List,N1,Y1,E1),
     unitstatus_look_unit(List,N2,Y2,E2),
     priority(E1,PE1),  priority(E2,PE2),
     priority_is_bad(E1,E2,Y1,Y2,P1,P2,PE1,PE2).

/* these are the two cases of non valid priorities  */
priority_is_bad(E1,E2,Y1,Y2,P1,P2,PE1,PE2):- E1=E2, P1<P2, Y1>Y2.


priority_is_bad(E1,E2,Y1,Y2,P1,P2,PE1,PE2):- not(E1=E2),
      P1<P2, PE1>PE2.

give_to_PDO_or_procurement(O_state,Name,Equip,Year,
Amount,N_state):-
            check_for_a_unit(O_state,Equip),

appendd(procurement(Year,Equip,Amount),O_state,N_state).

check_for_a_unit(State,Equip):- forward(O_equip,Equip),
            unitstatus_look_equip(State,O_equip).

check_for_a_unit(State,Equip):- forward(O_equip,Equip),
            check_for_a_unit(State,O_equip).

 /* this works when a unit does not need the equipment you are
giving */
/*    and there is a previous generation   */

give_to_PDO_or_procurement(Old_state,Name,Equip,Year,
     Amount,New_State):-
            forward(Old_equip,Equip),
            not(unitstatus_look_equip(Old_state,Old_equip)),

get_rid_of_remaining_procurements(Equip,Old_state,Old_state1),

appendd(pdo(Name,Year,Equip,Amount),Old_state1,New_State).
```

```
give_to_PDO_or_procurement(Old_state,Name,Equip,Year,
     Amount,New_State):-
            not(forward(Old_equip,Equip)),

get_rid_of_remaining_procurements(Equip,Old_state,Old_state1),

appendd(pdo(Name,Year,Equip,Amount),Old_state1,New_State).

get_rid_of_remaining_procurements(X,[],[]).
get_rid_of_remaining_procurements(X,[procurement(Year,X,
          Amount)|L],
             [pdo(procurement,Year,X,Amount)|LL]):-
         get_rid_of_remaining_procurements(X,L,LL).
get_rid_of_remaining_procurements(X,[Transaction|L],
     [Transaction|LL]):-
         get_rid_of_remaining_procurements(X,L,LL) .

get_rid_of_0_procurements(Old_state, New_state):-
    deleteall(Old_state,New_state).

/* the evaluation function calculates the least number of */
/* givens needed to get a unit to the highest procurement  */

eval(List,Z):-bagof(U,priorities(List,U),Prioritylist),
               minimum(Prioritylist,Z).

priorities(List,Return):- unit(Name,_,_),
                          unitstatus_look_unit(List,Name,_,Equip),
                          priority(Equip,Return).

/* the cost function is the number of time periods the plan lasts
*/

cost(List,Z):-bagof(U,unitstatus_look_year(List,U),Givelist),
                addit(Givelist,ZZ),member(delay(YY),List),
                dfactor(RR),
                Z is ZZ+YY*RR.

 /* define the goal of the program     */

goalreached(X):-  get_rid_of_0_procurements(X  New_state),
                  subgoal(New_state).
subgoal(New_state):- not(member(procurement(_,_,_),New_state)).
subgoal(New_state):- not(allocated(New_state,X)).
```

```
/* this is a format for the program   */
done_searching_file(List,Fname):-any_wrong_priority(List),
   tell(Fname),write('it cannot be done'),nl,
      write('there is a problem with the units data'),nl,
      write('equipoment already out of priority sequence'),nl,
               told,halt.

done_searching_file(List,Fname):- not(tracer),not(bigtrace),
    save(coreplanner),
     write('core image saved you can ctl c now'),nl,
    appendd(delay(0),List,Newlist),write(Newlist),

search(Newlist,X),sortplan(X,Y),tell(Fname),write(Y),statistics,
    countup(agenda(A,B,C),NA),write(NA),write(':incomplete '),
    countup(usedstate(S,SS),NB),write(NB),write(':examined '),
    told,halt.

done_searching_file(List,Fname):- tracer,not(bigtrace),
    save(coreplanner),
     write('core image saved you can ctl c now'),nl,
    appendd(delay(0),List,Newlist),
     search(Newlist,X),sortplan(X,Y),
    tell(Fname),write(Y),statistics,
    countup(agenda(A,B,C),NA),write(NA),write(':incomplete '),
    countup(usedstate(S,SS),NB),write(NB),write(':examined '),nl,
    print_mytrace, told,halt.

done_searching_file(List,Fname):- bigtrace,
    save(coreplanner),
     write('core image saved you can ctl c now'),nl,
    appendd(delay(0),List,Newlist),
search(Newlist,X),sortplan(X,Y),
    tell(Fname),write(Y),statistics,
    countup(agenda(A,B,C),NA),write(NA),write(':incomplete '),
    countup(usedstate(S,SS),NB),write(NB),write(':examined '),nl,
    remove_dups1,remove_dups2,
     listing(tposvariable),listing(tnegvariable),
    told,halt.

remove_dups1:-tposvariable(X,Y),bagof(P,tposvariable(X,P),PLIST),
      addit(PLIST,AMOUNTP),abolish_all_x_pos(X),
      assert(tposvariable(X,AMOUNTP)),fail.
remove_dups1.

remove_dups2:-tnegvariable(X,Y),bagof(P,tnegvariable(X,P),PLIST),
      addit(PLIST,AMOUNTP),abolish_all_x_neg(X),
      assert(tnegvariable(X,AMOUNTP)),fail.
remove_dups2.
```

```
abolish_all_x_pos(X):-retract(tposvariable(X,Y)),fail.
abolish_all_x_pos(X).
abolish_all_x_neg(X):-retract(tnegvariable(X,Y)),fail.
abolish_all_x_neg(X).

print_mytrace:-nl,
     write("trace of execution times of successor calculations"),
     nl,mytrace(A,B,C),printone(A,B,C),nl,fail.
print_mytrace.

print_mybigtrace:-print_postrace,print_negtrace.

print_postrace:-nl,
    write("trace of execution times of successor calculations"),
    nl,tposvariable(B,C),printtwo(B,C),nl,fail.
print_postrace.

print_negtrace:-nl,
     write("trace of execution times of successor calculations"),
     nl,tnegvariable(B,C),printone(B,C),nl,fail.
print_negtrace.

printone(A,B,C):-write(A),write("    "),write(B),
     write("  "),write(C).
printtwo(B,C):-write(B),write("  "),write(C).

/* this is the fall through if it cannot do it  */
done_searching_file(List,Fname):-
   tell(Fname),write('it cannot be done'),told,halt.

sortplan([],[]).
sortplan([X|L1],L2):-sortplan(L1,L3),insert_in_plan(X,L3,L2).

insert_in_plan(X,[],[X]).
insert_in_plan(X,[Y|L],[X,Y|L]):- cost_in_plan(X,Xcost),
            cost_in_plan(Y,Ycost),Xcost<Ycost.
insert_in_plan(X,[Y|L1],[Y|L2]):- cost_in_plan(X,Xcost),
            cost_in_plan(Y,Ycost), Xcost>=Ycost,
            insert_in_plan(X,L1,L2).

cost_in_plan(given(Unit,_,Year,_),Cost):- unit(Unit,Priority,_),
           Cost is Priority*100+Year.

cost_in_plan(pdo(Unit,Year,_,_),Cost):- Cost is 1000+Year*10.

cost_in_plan(delay(C),0).
```

REFERENCES

1. Rowe, Neil C., *Artificial Intelligence Through Prolog*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

2. Hutson, David V., "A Program for Scheduling A Patrol Air Wing Training Plan," M.S. Thesis, Naval Postgraduate School, Monterey California, June 1988.

3. Bachant, Judith and John McDermott, "R1 Revisited: Four Years in the Trenches," *AI Magazine*, Fall 1984.

4. Bruno, Giorgio, Antionio Elia and Pietro Laface, "A Rule-Based System to Schedule Production," *IEEE Computer*, July 1986.

5. Goehring, Dwight J., "Time Constrained Planning Using Simulated Annealing," Conference Paper, IEA/AIE-88, June 1-3, 1988.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Director, Information Systems (OP-945) Office of the Chief of Naval Operations Navy Department Washington, D.C. 20350-2000	1
5.	Suprentendent, Naval Postgraduate School Computer Technology Programs Code 37 Naval Postgraduate School Monterey, CA 93943-5000	1
6.	Professor Neil C. Rowe, Code 52Rp Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5000	2
7.	Professor Sehung Kwak, Code 52Kw Department of Computer Science Naval Postgraduate School Montery, CA 93943-5000	2
8.	US. Army Signal Center & Fort Gordon Attn: ATZH-BCR Fort Gordon, GA 30905	1
9.	CPT Emil K. Velez 4530 N. Sacramento Chicago, IL 60625	2