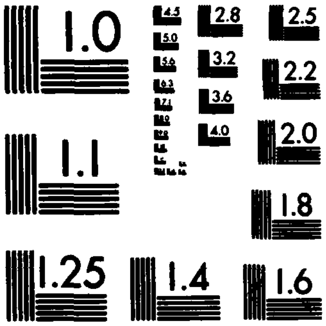


MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

13

AI&DS

FINAL TECHNICAL REPORT
AI&DS TR-1026-1

DECEMBER 1983

DYNAMIC SEQUENCE ASSIGNMENT

Cindy O'Reilly
Vladislav V. Rutenburg
Edison Tse
Richard P. Wishner

Contract No. N00014-82-C-0085
AI&DS Project No. 1026

Prepared for:

Office of Naval Research
Department of the Navy
Arlington, VA 22217

Approved for public release; distribution unlimited.

ADVANCED INFORMATION & DECISION SYSTEMS
Mountain View, CA 94040

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY
-

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release;
Distribution Unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AI&DS TR-1026-1

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Advanced Information &
Decision Systems

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State and ZIP Code)*
201 San Antonio Circle, Suite 286
Mountain View, CA 94040

7b. ADDRESS *(City, State and ZIP Code)*

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
Office of Naval
Research, Dept of the Navy

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
N00014-82-C-0085

8c. ADDRESS *(City, State and ZIP Code)*
802 N. Quincy Street
Arlington, VA 22217

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.

11. TITLE *(Include Security Classification)*
Dynamic Sequence Assignment (Unclassified)

12. PERSONAL AUTHOR(S)
O'Reilly, Cindy A.; Rutenburg, Vladislav V.; Tse, Edison; Wishner, Richard P.

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM 1/12/81 TO 12/30/83

14. DATE OF REPORT *(Yr., Mo., Day)*
December 1983

15. PAGE COUNT
100

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Mission Planning
Resource Allocation
Dynamic Programming
Decision Aids
Optimization
Marginal Return
Heuristic Search
Strike Planning

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

This report addresses the problem of building an interactive decision aid for near optimal aircraft mission planning. Mathematical formulations for dynamic sequential assignment of aircraft to targets/defenses in the mission planning process are presented. Several new mathematical algorithms for the solution of this problem are presented. An overview of the interactive mission planning methodology is also included.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS □

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
J. Roland Payne

22b. TELEPHONE NUMBER
(415) 941-3912

22c. OFFICE SYMBOL

EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd.)

LIST OF FIGURES

LIST OF FIGURES (Cont'd.)

1. INTRODUCTION

This final report describes the work that AI&DS has performed for ONR under the contract entitled "Dynamic Sequence Assignment." The ultimate goal of our long-range research effort is to design and develop an interactive analytic^l planning aid for aircraft mission planning. The primary objective of this contract was to investigate various mathematical algorithms and design methodologies and techniques to achieve this goal.

The decision aid to be developed should perform the functions dealing with all the low-level procedural, computational and search tasks, enabling the decision makers to concentrate on the important high-level planning issues. Thus, this decision aid will provide for a successful symbiosis between the man and the computer, utilizing the best capabilities of each of them.

The overall control of the decision process will be in the hands of the human decision maker, who will be in charge of providing the overall tactical guidelines. He will use the decision aid for: a) complex mathematical computation of the best plans within the given tactical guidelines, and b) evaluation and graphic simulation of tactical plans.

The following tasks were completed during this project, in preparation for the building of such a decision aid:

a) Designed a preliminary mission planning testbed

b) Developed mathematical and structural theory of mission planning

c) Created optimal or near-optimal, human-guided algorithms, for generation of the best plans within a given strategic guideline.

Our investigation began with developing basic mathematical theory and models of the forces and parameters present in modern air missions, and of the high-level control structure involved in such missions.

After these models were developed, we implemented the most representative scenario in the form of a basic computer testbed database. The testbed, in addition to providing a convenient environment for quick testing of human generated mission plans, also provides some important aids to planning. In particular, it contains an algorithm that automatically finds the best flight path for each individual sortie and evaluates the goodness of each such sortie.

Our next research goal was that of developing efficient mathematical algorithms necessary for automation of mission planning. The two major issues to be addressed were the high level sortie assignment problem and the low level problem of path selection. Dynamic programming (DP) formulation provided the best conceptual framework for solution of both these problems. The two most important mission control models have been successfully formulated as DP´s, and the relevant mathematical equations have been developed. An efficient DP formulation for the lower-level problem of path optimization for individual sorties has also been designed and implemented.

In the process of our research on dynamic programming we discovered a new algorithm, which combines the ideas of dynamic programming with branch-and-bound search techniques and provides significant advantages over the other two with respect to the mission planning problem. We called this algorithm Depth-First Dynamic Programming (DFDP).

In addition to DFDP, which is an optimal algorithm, we designed a number of faster heuristic solution algorithms. Most of these algorithms, as well as a streamlined version of DFDP, have been implemented on a PDP-20 computer. We have run a number of tests and comparisons between the developed algorithms and found important differences and trade-offs between them.

To improve the algorithms we developed, and to make them adhere more closely to real-life situations, we examined partitioning the problem, both spatially and temporally. We approached the problem spatially from two different perspectives - partitioning the scenario of interest into separate regions, and grouping sets of targets together. In analyzing it temporally, we considered multiple attack waves, where defenses may be able to reconstitute themselves before they are hit again.

We have also studied the overall design methodology for computer-aided mission planning, including decomposition of the overall problem into modular subproblems, and selection of proper utility measures.

Finally, we have developed basic ideas for extending our methods to several major issues involved in more complex models, including such issues as plan compression, multiple weapon loads, and uncertainty.

This report is organized as follows. Section 2 is devoted to a detailed description of our mathematical models of basic mission planning problems. Mathematical algorithms for the solution of such problems are described in Section 3. In Section 4, we highlight the Streamlined Depth-First Dynamic Programming algorithm and describe the software developed to implement it. Section 5 explores our partitioning ideas and presents some of the results we have obtained in implementing these ideas. In Section 6, we present an overview of the interactive mission planning methodology. Section 7 presents new modeling issues that need to be addressed in future research. Finally, in Section 8 we present our conclusions. There are three appendices in our report. Appendix A presents a dynamic programming approach to mission planning. Appendix B describes the Depth-First Dynamic Programming algorithm. In Appendix C we provide a detailed formulation of the Marginal Utility algorithm.

2. SCENARIO FORMULATION

In this section we present our mathematical formulations of basic mission planning scenarios. Subsection 2.1 introduces the basic battle model, while subsection 2.2 describes two of the most important command and control structures that apply to our scenario.

2.1. BASIC BATTLE SCENARIO

The current formulation is a high-level model of the surface-to-land battle situation. The forces involved in the battle are friendly aircraft carriers with airplanes and/or ship/submarine-borne surface-to-land cruise missiles on board poised against enemy land targets and enemy defenses protecting those targets.

The objective of an attack mission is to inflict the maximum possible damage to the enemy targets, subject to aircraft survivability constraints. Planning of this mission involves jointly making the following decisions:

a) which high-valued targets to attack,

b) which defensive threats should be destroyed to facilitate the attack,

c) which attacking units to assign to each objective,

d) the sequential order of these attacks, and

e) how to optimize the performance of each attack mission (e.g., path optimization, use of EW resources, etc.).

The rules of the battle are as follows:

There are n targets, each target T having its own initial military value V(T) assigned to it. There are m defenses protecting the approaches to the targets. Friendly forces consist of q aircraft carriers with NAi airplanes on board carrier i.

Each airplane A will carry out an assignment against one of the enemy objects and is allowed to choose the safest path for reaching that object within the available fuel allowance fi. If an airplane A attacks a target

T, then A will destroy T with probability PK(A -> T). If an airplane A attacks a defense D, then A will destroy D with probability PK(A -> D). If an airplane A flies near a defense, then it will be shot down with probability PK(D -> A), which depends on the proximity and the duration of the exposure. All single events are assumed independent. For example, if an airplane flies over two defenses, D1 and D2, then its probability of survival Psur(A) will be equal to

$$\mathrm{Psur}(A)=[1-\mathrm{PK}(D1\to A)]\cdot[1-\mathrm{PK}(D2\to A)] \qquad (2\text{-}1)$$

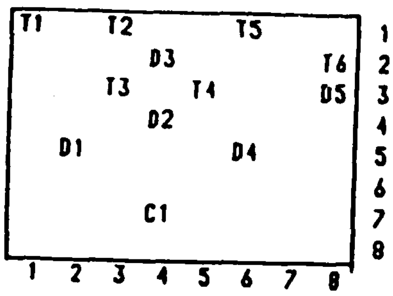
The objective is to find the sequence of assignments that maximizes the expected success E(L) of the attack, which is equal to

$$E(L)=\sum_{i=1}^{n} V(Ti)\cdot(1-PS(Ti)) \qquad (2\text{-}2)$$

where PS(Ti) stands for the probability that Ti survives all the attacks directed against it. The value of PS(Ti) depends on the number of airplanes attacking it, the probability that these airplanes will avoid all the defensive threats along their paths, and the probability of their success against the target.

<u>Example 1</u>:

Figure 2-1 introduces a small example of our scenario. This example will be referred to several times throughout this report. Figure 2-1 represents an aerial view of the battlefield. The field is divided into an 8 x 8 square grid. There are six targets, T1 through T6, in the top half of the field. On the right we can see the value V(T) associated with each target T. For example, target T3 is the most valuable with 350 points while T1 is the least valuable with only 100 points. Five enemy defenses, D1 through D5, protect the approaches to the targets.



	PE		Value:
D1	.99	T1	100
D2	.99	T2	300
D3	.99	T3	350
D4	.99	T4	300
D5	.99	T5	150
		T6	200

Airplanes: A1 A2 A3 A4 A5 A6 A7 A8 A9
A10 A11 A12 (all on board C1)

Figure 2-1 Example Scenario

There is one aircraft carrier, C1, with twelve airplanes on it. For simplicity of discussion, all the airplanes are presumed to have identical capabilities, as do all the enemy defenses. Each airplane has capacity for f = 17 units of fuel. It takes two units of fuel to fly from the center of one square to the center of the adjacent one if one is flying vertically or horizontally; it takes three units of fuel if one is flying diagonally, as seen in Fig. 2-2.

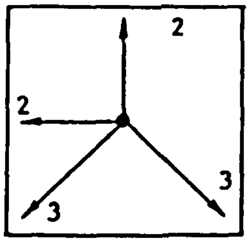
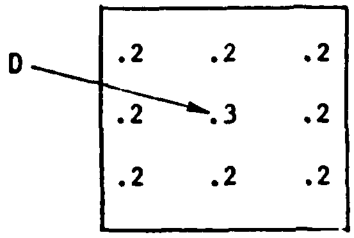


Figure 2-2 Fuel Consumption

An allowable flight path is thus a chain of vertical, horizontal and diagonal transitions whose "fuel length" is no greater than 17.

The following are the values assigned to various probabilities of kill: PK(A -> T) = .8; PK(A -> D) = 1.0; probability of kill by a defense D against an airplane A (i.e., PK(D -> A))that is flying by is computed accordingly to the following rule (see Fig. 2-3): every time an airplane transitions into one of the eight squares next to defense D, the airplane will be shot down with probability of 20%, while a transition into the central square, where D is situated, entails a 30% probability of kill.



.2	.2	.2
.2	.3	.2
.2	.2	.2

Figure 2-3 Defensive Threat Probabilities

2.2. MODELS OF PLANNING AND CONTROL

There can be many ways in which attacks can be executed. For example, they could be executed in parallel all at once, or they could be executed one at a time, or in several attack waves. The execution structure of a particular attack in real life depends on the situation in general and on the time limitations and intelligence capabilities in particular.

As our initial models of control we chose the two most representative situations. They come from the opposite extremes of the spectrum of all available controls. They are called Closed- and Open-loop Control Models, for the reasons that will be described subsequently.

2.2.1. Closed-loop Control Scenario

Missions are executed sequentially, each new one starting after the completion of the preceding one, with the knowledge of its outcome. Thus we have a perfect feedback situation.

Example 2:

Let us consider the situation in Example 1 (see Fig. 2-1). A decision maker may decide to start his attack by sending one of his airplanes, say A1, against T2. The optimal path computation routine predicts that A1 has a 32% chance of destroying T2. In the case that A1 is successful, the new battle position is obtained from that of Figure 2-1 by deleting A1 and T2 from consideration. Notice that we have accumulated 300 points as a result of destroying T2. In this situation we may decide that the next target to attack would be T3, and send A2 against it.

On the other hand, our first assignment (A1 -> T2) could have failed with probability 68%. Then we would have faced the same position as in Figure 2-1, except that now A1 is no longer available. No points have been accumulated, of course. This situation requires its own new decision, and we decide to send A2 against T2, also.

Notice that different outcomes can thus entail different future assignments. For each of the two possible outcomes of the first decision, there will be a second decision with two possible outcomes, and so on.

Example 2 illustrates the point that a complete solution to the closed-loop problem has to be in the form of a decision tree, for example that in Figure 2-4.

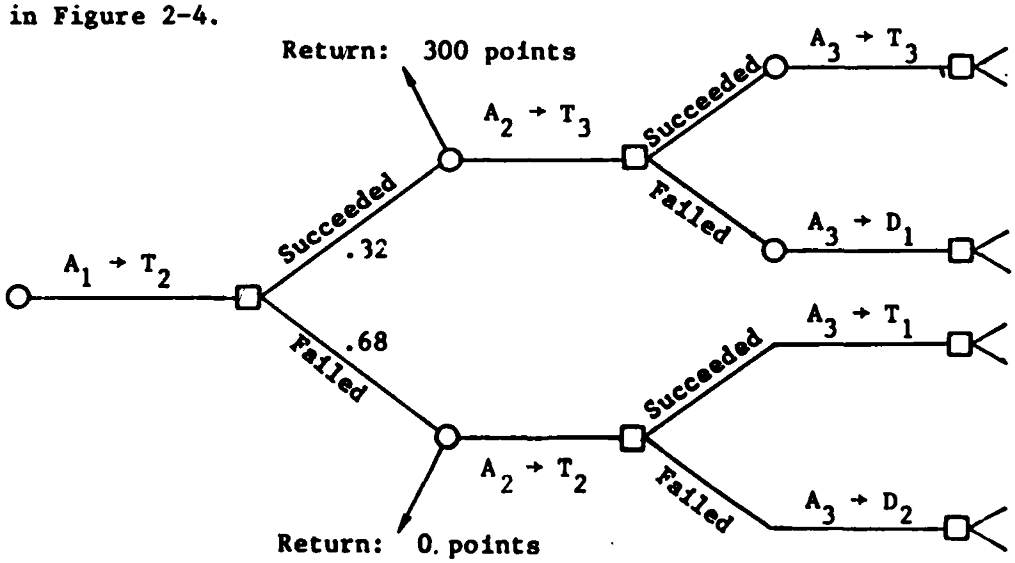


Figure 2-4 Decision Tree for the Closed Loop Scenario

In the closed-loop scenario the equation (2) for the expected return function becomes:

$$E(L) = \sum_{i \in OC} \sum_{j=1}^{n} V(Tj) \cdot I_{Tj}(i) \cdot \mathrm{Prob}(i) \qquad (2\text{-}3)$$

where OC is the set of all final outcomes, and Prob (i) is equal to the probability of final outcome i, and

$$I_{Tj}\ (i) = \begin{cases} 1 - \text{if Tj has been destroyed} \\ \qquad \text{in outcome i} \\ 0 - \text{otherwise} \end{cases} \tag{2-4}$$

2.2.2. Open-loop Control Scenario

In this scenario all missions are executed sequentially, but the outcomes of earlier missions do not get fed back to the controller and thus do not influence the future assignments. In the open-loop case, the probabilistic outcome of each attack is predicted before its execution. Consequently, there is no advantage in waiting for the outcome of one mission before proceeding with the next one, and the whole attack sequence can be planned in advance.

Example 3:

Let us see how the open-loop control will work for the scenario in Example 1. A particular open-loop attack plan is displayed in Fig. 2-5.

A1 → D1

A2 → D1

A3 → D3

A4 → D4

A5 → D4

A6 → T2

A7 → T3

A8 → T4

A9 → T5

A10 → T6

A11 → T1

A12 → T3

Figure 2-5 Attack Sequence for Open-loop Scenario

This plan assigns five airplanes to the task of creating holes in the enemy defensive line, as a first wave of attack. In the second wave, the remaining seven airplanes will then proceed with their attacks against the targets: target T3, the most valuable one, is attacked twice, while the other five targets are attacked once each.

After the first wave attack is completed, all the enemy defenses still have to be taken into consideration, but the three defenses that were subject to our attack now have a very low probability of existence (PE) as seen in Fig. 2-6.

D1	.08
D2	.99
D3	.47
D4	.08
D5	.99

Figure 2-6 Probabilities of Existence of Defenses After First Wave of Attack

Attacks against the targets are also evaluated in the "probability of existence" sense. Fig. 2-7 summarizes the outcomes of the second wave attack. The numbers in the second column stand for how much value we managed to extract from each target, and the numbers in the first column stand for its remaining value. For example, target T3, which had 350 points in the beginning, is now worth only 72 points, with 278 having been already extracted by airplanes A7 and A12.

T1	25	75
T2	97	203
T3	72	278
T4	136	164
T5	37	113
T6	104	96

Figure 2-7 Final Values of Targets

The total number of points extracted is thus 929, with 471 points still remaining.

The reason the two control scenarios described above are so important is that almost all real-life feedback command and control situations can be viewed as being a combination of those two.

The optimal scores for the two scenarios also provide upper- and lower-bounds for the performance of any "combination" scenario. The closed-loop case provides much more flexibility and responsiveness to the decision maker than the open-loop case, and thus supplies the higher performance bound (Rutenburg, 1982a).

Because of the importance of each of the two control scenarios, we have worked on modeling and solving each of the two. The same mathematical ideas apply to both scenarios, although the algorithmic details are different.

3. MATHEMATICAL SOLUTION ALGORITHMS

In this section we present some of our results in developing efficient mathematical algorithms for finding optimal solutions to mission planning problems.

The common denominator in our approach to the solution of problems of this class lies in decomposition of a problem into a hierarchy of sub-problems. In particular, the basic fundamental scenarios that we are dealing with in this report can be decomposed into two levels of optimization sub-problems. The top level is responsible for finding the optimal overall sequence of high-level mission assignments of aircraft to targets or defenses, while the lower level is responsible for finding the optimal use of resources (fuel, for example) within each flight mission and for evaluating the likelihood of success of that mission.

The idea that ties the two levels together is that finding a good high-level plan requires the ability to estimate the needs and consequences of individual missions, which is exactly the responsibility of the low-level sub-problem. Therefore, the lower level can be viewed in this context as an evaluation subroutine for the top level.

Our initial approach to path optimization is that of dynamic programming, with remaining fuel representing the stage variable and position coordinates being the state variables. It follows the spirit of earlier works in this area (see, for example, Appendix A).

The low-level sub-problem, which in our scenario involves path optimization, has been studied by many researchers over the years. The higher-level problem of optimal mission assignment has not, to our knowledge, been properly addressed in the past. Only highly ad hoc and heuristic solution methods are reported in the literature (Callero, Jamison, and Waterman, 1982; Case and Thibault, 1977; Engelman, Berg, and Bischoff, 1979). It is therefore a major goal of our research to develop a systematic mathematical theory of, and algorithmic solutions to, global mission planning.

This section consists of several subsections. The first one presents the dynamic programming (DP) solution to the low level path optimization problem. The second subsection presents DP formulations for both open- and closed-loop models of the high-level assignment problem. In subsection 3.3 we discuss the new depth-first DP algorithm, developed for the mission assignment problem. The rigorous definition of this algorithm is presented in Appendix B. We also developed a streamlined version of this approach which is not described in this seciton, but is highlighted in more detail in the next. Finally, subsection 3.4 reviews some important heuristic algorithms developed for faster solution of the sequence assignment problem.

3.1. DYNAMIC PROGRAMMING FORMULATION FOR PATH OPTIMIZATION

Our approach to path optimization is similar to the classical DP treatment of this problem (see Larson and Casti (1978); Bellman (1957); also Appendix A). In this subsection we presume that the reader is familiar with the basic DP algorithm ideas, and we briefly present the main details of our DP formulation.

We are given the starting position (X_0, Y_0) together with initial fuel F_0, and the final position (X_f, Y_f). We need to find a path $\{(X_0,Y_0),(X_1,Y_1), \ldots (X_f,Y_f)\}$, which maximizes the probability of survival between the starting and final positions subject to the constraint that one uses F_0 units of fuel or less.

The stage variable is represented by the amount of fuel left on board. The state within a stage is specified by the geographical position (X,Y). Notice that time does not explicitly enter into our formulation because it is presently modeled that fuel consumption rate is constant over time for each airplane, and thus, time can be computed from the amount of fuel used.

Let R(X,Y,F) represent the optimal survival transition function between points (X,Y) and (X_f,Y_f), with F units of fuel available. As the reader remembers, we are dealing with an integer grid model, and there are 8 possible transitions from point (X, Y) to the next point along a trajectory. These transitions and the fuel consumption along each one of them were described earlier (see Figure 2-2).

Thus, the main DP equation becomes:

$$R\ (X,\ Y,\ F) = \text{Max}\ \left| R\ (X \pm 1,\ Y \pm 1,\ F-3) * S\ (X,\ Y,\ X \pm 1,\ Y \pm 1), \right.$$
$$\left. R\ (X \pm 1,\ Y,\ F-2) * S\ (X,\ Y,\ X \pm 1,\ Y),\ R\ (X,\ Y\ \pm 1,\ F-2) * S\ (X,\ Y,\ X,\ Y \pm 1) \right| \qquad (3\text{-}1)$$

where $S(X, Y, X_1, Y_1)$ represents the probability of survival between points (X, Y) and (X_1, Y_1).

This can be computed using the formula

$$S\ (X,\ Y,\ X_1,\ Y_1) = 1 - \prod_{Di \in NBD\ (X_1,\ Y_1)} P_k\ (Di \rightarrow A) \qquad (3\text{-}2)$$

where NBD (X_1, Y_1) stands for the set of defenses in the immediate vicinity of (X_1, Y_1).

The boundary conditions are given by

$$R\ (X,\ Y,\ 0) = \begin{cases} 1 \text{ if } (X,\ Y) = (X_f,\ Y_f) \\ 0 \text{ otherwise} \end{cases} \qquad (3\text{-}3)$$

and

$$R\ (X_f,\ Y_f,\ F) = \begin{cases} 1 & \text{if } F \geq 0 \\ 0 & \text{if } F < 0 \end{cases} \qquad (3\text{-}4)$$

The above formulation allows us to solve the path optimization problem using a conventional backward dynamic programming algorithm.

3.2. DYNAMIC PROGRAMMING APPROACH TO MISSION PLANNING

Mission planning is inherently a very complex computational problem. Trying to solve even an average-sized mission assignment problem using methods of direct enumeration of all possible attack sequences proves to be an impossible task due to the faster-than-exponential growth of the problem size.

A more efficient approach is to apply the dynamic programming method to the higher-level sequence assignment problem.

3.2.1. Dynamic Programming Formulation for Open-loop

The current state of the battle under this scenario is characterized by the number of aircraft available on each carrier, by the probability of existence of each enemy defense, and by the current value of each target. Thus, a state S can be defined as

$$S=(NA_1, \ldots, NA_q, PE(D1), \ldots, PE(Dm), PE(T1), \ldots, PE(Tn)), \qquad (3\text{-}5)$$

where NA_i is the number of airplanes currently available on board carrier i , and PE(X) stands for the current probability of existence of object X.

The stage (i.e., decision point) corresponds to the total number of airplanes already assigned, being equal to

$$\sum_{i=1}^{q} (\overline{NA}_i - NA_i), \qquad (3\text{-}6)$$

where $\overline{NA}_i$ is the initial number of airplanes on board carrier i.

An action u in this formulation corresponds to assigning one of the available airplanes from one of the carriers on a mission against one of the enemy forces. The outcome of this action will take us to the next stage and a new state with one fewer airplane available and with the targeted enemy force having a reduced probability of existence. If that enemy force happens to be a target, say Ti, then we derive a reward L(S,u) equal to

$$L(S, u)=V(Ti)\cdot\Delta PE(Ti), \qquad (3\text{-}7)$$

where V(Ti) is the initial value of Ti, and $\Delta PE(Ti)$ corresponds to the change in probability of existence of Ti due to the current action.

The dynamic programming equation is then represented by

$$J(S(i))=\max_{u\varepsilon U_i}\{L(S(i) , u)+J(S(i+1))\} \qquad (3\text{-}8)$$

where J(S) is the optimal total reward that can be derived starting at state S, U_i is the set of all possible actions at state S(i), and S(i+1) is the state resulting from applying action u to state S(i).

3.2.2. Dynamic Programming Formulation for Closed-loop

A state S in this scenario is similar to that in the open-loop scenario, the difference being that an object is not represented by a probability of existence anymore; it either exists or doesn´t. Thus,

$$S=(NA_1, \ldots, NA_q, I_{D1}, \ldots, I_{Dm}, I_{T1}, \ldots, I_{Tn}), \tag{3-9}$$

where

$$I_X = \begin{cases} 1 - \text{if object X exists} \\ 0 - \text{otherwise} \end{cases}$$

The stage variable is the same as that for the open-loop case and is given by (3-6). An action u corresponds to the same type of a mission assignment as in the open-loop case. If the mission u is directed against an enemy target Ti then the reward L(S,u) is given by

$$L(S, u)=V(Ti)\cdot PS(u), \tag{3-10}$$

where PS(u) is equal to the probability of success of mission u (computable by path optimization DP).

Unlike the open-loop case, which corresponds to deterministic dynamic programming, the closed-loop case gives rise to stochastic dynamic programming, due to the random nature of mission outcomes. With probability equal to PS(u) the outcome will be successful and will bring us to a new state S(i+1) with one fewer airplane available, and the indicator variable for the destroyed enemy object now reset from 1 to 0. On the other hand, with probability equal to 1 - PS(u) we will fail and result in a state $\overline{S}(i+1)$, again with one fewer airplane available, but with no enemy object having been destroyed. The dynamic programming equation now becomes

$$J(S(i))=\max_{u\in U_i}\{L(S(i),\ u)+PS(u)\cdot J(S(i+1))+(1-PS(u))\cdot J(\overline{S}(i+1))\} \qquad (3\text{-}11)$$

3.3. DEPTH-FIRST DYNAMIC PROGRAMMING ALGORITHM

3.3.1. Basic Formulation

In our efforts to develop an efficient implementation of the dynamic programming (DP) ideas, we formulated a new version of the dynamic programming algorithm. This algorithm combines the best features of both dynamic programming and of depth-first tree search algorithms. That is why it has been named Depth-First Dynamic Programming (DFDP) (Rutenburg,1982b).

DFDP is related to depth-first tree search (DFTS) by the order in which the states in the state-space are explored. Like DFTS (Winston,1979) it follows an initial "path" through the state-space from the initial state to one of the final states, thus finding an initial solution path. It then backtracks to the previous state and explores other actions available at that state, then backtracks again, and so on. In the process of the search, the algorithm computes optimal rewards for all the visited states and also updates the currently best known global solution path.

DFDP is similar to dynamic programming in its use of the underlying telescoping effects to reduce the complexity of the search involved. By telescoping effects, we mean that the structure of the state-space is not a tree but rather a lattice, with many different paths leading to the same resulting state. By making the optimal reward for any state S (computed for a particular path leading to S) available for all the other paths leading to state S, DFDP algorithm reduces the computational complexity of the depth-first search from exponential down to polynomial of the order

$$O(Ns\cdot Nu), \qquad (3\text{-}12)$$

where Ns is the total number of states and Nu is the average number of actions per state. This is the same complexity as that of the traditional DP algorithm, which can be described as a backwards breadth-first search technique.

Appendix B provides a rigorous definition of a DFDP algorithm for a general class of dynamic programming problems and a detailed example illustrating its performance.

Example 4:

Let us apply the DFDP algorithm to the open-loop scenario described in Examples 1 and 3 (see Figures 2-1 and 2-7). Figure 3-1 shows how the current optimal value was changing during the computation. The first solution found was worth 802 points, then the algorithm found a better solution, worth 870 points, and so it went until the optimal solution, worth 1097 points, was found. This is significantly better than the 929 points obtained from manual planning in Example 3. Figure 3-2 presents the summary of the optimal plan. It involves sending two airplanes to destroy defense D2, then sending two airplanes against defense D3, thus blowing a hole in the enemy defensive line and also freeing the most valuable targets, T2, T3 and T4, from the heavy defensive protection. After the two defenses are severely impaired, the plan asks for two airplanes to be sent against targets T2 and T3, and for one airplane to be sent against each of the other targets.

802.0592

870.2677

911.0408

960.5516

972.1386

1049.802

1094.562

1097.316

Figure 3-1 Sequence of Currently Optimal Scores

A1 → D2

A2 → D2

A3 → D3

A4 → D3

A5 → T3

A6 → T4

A7 → T2

A8 → T5

A9 → T6

A10 → T1

A11 → T3

A12 → T2

Figure 3-2 Optimal Plan

3.3.2. Uses of Depth-First DP

Although similar in spirit, DFDP acquires several advantages over regular DP from its depth-first search approach. Its first advantage lies in its satisficing capabilities. This refers to situations when (for example, due to time pressures) it is not necessary to find the optimal solution, but rather a satisficing one, i.e., a solution whose value exceeds a given threshold on performance. Because DFDP starts with a particular solution and then keeps on improving it, this algorithm can terminate at the moment when one satisficing solution is found. This contrasts with the regular DP approach, which doesn´t find any solution until near the very end of computation.

A similar situation occurs when the algorithm is used in a real-time situation, when it is not known in advance how much time can be spent on solving a given problem. With DFDP, a good, currently best solution can be produced at any moment a solution is needed, a capability unavailable from regular DP.

Another advantage of DFDP lies in its ability to use branch-and-bound techniques to prune down the search space. Branch-and-bound is a well-known technique, used, for example, in integer programming (Nemhauser and Garfinkel,1972) and in AI tree search (Winston,1979). DFDP implementation makes the branch-and-bound ideas applicable for the first time (to our best knowledge) in dynamic programming.

The branch-and-bound approach involves computing an upper bound UB(S) on the optimal return J(S) for a given state in the state space. As we mentioned earlier, DFDP keeps track of the current optimal performance (COP) for the given problem throughout the computation. A state S need not be considered when

$$F(S)+UB(S)<COP \tag{3-13}$$

where F(S) is the score obtained by arriving to state S along the currently followed path.

The degree of success to be expected from the satisficing and branch-and-bound methods of DFDP depends greatly on the problem at hand and on the quickness with which near-optimal solutions can be found. The latter is a function of the order in which various actions originating from a given state are explored. A good order can be achieved by the use of a heuristic guidance function h(S) to estimate the value of the optimal performance function J(S) for a given state S. It is easy to see that if h(S) very closely approximates J(S), then the optimal solution is going to be found almost immediately, and that the algorithm efficiency decreases as h(S) becomes less reliable, just like in tree search situations (Nilsson, 1980). Finding good heuristic guidance is a hard task from the realm of artificial intelligence, and we are currently involved in developing such heuristic functions for our mission planning problem.

DFDP has one weak point, in that the current version has on the average higher storage requirements. This fact hasn´t caused any important inconveniences so far, but it is important to try to find implementations of DFDP with more efficient use of storage space.

3.4. RELATED MATHEMATICAL AND HEURISTIC ALGORITHMS

The mission assignment problem is intrinsically so complex that running even the best optimal algorithms will take a very long time on today´s mini-computers. In this section we present several heuristic algorithms which don´t guarantee optimality but are fairly fast, and some of them do provide optimality in most cases.

3.4.1. Sequential Assignment Algorithm

The sequential assignment algorithm chooses a set of airplane assignments, one at a time, in a greedy but myopic fashion. It is greedy in that during each assignment cycle, it chooses that assignment yielding the highest expected return; it is myopic in that it does not consider the effect that the current decision could have on future assignments.

Initially, only targets are considered as possible objectives since defenses have no explicit value. During the first assignment cycle, the expected return associated with each possible assignment of an airplane to a target is calculated based on the probability of surviving the optimal path to the target, the probability of then destroying the target, and the value remaining to the target. The assignment of airplane to target having the largest expected return is determined, with ties being broken arbitrarily. The chosen assignment, say airplane A1 to target T1, is irrevocably made, though not executed immediately. The value remaining to T1 for later calculations is reduced appropriately.

In all subsequent assignment cycles only those airplanes without assignments are considered. Potential assignments of airplanes to targets are evaluated as before. In addition, defenses are now treated as allowable objectives. Determining the expected value derived from attacking a defense is more difficult than for a target. Consider assigning airplane A2 to defense D1 in the second assignment cycle, remembering that the assignment of A1 to T1, has not actually been executed yet. We calculate the probability of existence for D1 after a hypothetical attack by A2 and calculate the expected return for the assignment A1 to T1, given that D1 has been probabilistically weakened. The net improvement in the assignment of A1 to T1 is the value given to the assignment of A2 to D1. The assignment of airplane to target or defense with the largest expected return is the one chosen in the second assignment cycle.

In general, the value associated with attacking a defense is calculated by determining the net improvement to all previously chosen assignments of airplanes to targets. Note that the assumed order in which missions are to be executed is not the order in which they are chosen. The evaluation procedure assumes that all missions against defenses are executed first, in the order in which they are chosen, followed by attacks on targets in any order. During any cycle, if the chosen assignment is to a target, only the value remaining to that target need be updated; if the assignment is to a defense, the probability of existence of that defense, all optimal paths affected by attacking that defense, and the value remaining to any target with a previously assigned airplane that is affected by attacking that defense must be updated.

After running through as many assignment cycles as there are airplanes, the algorithm terminates. Because the algorithm is myopic, there is no guarantee that the solution that it arrives at will be optimal.

3.4.2. Target Assignment Algorithm

The problem of deciding on a good attack plan, which specifies not only sorties against enemy targets but also sorties to suppress enemy defensive units, has not, to our knowledge, been previously addressed. However, the much easier problem of exclusively choosing sorties against enemy targets has been looked at. The most common algorithm used for this purpose has been what we call Target Assignment (TA) Algorithm.

This algorithm is effectively a special case of our sequential assignment algorithm, with the restriction that enemy defenses are not attacked. As the reader can easily see, this is a very straightforward greedy algorithm, whose only advantage lies in its simplicity and fast speed, at the expense of quality of solution.

3.4.3. Sequential Reassignment

This is a modification of the sequential assignment algorithm. It is guaranteed to find a solution at least as good as that found by sequential assignment; in many cases it will find a better solution at the cost of some additional computations.

In sequential assignment, only airplanes that have yet to receive assignments are considered in each assignment cycle. The assignment in each cycle is chosen based on those assignments already made in previous cycles. It may be the case that the objective chosen for a given airplane might not have been chosen had it been known at the time that some other assignment was going to be made during a subsequent assignment cycle. For example, the airplane that was assigned to some objective in the first cycle based on the initial situation may be able to derive greater value from a different objective,given knowledge that an assignment was made to some defensive site during the fifth assignment cycle.

In sequential reassignment, maximum expected returns are calculated for both previously assigned and previously unassigned airplanes during each assignment cycle. Previously assigned airplanes are considered first. If at least one of these can achieve expected net improvement by changing its assignment, then the airplane with the greatest possible net improvement is reassigned during that cycle, values are updated accordingly, and the next assignment cycle begins. If no improvement is possible through reassignment, then the best previously unassigned airplane is chosen, as in sequential assignment.

This algorithm must converge, though again not necessarily to the optimal solution. However, it does allow some mistakes in airplane to objective assignments to be corrected.

3.4.4. Marginal Utility

Marginal utility algorithms are commonly used for resource allocation problems. Suppose that a number of decision makers are competing for the use of global resources and that the overall objective function is separable. Suppose further that for any given resource allocation, each decision maker can determine its own marginal utility (resource price), i.e., the price it would pay or would accept in exchange for an incremental unit of resource.

Assume that the utility function for each decision maker is concave. This means that the lower the resource allocation (supply), the higher the marginal utility (price). Then marginal utility theory states that a necessary and sufficient condition for a set of resource allocations to be optimal is that each decision maker has the same marginal utility, this value being referred to as the "equilibrium price." Marginal utility algorithms perturb the resource allocations systematically until the equilibrium price is achieved.

However, the airplane assignment problem does not satisfy the conditions required by conventional marginal utility algorithms. The objective function is not separable: a decision maker´s return may depend on the decisions made by other decision makers. Also, the resource to be allocated (airplanes) is discrete rather than continuous. This means that marginal utility cannot be defined incrementally as the derivative of the return function and that a decision maker's buying price and selling price for resource may differ. A version of marginal utility more suited to the airplane assignment problem is described in Appendix C.

3.4.5. AI Expert Planner

This system consists of seven experts working as a team in solving the posed mission planning problem. A description of the roles played by these experts is as follows:

1) Topological connectivity evaluator (or pocket divider): Topologically divides the enemy targets into pockets such that targets within each pocket are reachable from one another without going through major defensive threats.

2) Pocket Specialists: Each pocket is dynamically assigned a pocket specialist that comes up with the marginal airplane utility curve, i.e., determines the return R(N) extractable from the pocket given N units of resource (i.e., airplanes) and assuming that these airplanes units got safely inside the pocket.

3) Ring-Cutting Experts: There is one ring-cutting specialist per pocket. This expert evaluates various paths of getting inside the pocket (cutting through the ring), and chooses several (by N-best technique for thresholding, or clustering) of the most promising paths.

4) Cooperative Planning Expert: This expert receives the sets of best paths from each ring-cutting expert and combines them into cooperative ring-cutting plans. It chooses several of the best of these plans on the basis of their minimality and cooperation, i.e., needing to destroy as few defenses as possible to accomplish the cutting, and also on the basis of the importance of the pockets that will be accessible through these cuts.

5) Mission Planning Expert: Evaluates each of the cooperative ring-cutting schemes passed on by Expert 4 and optimizes the airplane assignments (i.e., how many defenses to attack, which of the targets, etc.) within each scheme using the marginal pocket utility curves passed on by pocket evaluators, and tries to choose the most valuable overall assignment.

6) Constraint Expert: For the proposed assignments, checks if all the constraints (like fuel constraint, time constraint, launcher constraint, weapons constraint, etc.) are satisfied. If this best plan is satisfiable, then we´re done. If not, it notes which constraint is not satisfied, imposes it on Expert 5 and tells Expert 5 to re-do his mission planning with the added constraint. Expert 5 may also call the pocket and ring-cutting planners for re-evaluation, if necessary. Thus, we´ll have iteration to a solution.

7) Meta-level Planner: Coordinates the performance of the above experts. Calls them in the right sequence and in parallel (like experts of Type 2 and 3). If Expert 6 comes up with new constraints, the Meta-level planner will call Expert 5 and the Pocket Experts and Ring-cutters for re-evaluation with the added constraints.

This present structure represents a transition between mathematical algorithms and flexible AI systems. It already has a fairly flexible inter-expert control structure, but this structure will get much more flexible as the system becomes more detailed. It also has a mixture of mathematical and rule-based experts, with experts number 1 and 2 being fairly mathematical, and experts 4 and 6 being strongly rule-based. We should also notice that this expert system has a strong distributed flavor. As we saw, there are several pocket and ring-cutting experts working in parallel with each other, one of each expert for every pocket. These experts come up with local attack plans, and these plans are then merged and reconciled by the mission planning expert and the constraint expert.

4. STREAMLINED DEPTH-FIRST DYNAMIC PROGRAMMING

Of all the algorithms we studied that guarantee optimality of solution, DFDP is the fastest and most efficient. However, the mission planning problem is intrinsically so extensive that even the best of optimal algorithms will probably be too slow for prompt large-scale open-loop mission planning. Thus, there is a need for faster near-optimal algorithms that would be close to the optimal algorithms in terms of results. Streamlined depth-first dynamic programming is such an algorithm.

In this section, we describe the basic formulation of the SDFDP algorithm and the software developed which implements it. We also compare this algorithm with the target assignment algorithm presented in the last section.

4.1 BASIC FORMULATION

SDFDP is a streamlined modification of the main DFDP algorithm in the sense that it has the same structure and the same general control flow, but it greatly reduces the state-space by combining many states into one super-state. In order to compensate for this simplification, it uses a number of assignment and reassignment sub-algorithms which allow it to find the best states within each super-state. These sub-algorithms are very similar in spirit to the Sequential Assignment and Reassignment algorithms, described in Section 3.

In this formulation each (super-) state corresponds to indicating which sets of defenses are to be attacked, how many sorties to send against each defense, and in what order to attack. Transition between states is accomplished by choosing one or several new sorties against some new defense.

Within each such super-state there is a lot of freedom in choosing which airplanes to use against each defense, which targets to attack, and which airplanes to use for that purpose. Notice that in this formulation each super-state acts both as a transition and a final state, depending on whether we plan to attack any more defenses (remember that in open-loop scenarios defenses should always be attacked before targets).

Within each super-state, assignments of particular airplanes to the chosen defenses is accomplished by a To-Defense Sequential Assignment algorithm whereas the selection of targets and airplanes to attack these targets is accomplished by a To-Target Sequential Assignment algorithm with Reassignment. As the names indicate, these algorithms represent direct applications of the Assignment and Reassignment algorithms described in the last section.

Finally, to take care of hasty commitment of valuable airplanes against defenses, the Defense-to-Target Reassignment algorithm is used. This algorithm allows reassignment of a committed airplane A from a defense D to a valuable target T, by providing a new airplane A1 to replace A in attacking defense D. This algorithm is also similar in spirit to the Sequential Reassignment algorithm.

The streamlined DFDP algorithm provides several capabilities to help reduce the computational burden through the use of human guidance. In particular, the human is allowed to specify which line of defense each defensive threat belongs to, to suggest the preferred order for the DFDP search, and to pre-specify limits for the number of airplanes to be used against enemy defenses. These guidelines greatly reduce the complexity of the feasible search space, and implementation of further such human guidance capabilities holds promise for further improvements.

4.2 SDFDP SOFTWARE STRUCTURE

The software which implements the SDFDP algorithm was written in the Sail programming language, developed by the Stanford Artificial Intelligence Laboratory. The programs were developed and implemented on a PDP-20, under the TOPS-20 operating system.

The code is organized into eighteen procedures, five of which perform the major functions of the algorithm: (1) dynamic programming, (2) defense assignment, (3) target assignment, (4) target reassignment, and (5) defense-to-target reassignment. The other procedures handle the bookkeeping or perform minor support functions.

A flowchart of the general structure of the software is shown in Figure 4-1. Initialization is performed first, and the user is prompted for all of the scenario´s data, which he enters interactively. The dynamic programming procedure is then called to find the probability of survival from every launcher (carrier) to each square on the board, and then to each target and defense.

The first super-state assignment is to attack no defenses, hitting only targets. Thus, the target assignment procedure is called to find the best launcher-target pair which yields the best score. This assignment is made temporarily until the target reassignment procedure is called to determine whether a better assignment can be made. The best assignment is then recorded. This process is repeated until all missiles (airplanes) have been assigned to targets.

The program then continues by assigning one missile to the first defense the user has deemed as most important. The defense assignment algorithm is then called to determine the best launcher to use against this defense. The target assignment procedure is then called, followed by the defense-to-target reassignment algorithm. This algorithm searches each empty launcher to determine whether, assuming it had an extra missile, it could get a higher score than what was computed by the target assignment. If a higher score could be obtained, and a non-empty launcher can hit one of the defenses the empty launcher previously hit, and still give an overall better score, the reassignment is made. Otherwise, the original assignment is recorded. The target reassignment procedure is then called. This process is repeated until all missiles have been assigned.

This plan and its final score are then compared with the previous plan, and the best plan is saved. The program continues to generate new plans, repeating the same process as above, and always saving the best

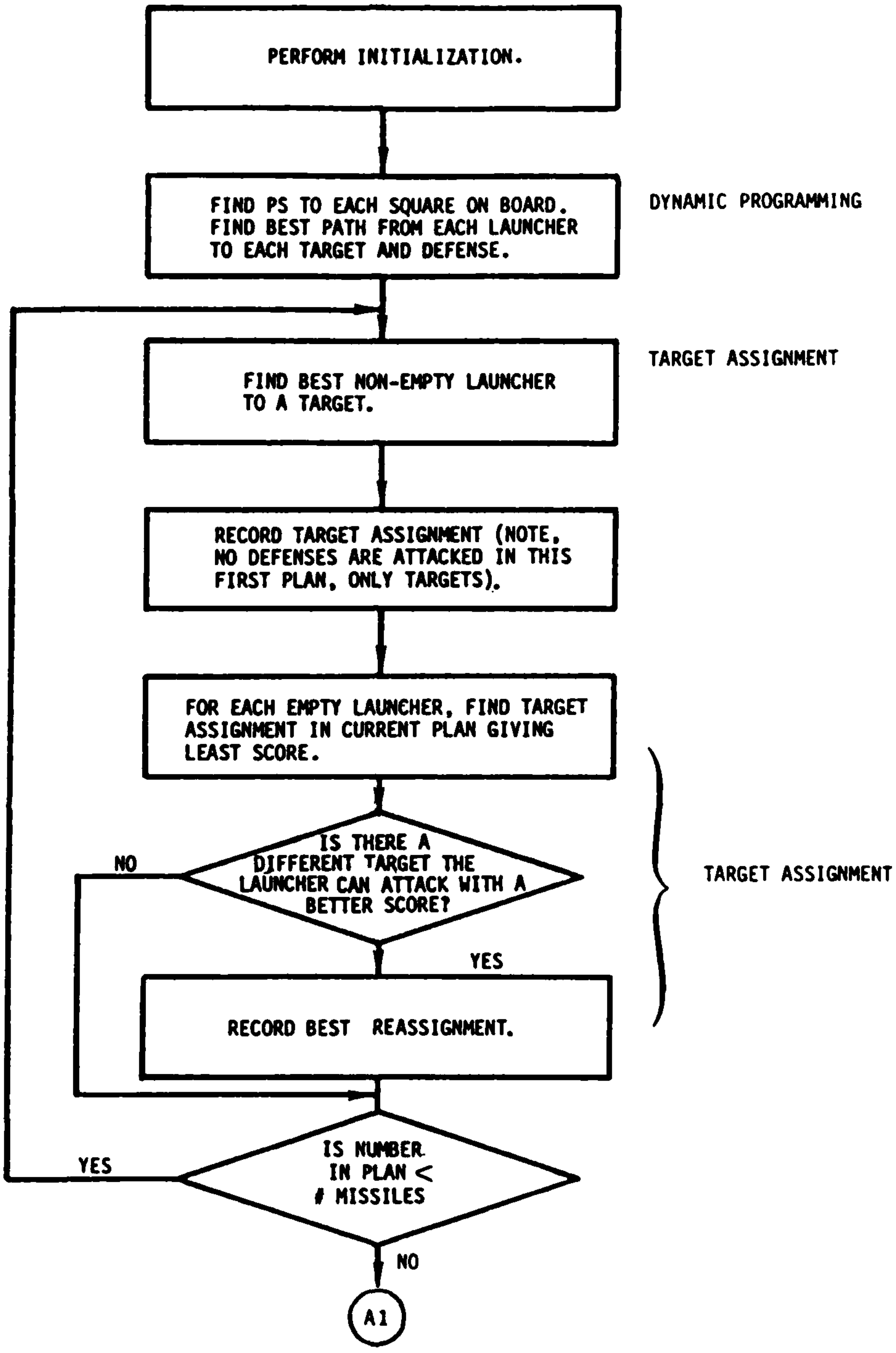

PERFORM INITIALIZATION.
FIND PS TO EACH SQUARE ON BOARD. FIND BEST PATH FROM EACH LAUNCHER TO EACH TARGET AND DEFENSE.
DYNAMIC PROGRAMMING
TARGET ASSIGNMENT
FIND BEST NON-EMPTY LAUNCHER TO A TARGET.
RECORD TARGET ASSIGNMENT (NOTE, NO DEFENSES ARE ATTACKED IN THIS FIRST PLAN, ONLY TARGETS).
FOR EACH EMPTY LAUNCHER, FIND TARGET ASSIGNMENT IN CURRENT PLAN GIVING LEAST SCORE.
NO
IS THERE A DIFFERENT TARGET THE LAUNCHER CAN ATTACK WITH A BETTER SCORE?
TARGET ASSIGNMENT
YES
RECORD BEST REASSIGNMENT.
YES
IS NUMBER IN PLAN < # MISSILES
NO
A1

Figure 4-1 Flowchart of SDFDP Software

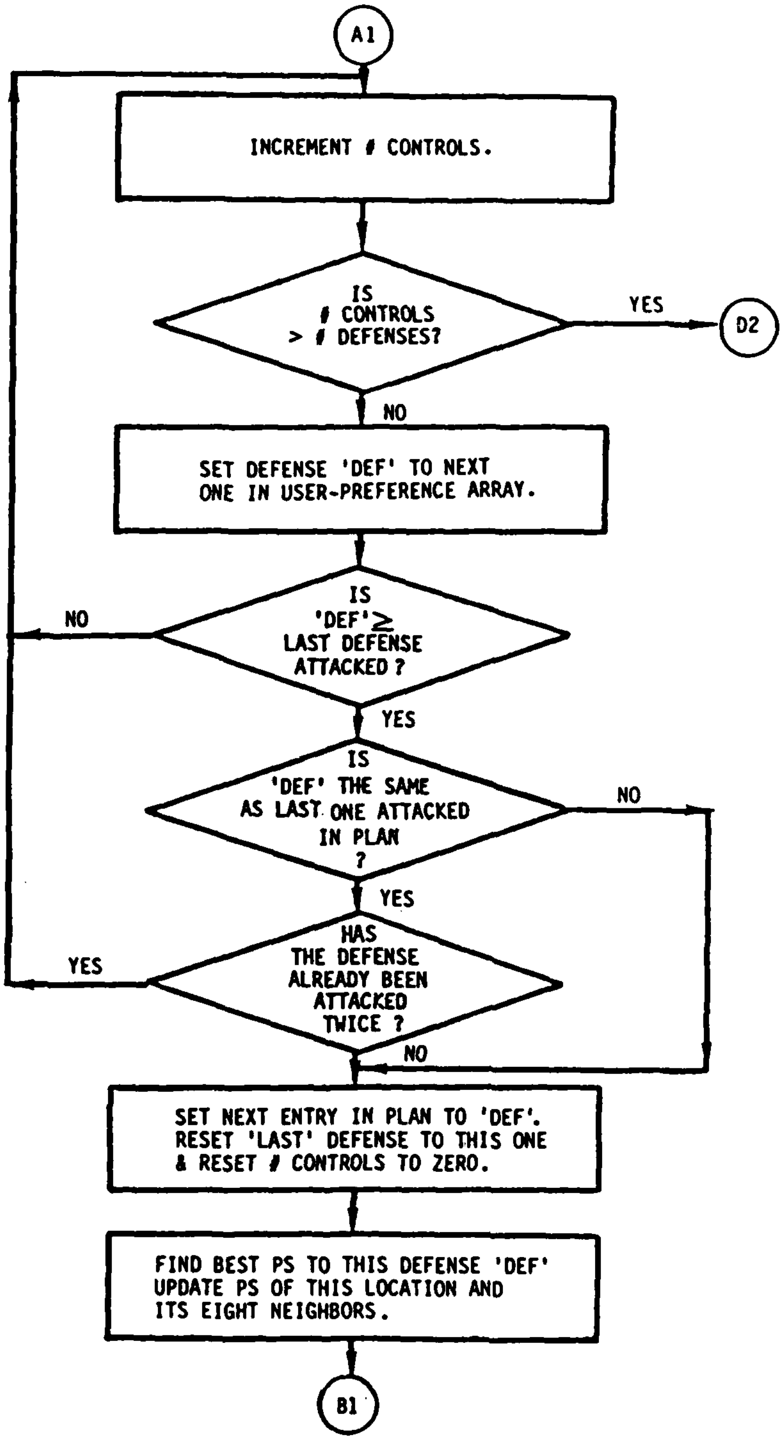

A1
INCREMENT # CONTROLS.
IS # CONTROLS > # DEFENSES?
YES
D2
NO
SET DEFENSE 'DEF' TO NEXT ONE IN USER-PREFERENCE ARRAY.
IS 'DEF'≥ LAST DEFENSE ATTACKED ?
NO
YES
IS 'DEF' THE SAME AS LAST ONE ATTACKED IN PLAN ?
NO
YES
HAS THE DEFENSE ALREADY BEEN ATTACKED TWICE ?
YES
NO
SET NEXT ENTRY IN PLAN TO 'DEF'. RESET 'LAST' DEFENSE TO THIS ONE & RESET # CONTROLS TO ZERO.
FIND BEST PS TO THIS DEFENSE 'DEF' UPDATE PS OF THIS LOCATION AND ITS EIGHT NEIGHBORS.
B1

Figure 4-1 Flowchart of SDFDP Software (Cont'd.)

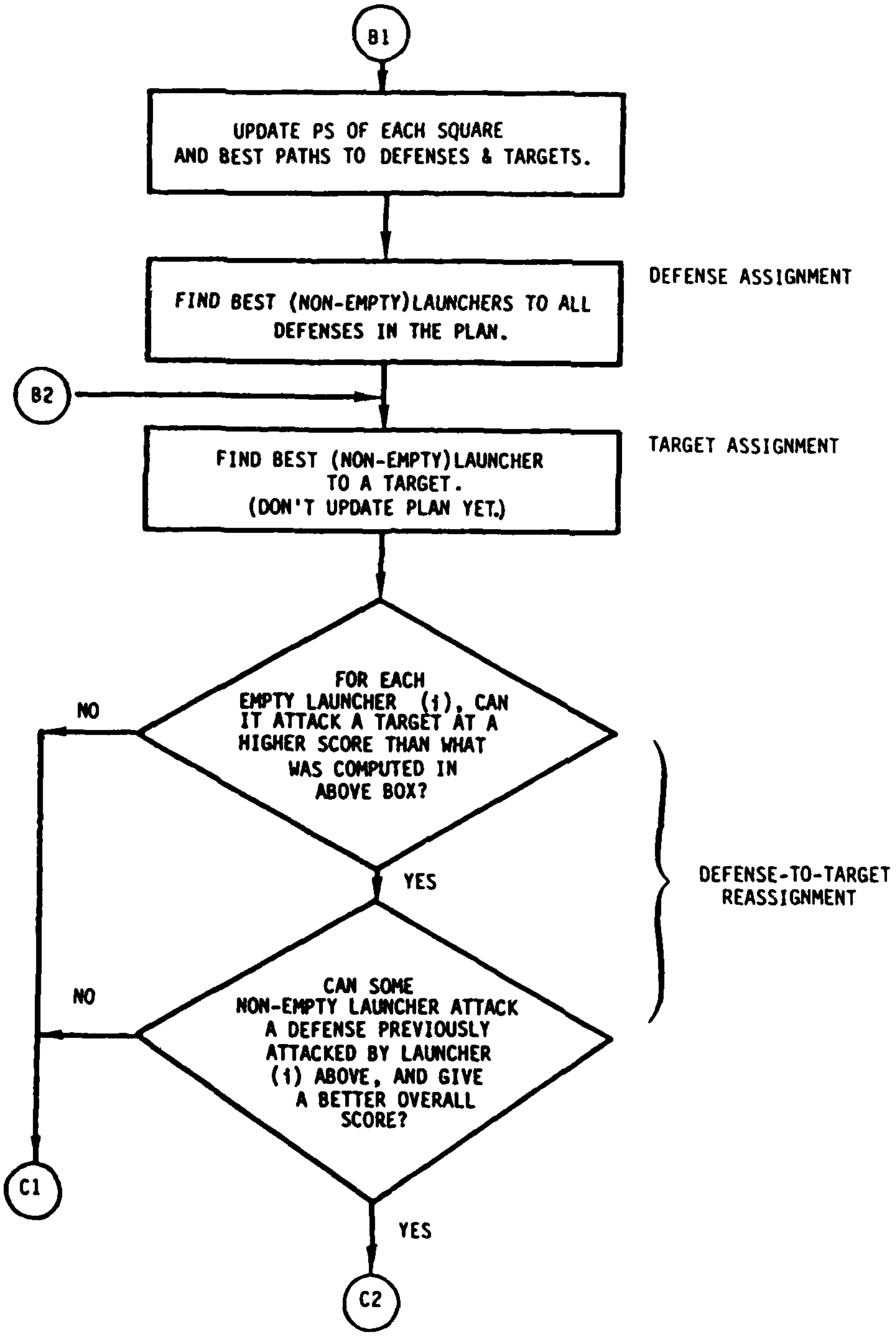

B1
UPDATE PS OF EACH SQUARE AND BEST PATHS TO DEFENSES & TARGETS.
FIND BEST (NON-EMPTY)LAUNCHERS TO ALL DEFENSES IN THE PLAN.
DEFENSE ASSIGNMENT
B2
FIND BEST (NON-EMPTY)LAUNCHER TO A TARGET. (DON'T UPDATE PLAN YET.)
TARGET ASSIGNMENT
FOR EACH EMPTY LAUNCHER (i), CAN IT ATTACK A TARGET AT A HIGHER SCORE THAN WHAT WAS COMPUTED IN ABOVE BOX?
NO
YES
DEFENSE-TO-TARGET REASSIGNMENT
CAN SOME NON-EMPTY LAUNCHER ATTACK A DEFENSE PREVIOUSLY ATTACKED BY LAUNCHER (i) ABOVE, AND GIVE A BETTER OVERALL SCORE?
NO
YES
C1
C2

Figure 4-1 Flowchart of SDFDP Software (Cont'd.)

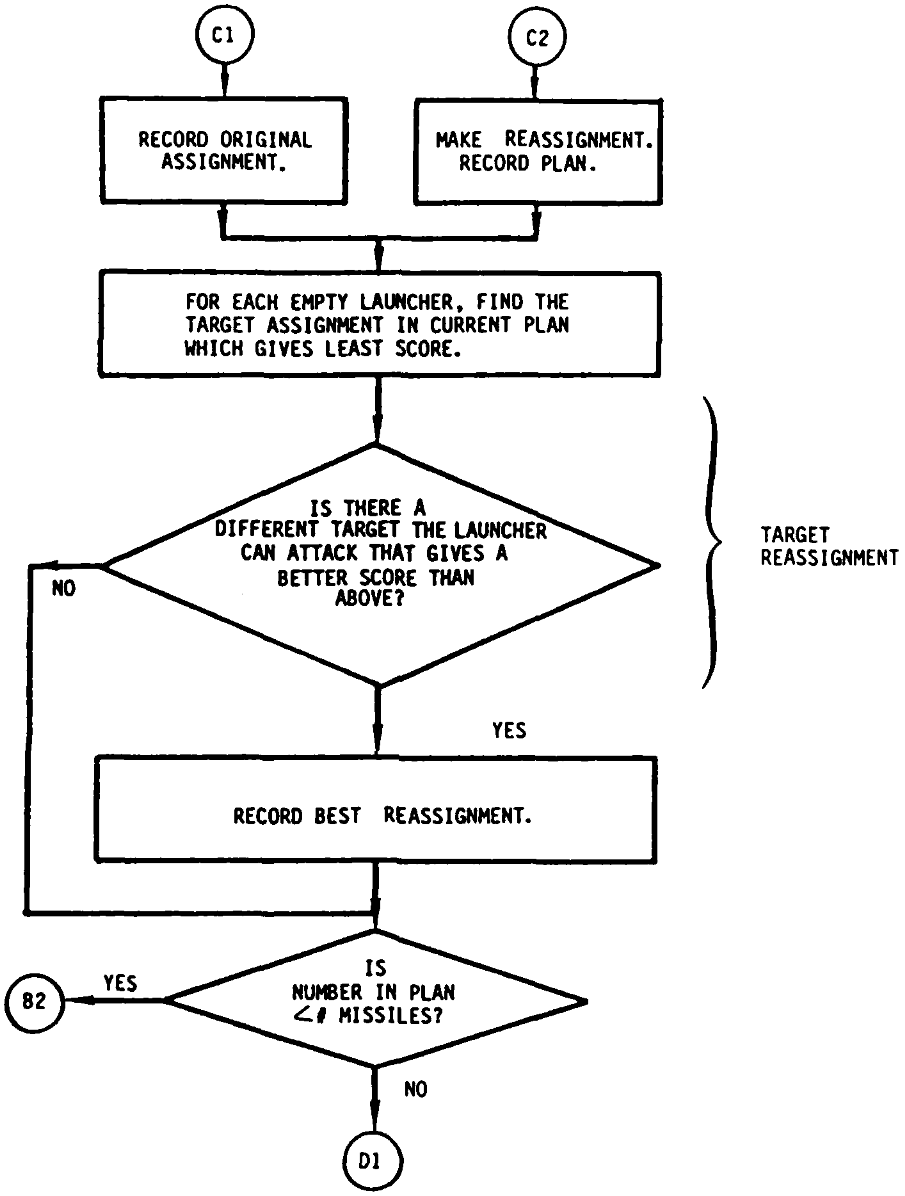

C1
C2
RECORD ORIGINAL ASSIGNMENT.
MAKE REASSIGNMENT. RECORD PLAN.
FOR EACH EMPTY LAUNCHER, FIND THE TARGET ASSIGNMENT IN CURRENT PLAN WHICH GIVES LEAST SCORE.
IS THERE A DIFFERENT TARGET THE LAUNCHER CAN ATTACK THAT GIVES A BETTER SCORE THAN ABOVE?
NO
YES
TARGET REASSIGNMENT
RECORD BEST REASSIGNMENT.
IS NUMBER IN PLAN ∠# MISSILES?
YES
B2
NO
D1

Figure 4-1 Flowchart of SDFDP Software (Cont'd.)

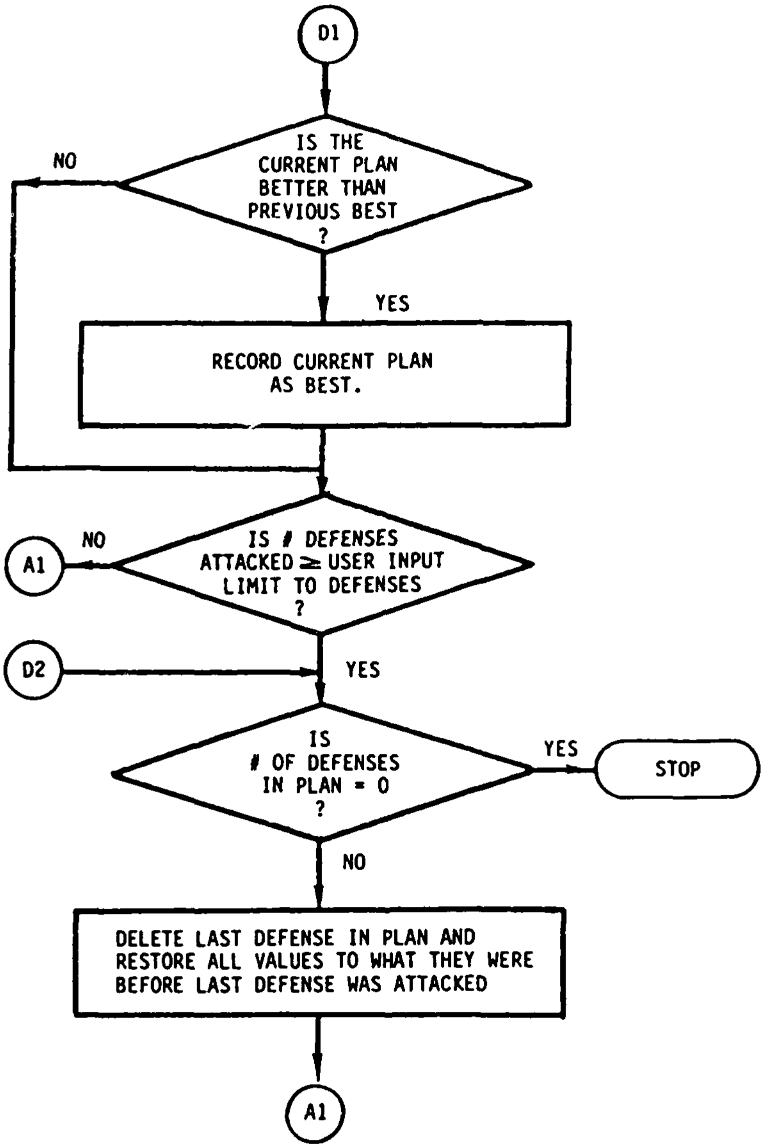

D1
IS THE CURRENT PLAN BETTER THAN PREVIOUS BEST ?
NO
YES
RECORD CURRENT PLAN AS BEST.
IS # DEFENSES ATTACKED ≥ USER INPUT LIMIT TO DEFENSES ?
NO
A1
D2
YES
IS # OF DEFENSES IN PLAN = 0 ?
YES
STOP
NO
DELETE LAST DEFENSE IN PLAN AND RESTORE ALL VALUES TO WHAT THEY WERE BEFORE LAST DEFENSE WAS ATTACKED
A1

Figure 4-1 Flow Chart of SDFDP Software (Cont'd.)

one. The way it generates these new plans is in the spirit of a depth-first search, with two practical heuristics incorporated to reduce the problem size. These heuristics are: (1) a given defense can only be attacked twice, and (2) defenses must be attacked in increasing order, i.e., Dl before D2, etc. Also, the order in which the defenses are chosen depends upon the order in which the user specifies. Thus, if there are two defenses (Dl, D2) and the limit-to-defense is three, the following plans will be generated sequentially.

- Plan 1: no defenses, the rest targets
- Plan 2: Dl, targets
- Plan 3: Dl, Dl, targets
- Plan 4: Dl, Dl, D2, targets
- Plan 5: Dl, D2, targets
- Plan 6: Dl, D2, D2, targets
- Plan 7: D2, targets
- Plan 8: D2, D2, targets

This sequence of plans assumes that the preference array for defenses is (Dl, D2, D3).

4.3 IMPLEMENTATION DETAILS

This section describes several implementation details and capabilities of the SDFDP software.

4.3.1 Input Data

All of a scenario´s input data is entered by the user at the start of each run. The user is therefore able to enter any scenario that he wishes. The data needed is: board size, defense locations, target locations and values, probability of kill of planes (missiles), carrier (launcher) location and number of planes on each, and the amount of fuel each plane has onboard.

Weapons on different launchers are allowed to have different probabilities of kill (PK) against each target and defense. Currently, all weapons on the same launcher must have the same kill capability. However, if the scenario calls for different capabilities, the user only needs to group like weapons on the same launcher and then colocate different launchers. Multiple launchers located at the same position may be thought of as one launcher.

Currently, the PK of defense against missile is fixed at .3 for missiles flying directly over the defense, and .2 for neighboring locations to the defense. The capability for variable PK would require only slight modifications.

4.3.2 Scenario Size

Theoretically, any size scenario could be handled by the algorithm, by a simple re-dimensioning of arrays. Realistically speaking, the user is limited by both storage cost and computing time. Computing time varies with board size, number of targets, and especially with the number of defenses(since they comprise the super-state defined in Section 4.1). A detailed timing study has not been performed, but as a rule-of-thumb, any scenario with a board-size of larger than 13 X 13 and more than about 8 defenses or targets, would probably taking more than 15 CPU minutes. The scenario shown in Figure 4-2 took about 4.8 CPU minutes. More timing statistics are given in Section 5.

4.3.3 Output

The best plan found is outputted to the terminal screen or a file. The output is simply a list of airplanes (missiles), their final destinations, and the score obtained. Sample outputs are scattered throughout this report.

4.4 EXAMPLES

This section presents two examples of the SDFDP algorithm. Both of them show the complicating factors involved in even very small scenarios. These factors make trying to generate an optimal plan "intuitively" virtually impossible.

The first example is shown in Figure 4-2. Note the position of the two carriers. Now intuitively, a human would guess that each carrier would most probably send its weapons to the defenses and targets closest to themselves. But instead, we see in the solution that Carrier 1 attacks both D1 and D2 and does not attack even one of the targets only one move away from it. Instead, it sends its final plane down around to T9. We see, after careful examination, that this is due to the high value of the targets on the right half of the board. But this example, we believe, points out the necessity of a near-optimal automated algorithm, such as SDFDP.

Next, we consider a more complex example. The scenario is shown in Figure 4-3(a) along with the target statistics. In this example, the launch platforms are air bases, and the targets are bridges, tanks, or engineering companies. We know that the enemy is situated at the upper left corner of the board, and not expected to reach the bridges until the next day. We are planning today's mission and therefore we can assign values as appropriate. Since the bridges are not considered to be important yet, we assign them a lower value than the tanks and especially the engineering company. We assign a very high value to the engineering company, knowing that during our next mission, we will be attacking the bridges, and the company will be vital to repairing them.

The two air bases have five weapons on each. The weapon statistics are shown in Figure 4-3(b). (Note that in implementing the different kill probabilities on each air base, we actually had to colocate two air bases at each location, as mentioned in Section 4.3.1.)

The plan generated by SDFDP is shown in Figure 4-3(c).

This example brings out at least two important points. First, it would be virtually impossible for a human to be able to quickly plan such a mission, taking into account the number of complexities inherent in the problem. Second, the SDFDP algorithm is general enough to handle any scenario for any type of mission, provided a human is able to assign numerical values to the targets, and a probability of kill to the

	1	2	3	4	5	6	7	8
1		T2	D7	T4				T8
2	T1	T3		T5	D5		T6	
3	D6		C1		D4			T9
4					D3		T7	
5			D1				D2	
6								
7								C2

TARGET	VALUE
1	300
2	300
3	300
4	300
5	300
6	450
7	470
8	400
9	400

FUEL = 20

CARRIER	# PLANES
1	5
2	3

PLAN

PLANE #	FROM CARRIER #	TO DEFENSE #	LEAVES IT WITH PE
1	1	1	.44
2	1	1	.09
3	1	2	.47
4	1	2	.12

PLANE #	FROM CARRIER #	TO TARGET #	FOR A SCORE OF
6	2	7	348.69
7	2	6	333.86
8	2	8	296.76
5	1	9	274.57

TOTAL SCORE = 1,253.88

Figure 4-2 Example of SDFDP

	1	2	3	4	5	6	7	8	9	10	11
1	T7		T8								
2											
3		T3		T6							
4	D5		T4	D6							
5			T5								
6			D4								
7											
8						T2					
9					D2	T1	D3				
10						D1					
11											
12			A1								A2

TARGET DATA

TYPE	TARGET #	VALUE
BRIDGES	1	150
	2	130
TANKS	3	180
	4	200
ENGINEERING CO.	5	510
	6	500
OTHER	7	100
	8	100
	TOTAL VALUE	1,870

Figure 4-3(a) Planning Example

RESOURCES/CAPABILITY

AIR BASE	WEAPONS
1	1 PGM, 4 BOMBS
2	1 PGM, 4 BOMBS

PROBABILITY OF KILL:

		TARGET							
		1	2	3	4	5	6	7	8
		(BRIDGES)		(TANKS)		(ENG. CO.)		(OTHER)	
WEAPONS:	PGM'S	.9	.9	.7	.7	.5	.5	.7	.7
	BOMBS	.5	.5	.7	.7	.9	.9	.7	.7

		DEFENSE					
		1	2	3	4	5	6
		(SAM'S)				(MAN)	
WEAPONS:	PGM'S	.9	.9	.9	.9	.5	.5
	BOMBS	.5	.5	.5	.5	.9	.9

FUEL:

EACH WEAPON INITIALLY HAS 25 UNITS OF FUEL. (IT TAKES 2 UNITS TO MOVE HORIZONTALLY/VERTICALLY; 3 TO MOVE DIAGONALLY.)

Figure 4-3(b) Planning Example

AIR BASE	WEAPON	PLAN TO DEFENSE	LEAVES IT WITH PE
1	PGM	4 (SAM)	.496
2	BOMB	6 (MAN)	.496
2	BOMB	6 (MAN)	.154

AIR BASE	WEAPON	TO TARGET	FOR A SCORE OF
2	BOMB	6 (ENG. CO.)	422.76
2	BOMB	5 (ENG. CO.)	361.00
1	BOMB	4 (TANKS)	113.01
1	BOMB	5 (ENG. CO.)	105.47
1	BOMB	3 (TANKS)	81.37
2	PGM	2 (BRIDGE)	74.88
1	BOMB	6 (ENG.CO.)	58.83
		TOTAL SCORE =	1,217.32
		TOTAL POSSIBLE SCORE =	1,870.00

Figure 4-3(c) Planning Example

defenses.

4.5 COMPARISONS BETWEEN SDFDP AND THE TARGET ASSIGNMENT ALGORITHM

In this section we consider several examples of plans generated by the SDFDP algorithm and compare the performance of these plans with that of the plans generated by the Target Assignment (TA) algorithm.

The purpose of this comparison is two-fold. First, we want to see how well SDFDP solves mission planning problems in comparison to the traditional mission planning algorithms. Second, we want to see how much advantage can be gained from sending some airplanes to suppress enemy defenses as opposed to sending all the airplanes against enemy targets, as most of the models existing in literature do.

We have also informally compared our algorithmic plans against plans generated manually by human players. In all our comparison tests SDFDP generated what appeared to be optimal solutions, and these solutions were consistently better than not only the TA algorithm but also humans. Figure 4-4 gives computer printouts for five such comparison tests.

Figure 4-4(a) offers the TA solution for the scenario that was considered in detail in the preceding sections (see Examples 1 and Figure 2-1). The score obtained by TA is 802 points, which is significantly smaller than 1,097 points obtained by DFDP (see Figure 3-1).

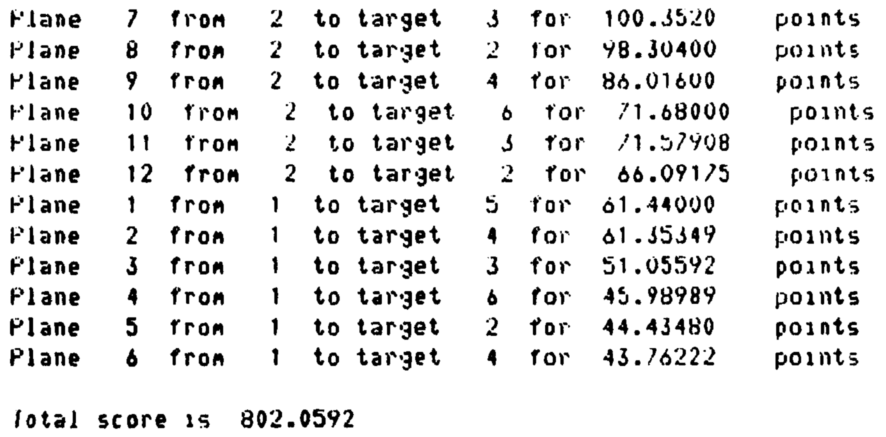
Figures 4-4(b),(c) and (d) refer to modifications of the scenario from Example 1. In Figure 4-4(b), all the enemy forces are the same as in Example 1 except that there are 2 carriers with 6 planes on each, rather than 1 carrier with 12 planes as before. These carriers are situated in different corners of the board, and, thus, have different capabilities.

The optimal plan, generated by SDFDP, reflects this new situation. Because airplanes from carrier C2 now don´t have enough fuel to attack targets T5 and T6 by way of D2, we had to send one sortie to take out defense D5, and that became the entrance point for attacking T5 and T6.

Of course, the optimal result of 1011 points is not as high as the optimal result of 1097 for the scenario from Figure 3-1.

We can also see that the TA gives a result of almost 200 points less. This superiority of SDFDP over TA was demonstrated by all our test examples. Figure 4-4(e) represents a typical example. It deals with a position significantly different from the rest, but, again, attacking enemy defenses allows an improvement in results.

The magnitude of such improvement varies significantly from one example to another. There are many factors that determine the amount of improvement, one of them being maneuverability of our forces. Figures 4-4(c) and (d) illustrate this point. They both refer to a position derived from position 4-4(b) through minor changes, which amount to opening small holes in the enemy defensive line but allowing our forces to take advantage of these holes only by flying very roundabout routes. In Figure 4-4(c), our airplanes were given 20 units of fuel, which was enough for them to reach their targets along these roundabout routes. As you can see, in this situation little advantage can be derived from attacking enemy defenses before going against targets. On the other hand, when the airplanes were given only 15 units of fuel (as in Figure 4-4(d), they couldn't take advantage of the holes in enemy lines and had to fly over several enemy defenses. In this case blowing holes by attacking enemy defenses becomes much more essential and the results in Figure 4-4(d) confirm this observation.



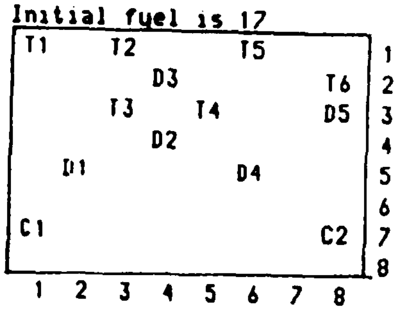
```
Plane   7  from   2  to target   3  for  100.3520    points
Plane   8  from   2  to target   2  for  98.30400    points
Plane   9  from   2  to target   4  for  86.01600    points
Plane   10  from   2  to target   6  for  71.68000    points
Plane   11  from   2  to target   3  for  71.57908    points
Plane   12  from   2  to target   2  for  66.09175    points
Plane   1  from   1  to target   5  for  61.44000    points
Plane   2  from   1  to target   4  for  61.35349    points
Plane   3  from   1  to target   3  for  51.05592    points
Plane   4  from   1  to target   6  for  45.98989    points
Plane   5  from   1  to target   2  for  44.43480    points
Plane   6  from   1  to target   4  for  43.76222    points

Total score is  802.0592
```

(a) TA Solution for Position in Figure 2-1

Figure 4-4 Comparisons of SDFDP and Target Assignment (TA) Algorithms

POSITION

Initial fuel is 17



```
T1   100      0        T2   300      0
T3   350      0        T4   300      0
T5   150      0        T6   200      0
D1   .99               D2   .99
D3   .99               D4   .99
D5   .99
C1   .99 Y   A1  A2    A3  A4  A5  A6
C2   .99 Y   A7  A8    A9  A10 A11 A12
```

SDFDP PLAN

```
Plane   1   from   1   to defense   1
Plane   2   from   1   to defense   2
Plane   3   from   1   to defense   3
Plane   7   from   2   to defense   5
Plane   8   from   2   to target    3   for   237.4645    points
Plane   9   from   2   to target    4   for   203.5410    points
Plane   4   from   1   to target    2   for   133.3527    points
Plane   10   from   2   to target    6   for   121.3681    points
Plane   11   from   2   to target    5   for   99.80928    points
Plane   12   from   2   to target    3   for   76.35195    points
Plane   5   from   1   to target    2   for   74.07623    points
Plane   6   from   1   to target    4   for   65.44453    points

Total score is   1011.408
```

TA PLAN

```
Plane   7   from   2   to target   3   for   100.3520    points
Plane   8   from   2   to target   4   for   98.30400    points
Plane   1   from   1   to target   2   for   98.30400    points
Plane   9   from   2   to target   6   for   81.92000    points
Plane   10   from   2   to target   5   for   76.80000    points
Plane   11   from   2   to target   3   for   71.57908    points
Plane   12   from   2   to target   4   for   66.09175    points
Plane   2   from   1   to target   2   for   66.09175    points
Plane   3   from   1   to target   3   for   51.05592    points
Plane   4   from   1   to target   2   for   44.43480    points
Plane   5   from   1   to target   1   for   40.96000    points
Plane   6   from   1   to target   4   for   38.88045    points

Total score is   834.7738
```

(b) Case 2

Figure 4-4 Comparisons of SDFDP and Target Assignment (TA) Algorithms

```
Initial fuel is 20
 _______________________
|T1    T2       T5      |1
|         D3          T6|2
|      T3    T4       D5|3
|         D2            |4
|                       |5
|      D1       D4      |6
|                       |7
|         C1 C2         |8
| 1  2  3  4  5  6  7  8|
 -----------------------
```

```
T1   100      0      T2   300      0
T3   350      0      T4   300      0
T5   150      0      T6   200      0
D1   .99             D2   .99
D3   .99             D4   .99
D5   .99
C1   .99 Y   A1 A2   A3 A4 A5 A6
C2   .99 Y   A7 A8   A9 A10 A11 A12
```

```
Plane   1  from   1  to defense   3
Plane   2  from   1  to defense   5
Plane   3  from   1  to defense   5
Plane   7  from   2  to target   3  for  199.2704    points
Plane   4  from   1  to target   2  for  170.8032    points
Plane   8  from   2  to target   4  for  167.6711    points
Plane   9  from   2  to target   6  for  151.3583    points
Plane   10  from   2  to target   5  for  94.23957    points
Plane   11  from   2  to target   3  for  85.81699    points
Plane   5  from   1  to target   1  for  80.00000    points
Plane   12  from   2  to target   4  for  73.95911    points
Plane   6  from   1  to target   2  for  73.55742    points

Total score is  1096.676
```

```
Plane   7  from   2  to target   3  for  179.2000    points
Plane   1  from   1  to target   2  for  153.6000    points
Plane   8  from   2  to target   4  for  122.8800    points
Plane   9  from   2  to target   3  for  87.44960    points
Plane   10  from   2  to target   6  for  81.92000    points
Plane   2  from   1  to target   1  for  80.00000    points
Plane   11  from   2  to target   5  for  76.80000    points
Plane   3  from   1  to target   2  for  74.95680    points
Plane   12  from   2  to target   4  for  72.54835    points
Plane   4  from   1  to target   4  for  42.83255    points
Plane   5  from   1  to target   3  for  42.67541    points
Plane   6  from   1  to target   6  for  42.31987    points

Total score is  1057.183
```

(c) Case 3

Figure 4-4 Comparisons of SDFDP and Target Assignment (TA) Algorithms

```
Initial fuel is 15
+----------------------------+
|T1    T2        T5          | 1
|         D3              T6 | 2
|      T3     T4          D5 | 3
|         D2                 | 4
|                            | 5
|      D1          D4        | 6
|                            | 7
|         C1  C2             | 8
| 1  2  3  4  5  6  7  8     |
+----------------------------+
```

```
T1   100      0        T2   300      0
T3   350      0        T4   300      0
T5   150      0        T6   200      0
D1   .99               D2   .99
D3   .99               D4   .99
D5   .99
C1   .99 Y   A1 A2     A3 A4 A5 A6
C2   .99 Y   A7 A8     A9 A10 A11 A12
```

```
Plane   1  from   1  to defense   1
Plane   2  from   1  to defense   1
Plane   7  from   2  to defense   2
Plane   8  from   2  to defense   2
Plane   3  from   1  to target   3  for  210.5028    points
Plane   9  from   2  to target   4  for  168.4397    points
Plane   4  from   1  to target   2  for  107.8014    points
Plane   5  from   1  to target   3  for  83.89872    points
Plane   10  from   2  to target   4  for  73.86659    points
Plane   6  from   1  to target   2  for  69.06426    points
Plane   11  from   2  to target   5  for  61.44000    points
Plane   12  from   2  to target   6  for  52.42880    points

Total score is  827.4423
```

```
Plane   1  from   1  to target   3  for  114.6880    points
Plane   7  from   2  to target   4  for  78.64320    points
Plane   2  from   1  to target   3  for  77.10704    points
Plane   8  from   2  to target   5  for  61.44000    points
Plane   9  from   2  to target   4  for  58.02736    points
Plane   10  from   2  to target   6  for  52.42880    points
Plane   3  from   1  to target   3  for  51.84060    points
Plane   11  from   2  to target   4  for  42.81583    points
Plane   12  from   2  to target   6  for  38.68491    points
Plane   4  from   1  to target   3  for  34.85347    points
Plane   5  from   1  to target   2  for  32.21226    points
Plane   6  from   1  to target   4  for  31.59192    points

Total score is  674.3334
```

(d) Case 4

Figure 4-4 Comparisons of SDFDP and Target Assignment (TA) Algorithms

```
Initial fuel is 15
 ________________________________
|          T4                    | 1       T1  100    0      T2  150    0
|    T2          D3 T5     D5    | 2       T3  200    0      T4  250    0
| T1    T3              T6       | 3       T5  200    0      T6  150    0
| D1    D2          D4           | 4       D1  .99           D2  .99
|                                | 5       D3  .99           D4  .99
|       C2                       | 6       D5  .99
| C1                    C3       | 7       C1  .99 Y  A1 A2   A3 A4
 --------------------------------        C2  .99 Y  A5 A6   A7 A8
  1  2  3  4  5  6  7  8                   C3  .99 Y  A9 A10  A11 A12
```

```
Plane   9  from  3  to defense  2
Plane   1  from  1  to defense  2
Plane   2  from  1  to defense  3
Plane   5  from  2  to target   4  for  171.4040   points
Plane  10  from  3  to target   3  for  151.3583   points
Plane   6  from  2  to target   2  for  113.5187   points
Plane   7  from  2  to target   5  for  111.4922   points
Plane  11  from  3  to target   6  for  76.80000   points
Plane   8  from  2  to target   1  for  60.54331   points
Plane   3  from  1  to target   4  for  53.38339   points
Plane   4  from  1  to target   5  for  44.80173   points
Plane  12  from  3  to target   6  for  37.47840   points

Total score is  820.7800
```

```
Plane   5  from  2  to target   4  for  81.92000   points
Plane   9  from  3  to target   5  for  81.92000   points
Plane  10  from  3  to target   3  for  81.92000   points
Plane  11  from  3  to target   6  for  76.80000   points
Plane  12  from  3  to target   2  for  61.44000   points
Plane   6  from  2  to target   4  for  55.07646   points
Plane   7  from  2  to target   3  for  48.36557   points
Plane   8  from  2  to target   4  for  37.02900   points
Plane   1  from  1  to target   2  for  36.27418   points
Plane   2  from  1  to target   1  for  35.84000   points
Plane   3  from  1  to target   5  for  30.95396   points
Plane   4  from  1  to target   3  for  28.55503   points

Total score is  656.0942
```

(e) Case 5

Figure 4-4 Comparisons of SDFDP and Target Assignment (TA) Algorithms

5. PARTITIONING

In this section, we explore the idea of partitioning the problem, both spatially and temporally. Spatial partitioning may be necessary to reduce computing time and enable more realistically sized scenarios to be studied. By temporal partitioning of the problem, we mean allowing multiple "attack waves" to be generated, in which particular defenses may be able to reconstitute in time for the next attack wave. This approach also allows more realistic situations to be examined.

5.1 SPATIAL PARTITIONING

We approached the problem of spatial partitioning in two ways. In the first approach, presented in Section 5.1.1, we split the scenario of interest into various regions and treat each region as a separate problem. In Section 5.1.2, we discuss the issue of grouping like targets in the scenario together. Both approaches allow larger scenarios to be run, using less computing time. This result will be detailed in the outline of the first approach.

5.1.1 Partitioning the Board

Our heuristic modification to the problem of finding a near (or possibly true) optimal plan to the dynamic sequence assignment is as follows. Divide the board into two or more overlapping regions, where each region includes all the launch platforms, but each defense/target is in exactly ONE of the regions. Then use an optimization algorithm, such as SDFDP, to find the best plan for each region separately, varying the total number of weapons allowed between the launchers per region. For example, if in the original problem there were two launchers, each with two weapons onboard, we would split the board into at least two regions and make several runs on each region, each time allowing a different number of weapons (from 0-4) to be used on that area. After running the algorithm on all the partitions, sum the scores of those regions whose total number of weapons when taken together sum to the number of weapons allowed in the original problem. In the above example, if we partitioned the board into two regions, we would sum the target scores obtained from allowing 0 weapons in Region 1 (R1), and 4 in Region 2 (R2), 1 in R1 and 3 in R2, 2 in R1 and 2 in R2, 3 in R1 and 1 in R2, and finally 4 in R1 and 0 in R2. From these total scores, the maximum target score would yield the best plan.

Note that as the number of launch platforms in the problem increases, the complexity of this approach grows raoidly. When we vary the number of weapons allowed per region, we also must decide how many to allow on each launcher. Thus, if for a given run; we're allowing three weapons to be used on a region with two launchers, there are 4 possibilities as to how to distribute these three among the launchers (0 on L1 and 3 on L2, 1 on L1 and 2 on L2, 2 on L1 and 1 on L3, or 3 on L1 and 0 on L2).

At first glance, the total number of runs of the algorithm that must be performed seems unreasonably large. Our claim, however, is that, by being "smart" about the problem, the number of times the algorithm must be run can be decreased to a very reasonable and feasible number. For example, if a given region has only a few low-valued targets or defenses, while another region is very target rich, then it is not necessary to run the case where <u>all</u> weapons are allowed to attack the first region. As a user becomes more familiar with using this heuristic approach, he will get a "feel" for what is reasonable or unreasonable to do.

We implemented this approach using our SDFDP software. The next section describes a feature of our SDFDP software that was a necessary addition in the implementation of the approach. An example of the approach follows.

5.1.1.1 Non-rectangular boards

A first step in implementing spatial partitioning was adding the capability to the software of handling non-rectangular boards. All of the examples presented thus far in the report have been with scenarios laid out on a rectangular grid. To be able to partition the board into various regions, it was necessary that these regions be of any shape.

Rectangular regions would be too constraining, especially if launch platforms were located on either side of the board. Also, the ability to outline only the essential areas of the board saves computing time. Examples of non-rectangular boards are illustrated in the next section.

5.1.1.2 Example of Partitioning

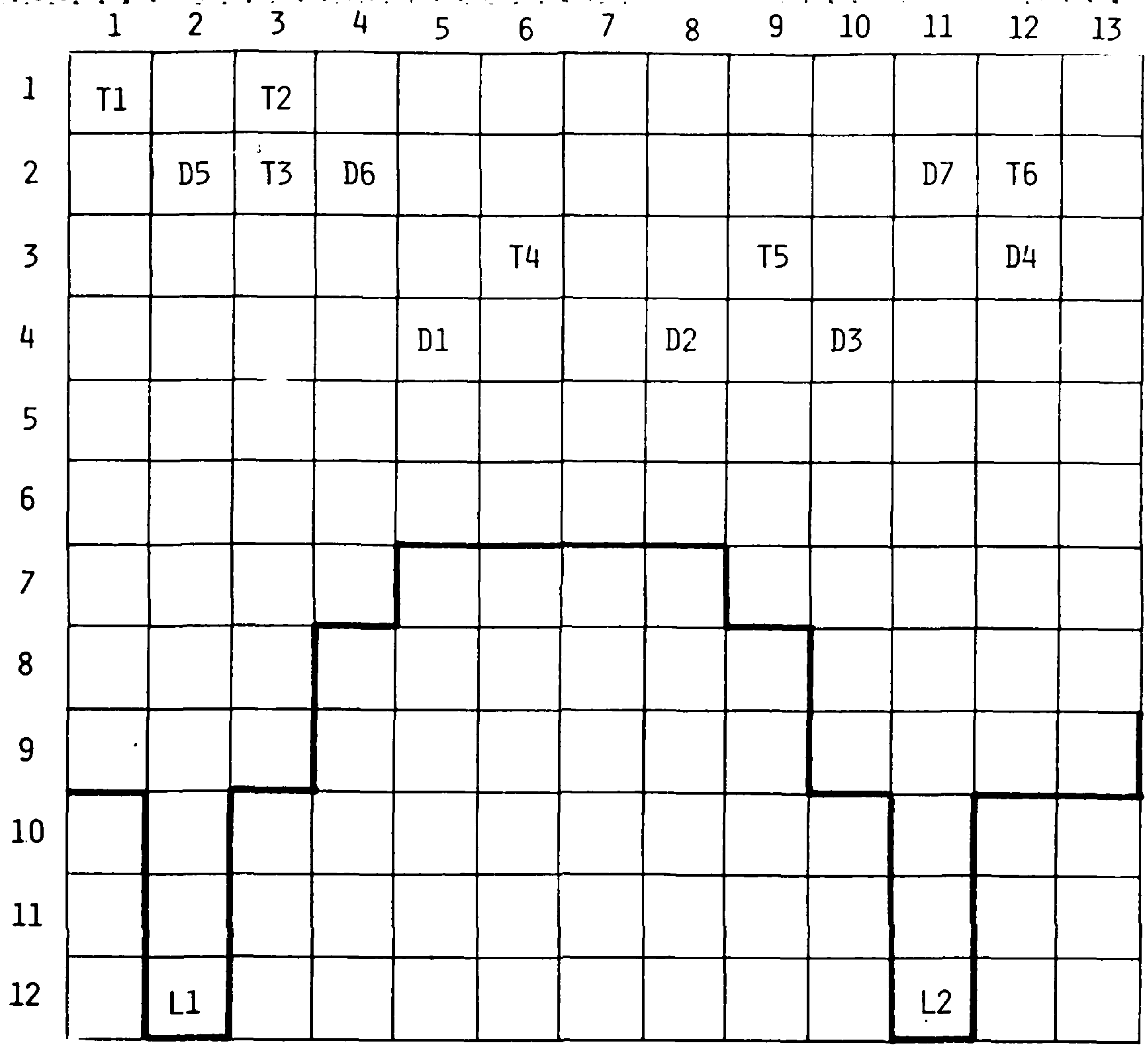
Consider the scenario shown in Figure 5-1(a). The target and resource statistics are as shown below the board. Running the SDFDP software on the entire region outlined in bold print took approximately 16 CPU minutes on a PDP-20, under the TOPS-20 Operating System. The plan generated is shown in Figure 5-1(b).

Now, using spatial partitioning as described in Section 5.1.1, we divide the board into two regions, shown in Figure 5-2 (a) and (b), outlined in bold print. The results of running the program separately on Partitions A and B are shown in the graph of Figure 5-2(c). Because of the number of targets in Partition A compared to the number in Partition B, it was reasonable to begin by allowing 3 weapons in Partition A. We thought that anything lower would certainly not be optimal. We then continued to increase the number of weapons *allowed* until we reached 7. We believed that anything higher (that is 8) would also be unreasonable. We then repeated the process for Partition B. Then, by summing the two curves A and B, it is easy to see the maximum score occurs at 6 weapons on Partition A and 2 on B. The total score was 639.70, which is the same as that found by running the entire region at once. The total computing time using this approach was 54 CPU seconds per sample point. Thus, a total of 9 CPU minutes were required to generate the graph of Figure 5-2(c). This results in a 44% decrease in computing time.

5.1.2 Target Grouping

In this section, we discuss the issue of target grouping in order to enhance optimum dynamic sequence assignment via dynamic programming or other mathematical optimization methods. The grouping is based on a set of heuristics that groups targets with similar characteristics together and treats them as a set of super-targets. Assignment is then done over the super-targets and final refinement is made via

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	T1		T2										
2		D5	T3	D6							D7	T6	
3						T4			T5			D4	
4					D1			D2		D3			
5													
6													
7													
8													
9													
10													
11													
12		L1									L2		



TARGET	VALUE
1	200
2	300
3	100
4	200
5	200
6	300

FUEL = 32

LAUNCHER	# MISSILES
1	4
2	4

Figure 5-1(a) Example of Partitioning -- Entire Board

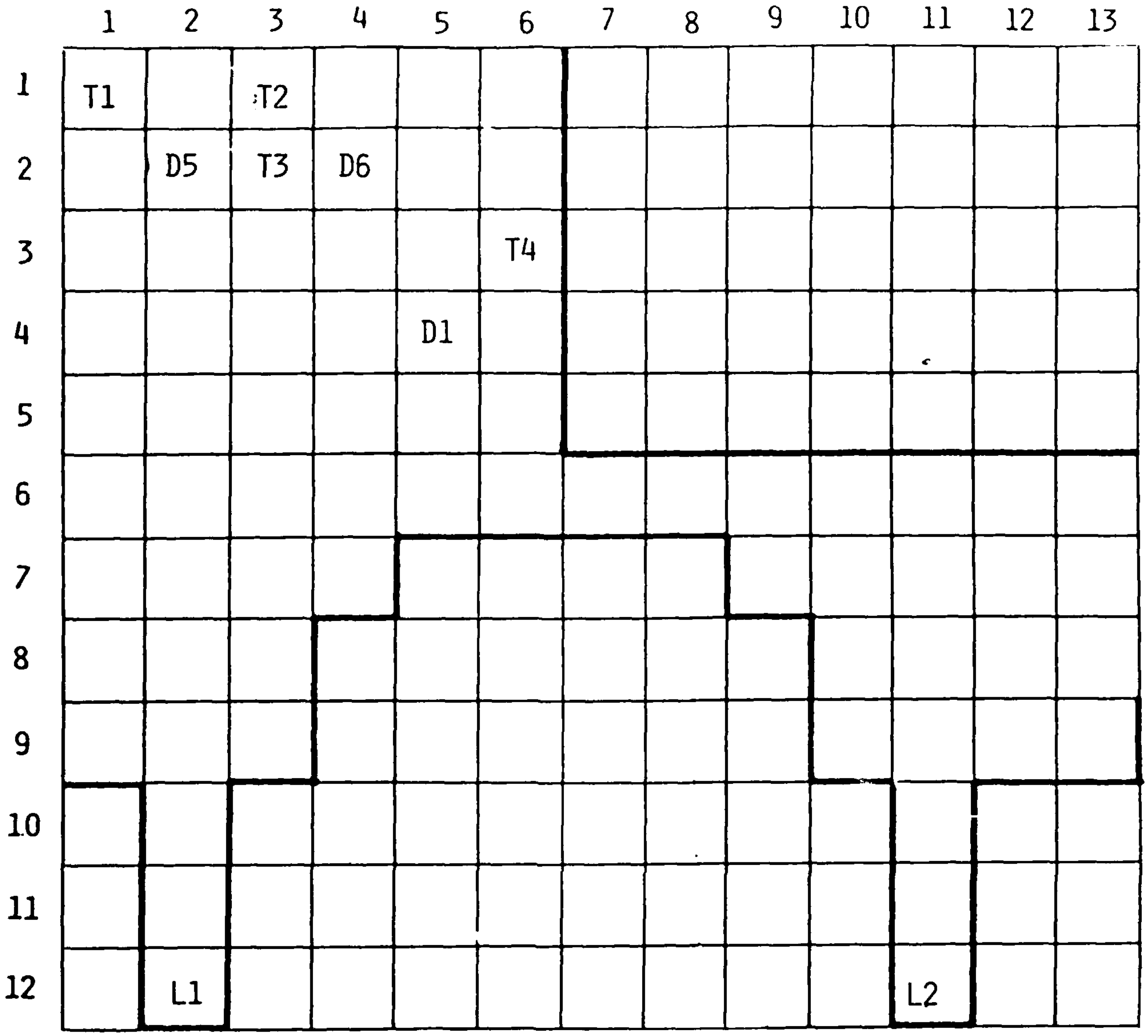
16 CPU MINUTES

PLAN

MISSILE #	FROM LAUNCHER #	TO DEFENSE #	LEAVES IT W/PE
5	2	5	.55
1	1	5	.14

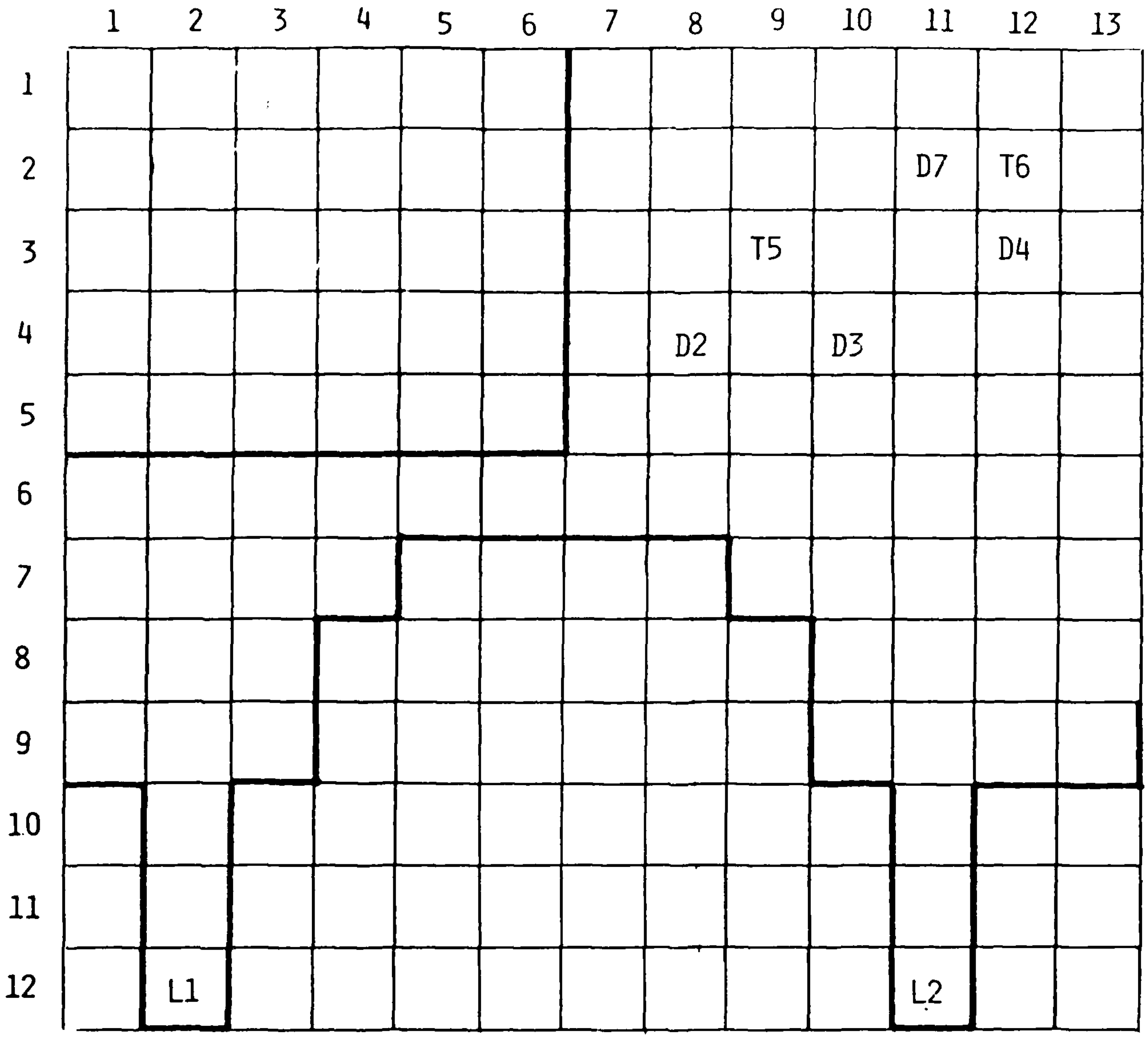
MISSILE #	TO LAUNCHER #	TO TARGET #	FOR A SCORE OF
2	1	2	173.50
3	1	1	146.73
6	2	6	98.30
7	2	4	81.92
8	2	6	66.09
4	1	2	73.16
		TOTAL =	639.70

Figure 5-1 (b) Example of Partitioning
Entire Board Results



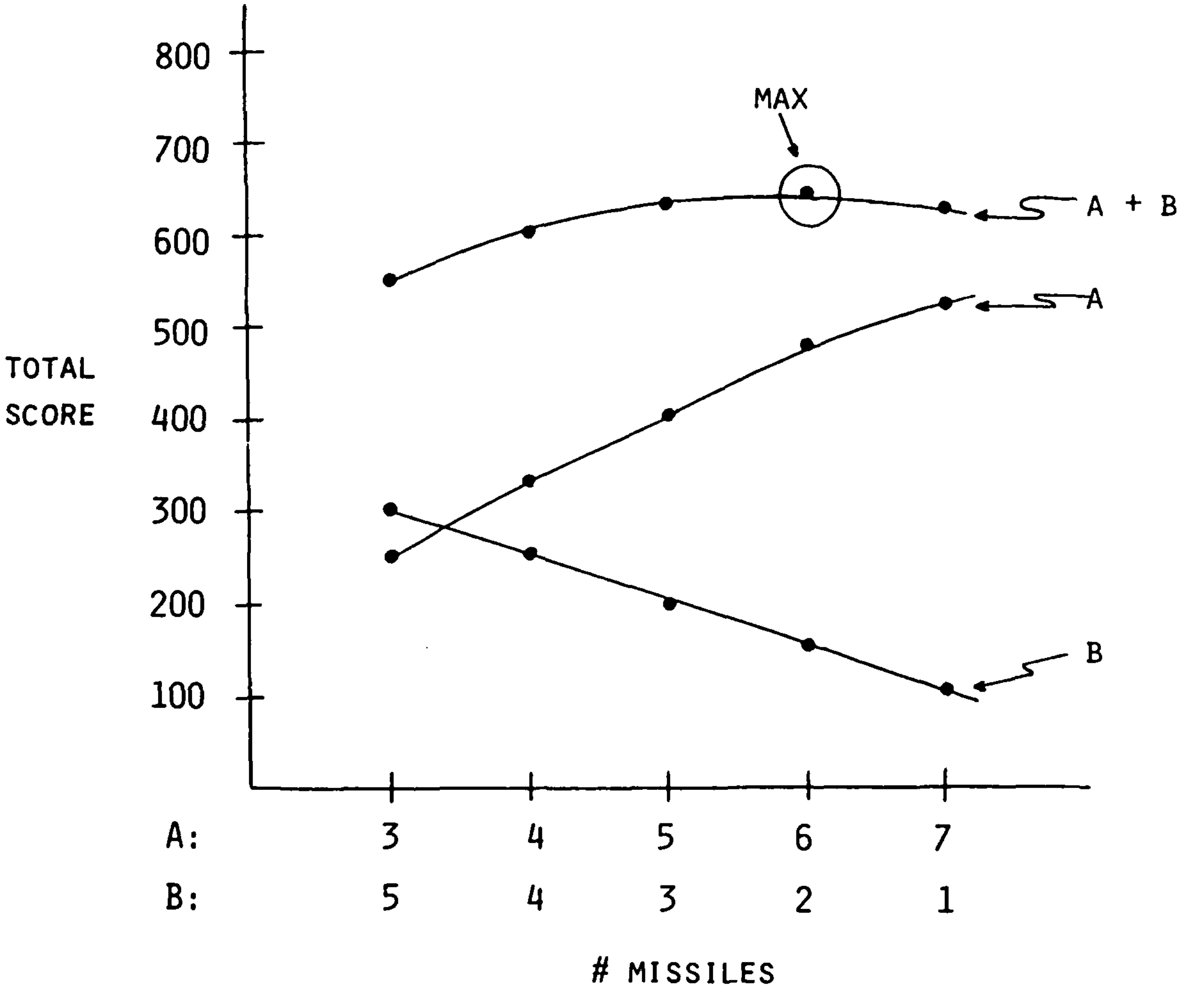
PARTITION A

Figure 5-2(a) Board Partitioning



PARTITION B

Figure 5-2 (b) Board Partitioning



AVERAGE OF 54 CPU SECONDS PER SAMPLE POINT
(TOTAL≈9 CPU MINUTES)

Figure 5-2(c) Board Partitioning

reassignment of assigned resources to a super-target to individual targets in the super-target.

We shall group targets together based on the following guidelines:

1. Expected value - we shall group targets together with similar expected value, which is given by

$$\text{Expected value} = V\, P_k$$

where V= value of target

P_k= probability that the target is destroyed by one resource

P_k includes the "ease" of destroying the target. Thus, two targets with different values may be grouped together if the one with a lower value is easier to destroy.

2. Reachability - if two targets can be reached from the same source, we may group them together.

We shall describe in the next section how such guidelines can be used in grouping targets for DSA. Furthermore, we shall discuss the influence of one's risk profile on the grouping of targets and its implications.

5.1.2.1 Grouping in Terms of Value

Let $T = \{t_1, \ldots t_n\}$ be a group of targets protected by a defense net $D = \{d_1, \ldots d_m\}$. The basic problem is how to allocate resources from different locations, $R_i = \{r_{i1}, \ldots r_{it_i}\}$, $i = 1, \ldots \ell$, to D and T so as to accomplish certain missions.

Let

$$\mu_i = V_i\, P_k^{(i)} = \text{expected value of target } i$$

Note that $P_k^{(i)}$ is computed as if t_i is not protected by any defense unit, and thus only relates to the physical "hardness" and the physical location (whether it is exposed to attack or not) of the target. We determine a subset based on expected value partition:

$$\mathcal{T}_V \overset{\Delta}{=} \{T_V(i),\ i = 1, \dots g_V\} \text{ where}$$

$$t_j \in T_V(i) \quad \text{if } \bar{\mu}^{(i)} \leq \mu_j \leq \bar{\mu}^{(i+1)},\ i = 1, \dots g_V$$

The determination of $\{\bar{\mu}^{(i)}\}$ is subjective; but once the decision maker specifies $\{\bar{\mu}^{(i)}\}$, the computer can easily create:

$$\mathcal{T}_V = \{T_V(i),\ i = 1, \dots g_V\}$$

5.1.2.2 Grouping in Terms of Reachability

Let $S \overset{\Delta}{=} \{s\ , \dots\ s_\ell\}$ be a sequence of <u>ordered critical points</u> (OCP) in the region. We shall use the notation:

$$Rj \xrightarrow[S]{} t_i$$

to represent the fact that a resource unit from region R_j can reach the target t_i and return to the region R_j through a path that passes through $s_1, \dots s_\ell$ in an orderly fashion.* The reachability criterion is based on fuel constraint (see Figure 5-3.) Let $\mathcal{S} = \{S^{(1)}, \dots S^{(q)}\}$ be a set of sequences of OCP. $\mathcal{S}$ is said to be a complete OCP set with respect to T if, for any $t_i \in T$, there exists some $j = 1, \dots m$ and some $S^{(k)} \in \mathcal{S}$ such that

$$R_j \xrightarrow[S^{(k)}]{} t_i.$$

Conceptually, the specification of a complete OCP set $\mathcal{S}$ represents ways one can "cover" all the targets by resources located in different regions. Given a set of targets T and a set of regions, there are an infinite number of different complete OCP sets one can construct. As we shall discuss later, the generation of a specific complete OCP set depends on many considerations, including *individuals' risk profiles.*

* We can relax the requirement that the resource must return to the originating region. The extension is rather straightforward, and therefore, for simplicity of illustration, we assume it is true in our discussions.

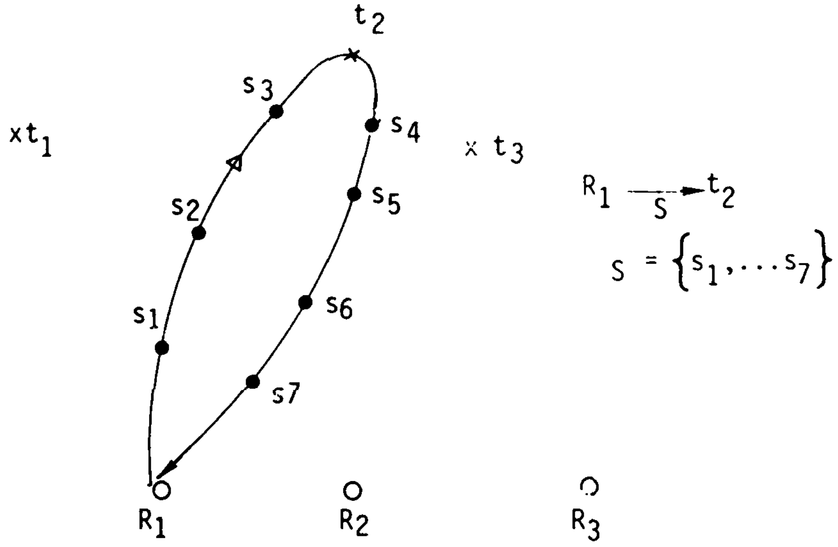


Figure 5-3 Illustration of Reachability

Given a OCP sequence set S, let us define

$$T_R\ (j,\ S) \overset{\Delta}{=} \{t_i \in T | R_j \underset{S}{--->} t_i\}$$

The set $T_R(j,\ S)$ represents the set of targets that can be reached by a resource unit in R_j via the same S (see Figure 5-4). Testing of whether t_i belongs to $T_R(j,\ S)$ is rather straightforward: Suppose the path is represented by

$$R_j ---> s_1 ---> s_2 ---> \ldots\ s_a ---> t_i ---> s_{\ a+1} \ldots ---> s_\ell ---> R_j.$$

Then assume a straight line path between the points and check to see whether the fuel constraint is satisfied through the piecewise straight line path as specified above. Therefore, given S, the construction of $T_R(j,\ S)$, j = 1, ... m, is immediate.

Given a complete OCP set $\mathcal{S}$, we can construct, for each $S^{(k)} \in \mathcal{S}$, the sets $T_R(j,\ S^{(k)})$, j = 1, ... m. We can now partition the target set T into groups represented by:

$$\tau_R(\mathcal{S}) = \tau_R(1,\ \mathcal{S})\ V\ \tau_R(2,\mathcal{S}) \ldots\ V\ \tau_R(m,\ \mathcal{S})$$

$$\tau_R(i, \mathcal{S}) = \underset{S^{(k)} \in \mathcal{S}}{V}\ T_R(i,\ S^{(k)})$$

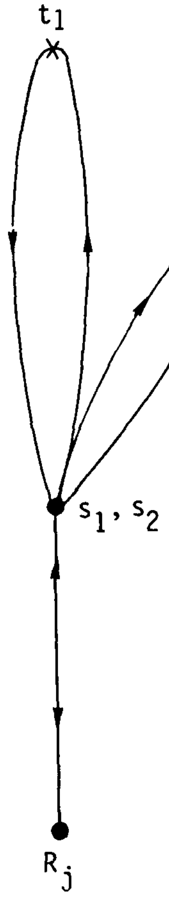
where the set operation AVB is illustrated in Figure 5-5. The set $\tau_R(\mathcal{S})$ represents grouping of targets via reachability from the source through paths that pass through the same sequence of OCP. The above definitions are also illustrated by Figure 5-6.

5.1.2.3 General Grouping in Terms of Characteristics

We can view two targets to be similar if they have roughly the same value and can be reached by resources from the same origin passing through the same sequence of OCP. Therefore, it is reasonable to define the grouping:

$$\tau(\mathcal{S}) = \tau_V\ V\ \tau_R(\mathcal{S})$$

Note that the grouping as suggested above is rather subjective. From Section 5.1.2.1, we see that τ_V is dependent on (1) the "value" of the target which is a subjective quantity (see also [1]) and (2) the notion of "similar value" (determination of $\bar{\mu}^{(i)}, i=1, \ldots g_V$). From Section



$S = \left\{ s_1, \ s_2 \right\}$ where $s_1 = s_2$

$$\left. \begin{array}{l} R_j \xrightarrow[S]{} t_i \\ \\ R_j \xrightarrow[S]{} t_2 \end{array} \right\} \quad t_1, \ t_2 \in T_R(j, \ S)$$

Figure 5-4 Illustration of the set $T_R(j, S)$

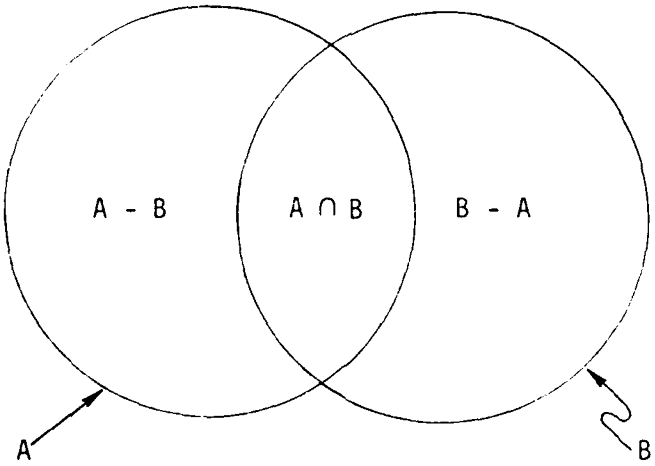
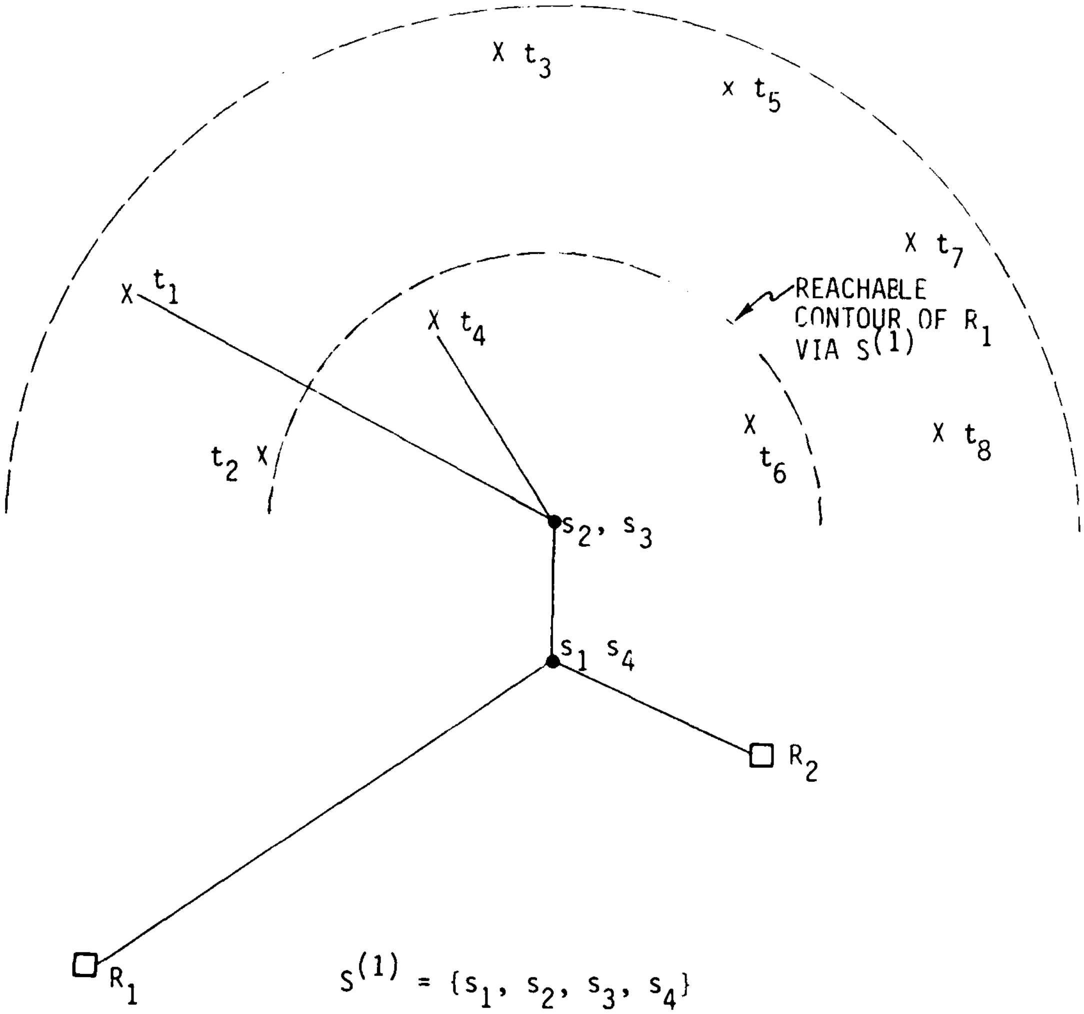


Figure 5-5 $AVB = \{A - B, A \cap B, B - A\}$



$S^{(1)} = \{s_1, s_2, s_3, s_4\}$

$\tau_R(1, S^{(1)}) = \{t_4, t_6\}$

$\tau_R(2, S^{(1)}) = T$

$\mathcal{S} = \{S^{(1)}\}$ is complete

$\tau_R(\ \mathcal{S}\ ) = \{\ (t_4, t_6), (t_1, t_2, t_3, t_5, t_7, t_8)\ \}$

Figure 5-6: Illustration of Complete OCP S and $\tau_R(S)$

5.1.2.2, we can see that $\tau_R(\mathcal{S})$ is dependent on the set $\mathcal{S}$ being chosen. As we shall see later, the selection of S is based on many factors: knowledge of path condition, defense strength, and the risk profile of the decision maker. Once $\{\bar{\mu}^{(1)},\ldots \bar{\mu}^{(g_v)}\}$ and $\mathcal{S}$ are specified, the grouping of targets into super-targets is rather straightforward.

In the next section, we shall expand our discussion on the selection of $\mathcal{S}$ based on different factors.

5.1.2.4 Penetration Path and Ordered Critical Points

So far, we have not incorporated the existence of defense networks in our grouping of targets. We shall now consider its effect in grouping and resource allocation strategy. Surrounding a defense site, we can map out a defense contour which represents the probability that a resource, transversing through the region, will be destroyed by the defense force located at the defense site (see Figure 5-7). Given a defense net protecting a set of targets, we have a defense contour surrounding the net (see Figure 5-8). From Figure 5-8, we see that there are three penetration paths, P_1, P_2, P_3, that pass through OCP $\{s^{(1)}, s^{(1)}\}$, $\{s^{(2)}, s^{(3)}, s^{(4)}, s^{(2)}\}$, $\{s^{(5)}, s^{(5)}\}$ respectively which are located at the "valley" of the defense contour. However, the path P_3 that passes through $\{s^{(5)}, s^{(5)}\}$ has a "lower penetration factor" in the sense that the probability that a resource can penetrate through the specified path unharmed is lower.

One can modify the defense contour by assigning resources to neutralize the defense, and thereby creating OCP which has a penetration path with a higher penetration factor. In the constraint of fixed resource, more resource allocated to defense sites will create more OCP with higher penetration factors, but then will reduce the number of resource units that can allocate to targets. Since the "scoring" is only related to the destruction of targets, the allocation of resources to defenses is more of a reflection on the security consciousness of the decision maker; whereas reservation of more resource units to targets reflects the decision maker's desire to "win big."

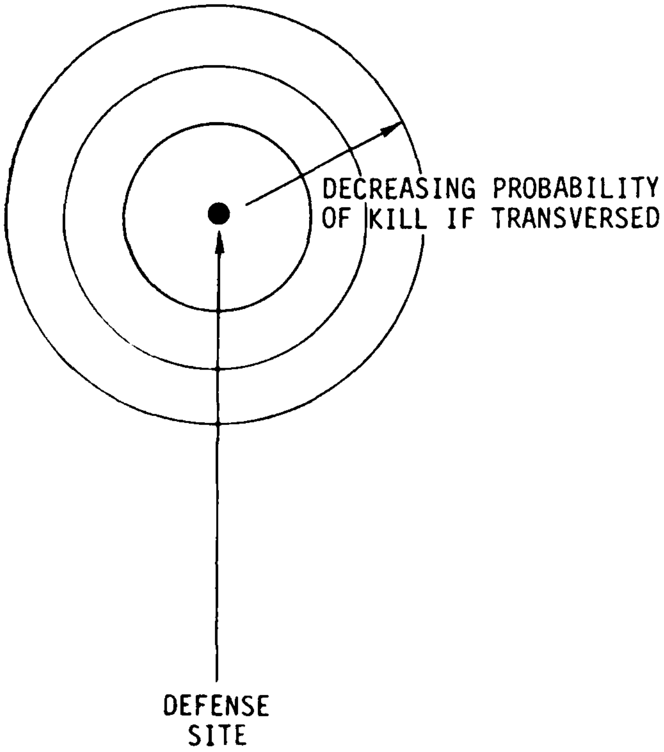


Figure 5-7: Defense Contour Surrounding a Defense Site

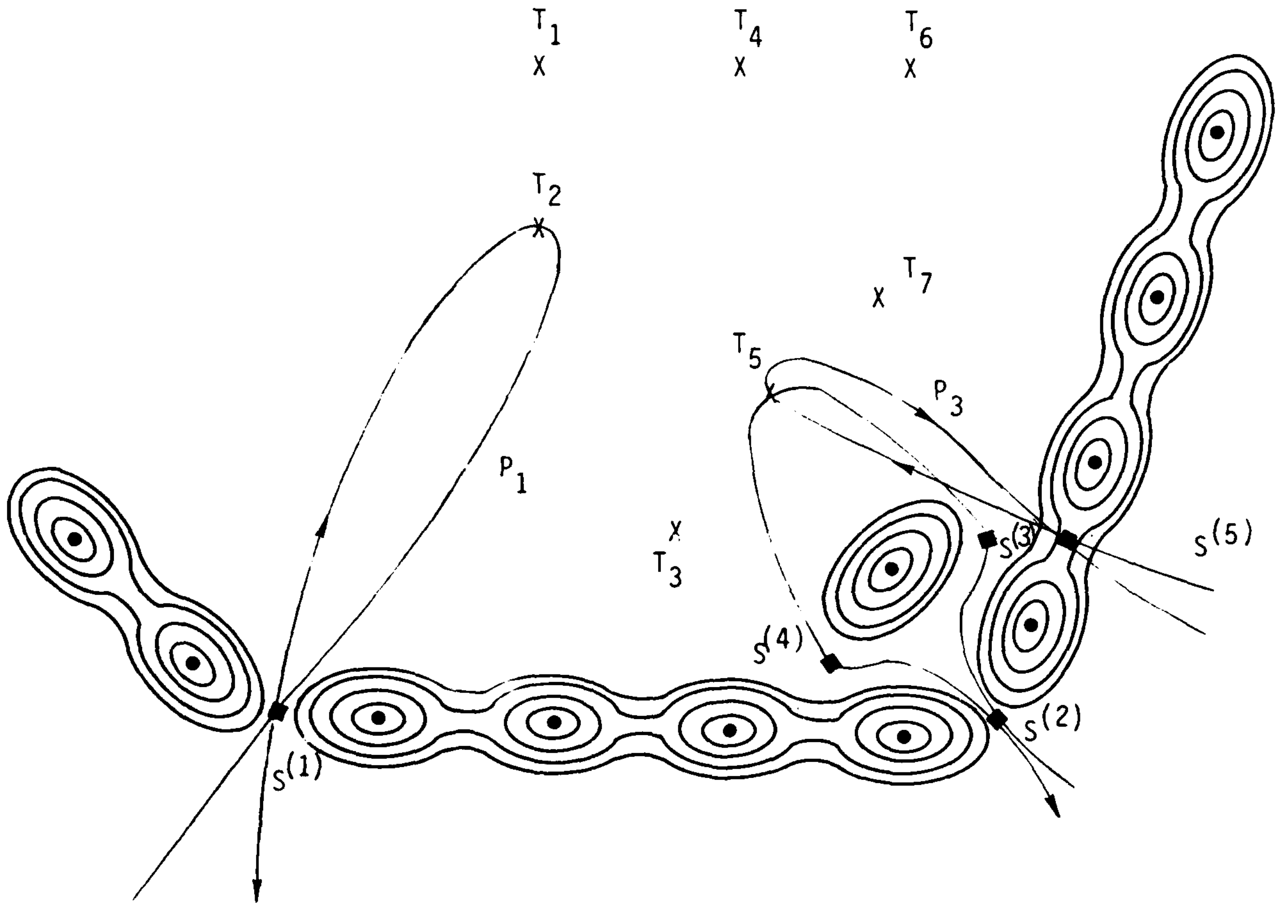


Figure 5-8: Penetration Paths through Defense Net

5.1.2.5 Risk Profile and Problem Formulations

We shall classify two types of decision makers: an aspiration driven type and a security conscious type. In this section, we shal. not be concerned with why a decision maker belongs to one type or the other; but rather, we are concerned with how such difference in risk profile will influence the problem formulation for resource allocation.

An aspiration driven decision maker is more concerned with "winning big" and thus first defines such a notion (e.g., total value of targets to be destroyed) and then tries to increase or maximize the chance of winning. A security conscious individual is more concerned with first creating penetration paths with reasonable penetration factors and then tries to "score" as much as possible by transversing through these paths. These can be translated into two different problem formulations.

(1) (Aspiration Driven) Given a set T to be covered to reflect one´s winning concept, how should one select $\mathcal{S}$, which is complete with respect to T, such that the corresponding penetration paths will have as high a penetration factor as possible?

(2) (Security Driven) Given a specific level of penetration factor to reflect one´s security notion, how should one select a set of penetration paths that will allow one to cover as large a subset of T as possible with the highest total value?

5.1.2.6 Optimization Problems

In this section, we shall describe the above formulations via two different optimization problems. For ease of discussion, we shall assume homogeneous resource and one platform region (R_1). The extension to non-homogeneous resource and multiple platforms is conceptually straightforward, but the indexing becomes very complicated.

Let T be a given a set of targets to be considered. Let $\mathcal{S} = \{S^{(1)}, \ldots S^{(q)}\}$ be complete with respect to T, and N($\mathcal{S}$) a specific allocation of resources to defenses in order to create penetration paths that pass through $\mathcal{S}$.

Associated with $S^{(i)}$ and $N_i(S^{(i)})$ is the penetration factor

$$\mu(S^{(i)}, N(S^{(I)})).$$

Let R be the total resource at R_1 and define

$$\mathcal{A}(\mathcal{S}, N(\mathcal{S})) \overset{\Delta}{=} \{ a_i (R_1, S^{(k_i)}); i=1, \ldots n \,\Big|\, S^{(k_i)} \in \mathcal{S}, R_1 \xrightarrow[S^{(k_i)}]{} t_i, \sum_{i=1}^{n} a_i \leq R - \sum_{i=1}^{a} N_i$$

as the set of admissible allocation vectors for targets where the resources are to travel along paths characterized by the set of complete OCP $\mathcal{S}$. Let

$$\mathcal{J} \overset{\Delta}{=} \text{set of all subsets of } \{1,\ldots n\}$$

and

$$\mathcal{J}_k = \{I \in \mathcal{J} \,|\, I = \{i_1, \ldots i_m\}, m < n \text{ s.t. } \sum_{j=1}^{m} V_{i_j} \geq k\}.$$

Thus, $\mathcal{J}_k$ represents the set of all subsets of $\mathcal{J}$ with a total value greater than or equal to K.

Since there is only one platform, we can assume that all units allocated to the same target will take the same path. Thus, we have

$$\text{Prob } \{T_i \text{ is destroyed } |\mathcal{S}, N(\mathcal{S}), a \in \mathcal{A}(\mathcal{S}, N(\mathcal{S}))\} = 1 - {}_{k_i}(1 - P_i)^{a_i}$$

where

$$\mu_{k_i} \overset{\Delta}{=} \mu(S^{(k_i)}, N(\mathcal{S}))$$

$$a_i = a_i(R_1, S^{(k_i)})$$

Let us denote

$$T_I \overset{\Delta}{=} \{T_i \,|\, i \in I\}.$$

We have

$$\text{Prob } \{T_I \text{ is destroyed } |\mathcal{S}, N(\mathcal{S}), a \in A(\mathcal{S}, N(\mathcal{S}))\} = \prod_{i=1}^{n} \Psi_i(I \,|\, a, \mathcal{S}, N(\mathcal{S}))$$

$$\overset{\Delta}{=} \Phi(I \,|\, a, \mathcal{S}, N(\mathcal{S}))$$

where

$$\Psi_i(I \,|\, a, \mathcal{S}, N(\mathcal{S})) = \begin{cases} 1 - \mu_{k_i}(1 - P_i)^{a_i} & i \in I \\ \mu_{k_i}(1 - p_i)^{a_i} & i \notin I \end{cases}$$

The event that the total value is greater than a certain value k is given by

$$\text{Prob } \{\sum_{i=1}^{n} \mu_i \geq k \mid a, \mathcal{S}, N(\mathcal{S})\} = \sum_{I \in g_k} \Phi(I \mid a, \mid \mathcal{S}, N(\mathcal{S}))$$

We have the following mathematical optimization problems:

(A) (Aspiration Driven) Specify K, then

$$\max_{\mathcal{S}, N(\mathcal{S})} \max_{a \in \mathcal{A}(\mathcal{S}, N(\mathcal{S}))} \text{Prob}\{ \sum_{i=1}^{n} \mu_i \geq k \mid a, \mathcal{S}, N(\mathcal{S}$$

(B) (Security Conscious) Specify 1 >α>0, and

$$\max_{\mathcal{S}, N(\mathcal{S})} \max_{a \in \mathcal{A}(\mathcal{S}, N(\mathcal{S}))} \{K \mid \text{Prob}\{ \sum_{i=1}^{n} \mu_i \geq k \mid a, \mathcal{S}, N(\mathcal{S}))\} \geq \alpha\}$$

The optimization problems are broken into two stages where stage 1 is involved with resource allocation after a complete set of OCP is specified and the number of remaining resources for targets is specified. The second stage is to utilize resources to create a complete set of OCP with a high penetration factor. The resource constraint induces a tradeoff between a high penetration factor with potential "big win." It is interesting to note that the first stage is analytical,* and can be accomplished via a mathematical optimization algorithm like maximum marginal return, dynamic programming, etc. The second stage is more heuristic than analytic. Judgement based on past experience, analogous situation, understanding of one's own resources and enemy's defense capabilities, and the risk profile of the decision maker play an important role in the determination of $\mathcal{S}$ and N($\mathcal{S}$). Without solving the above optimization problems, we can, in general, conclude the solutions characteristic of these two problems. These are illustrated in Figure 5-9. We see that an aspiration driven decision maker has a tendency to allocate more resources to targets and "take a chance", whereas a security conscious decision maker has a tendency to allocate more resources to defenses in order "to make sure."

* Once S is specified, o. .n construct super-targets as discussed in sections 5.1.2.1 - 5.1.2.. . and then apply a mathematical optimization algorithm.

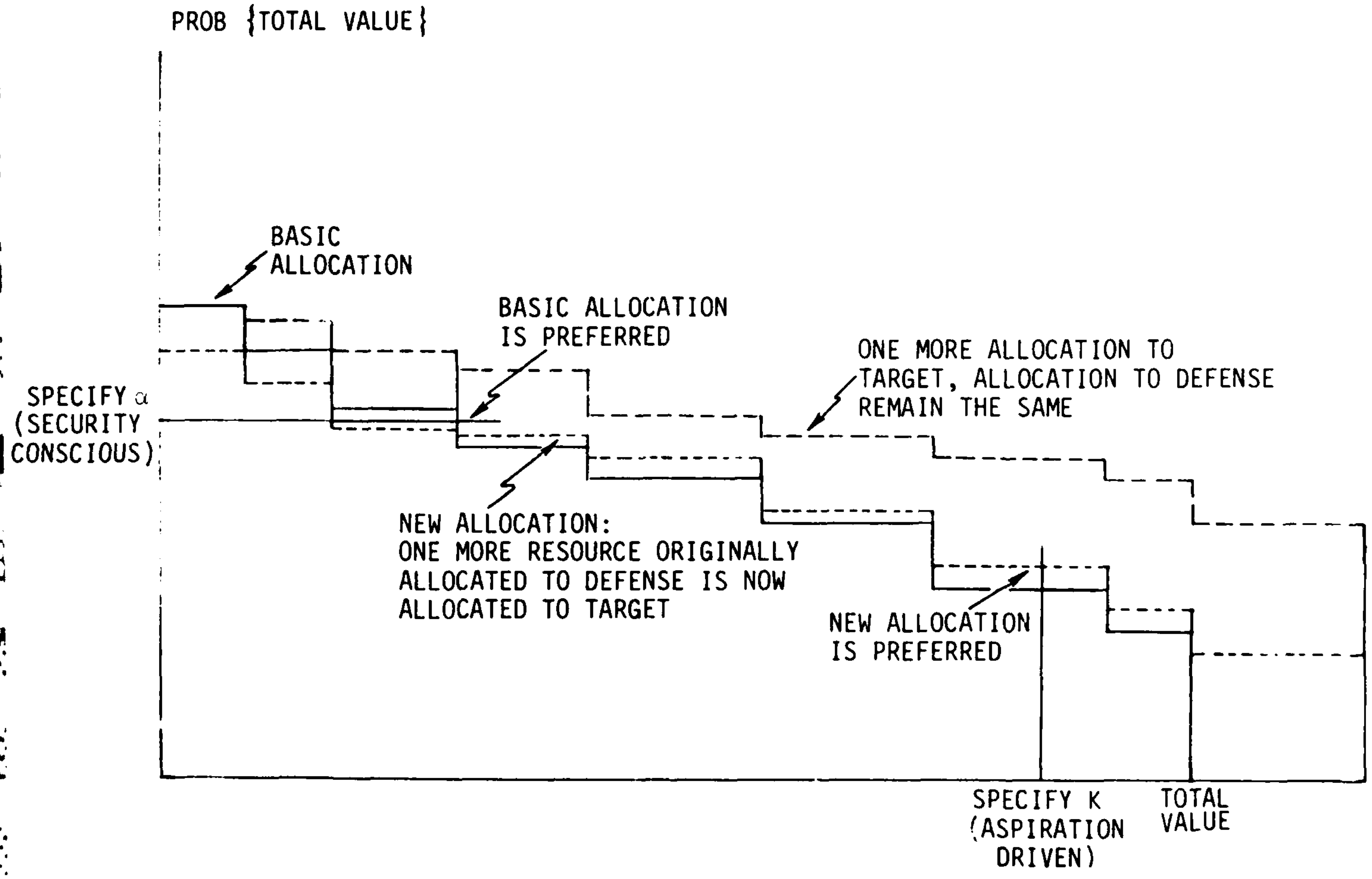


Figure 5-9: Comparison of Security Conscious and Aspiration Driven Decision Making

5.1.2.7 Interactive Dynamic Sequence Assignment

The dynamic sequence assignment process can also be modeled as an iteration of the following subprocesses.

(a) Determine T, $\mathcal{S}$ and N($\mathcal{S}$).

(b) Construct super-targets and then perform marginal analysis to allocate remaining resources to targets via penetration paths represented by $\mathcal{S}$.

(c) Refine paths to defense sites and targets; determine launch time for each resource, etc.

(d) Analyze resulting allocation to see whether there are undesirable features; if so, iterate through A to generate a better assignment.

We shall discuss each of these sub-processes separately.

Process (a):This is a process which is influenced mostly by the risk behavior of the decision maker. If the decision maker is aspiration driven, then he will start off by reserving a minimum number of resources for the targets which can achieve a certain total value (goal), and then consider whether the remaining resources should be allocated among defense and/or targets in order to maximize the probability of achieving the goal. A security conscious decision maker will, however, first determine the allocation of resources to the defenses in order to assure a certain level of penetration factor; and then allocate the remaining resources to targets. In either case, the process is a heuristic one.

Process (b):This is an analytical process. The allocation problem can be solved via mathematical optimization or maximum marginal analysis.

Process (c):This is a set of path optimization problems which can be solved by dynamic programming.

Process (d):This is a process of analytical studies via "what if" sensitivity analysis.

The overall flow of the process and the tasks associated with each subprocess are illustrated in Figure 5-10. During the iterative assignment process, the decision maker may switch from "security conscious" mode to "aspiration driven" mode depending on the analysis of the previously obtained potential assignment. When the decision maker is in a different mode, the flow of the assignment generation process is different (Figure 5-10). The stage where heuristic or expert system concepts is most applicable is in the determination of penetration paths to targets of interest. The breakdown as given in Figure 5-10 offers a guideline in the design of a decision system in enhancing the dynamic sequence assignment process.

5.2 TEMPORAL PARTITIONING

There may exist scenarios of interest in which defenses are able to be reconstituted before a second wave of attack begins. We examined ways of extending our SDFDP algorithm to handle scenarios with multiple attack waves. In this section, we will describe our research in this area.

5.2.1 Basic Formulation

We have concentrated our effort on the case of two attack waves, each beginning at a user-specified time. (Adding more attack waves would result in only minor modifications to the algorithm.) The user is asked to enter the times of the start of the attack waves, as well as the length of time each defense takes to reconstitute. Thus, if the length of reconstitution time of a defense is shorter than the time between waves, then that defense is considered to be reconstituted by

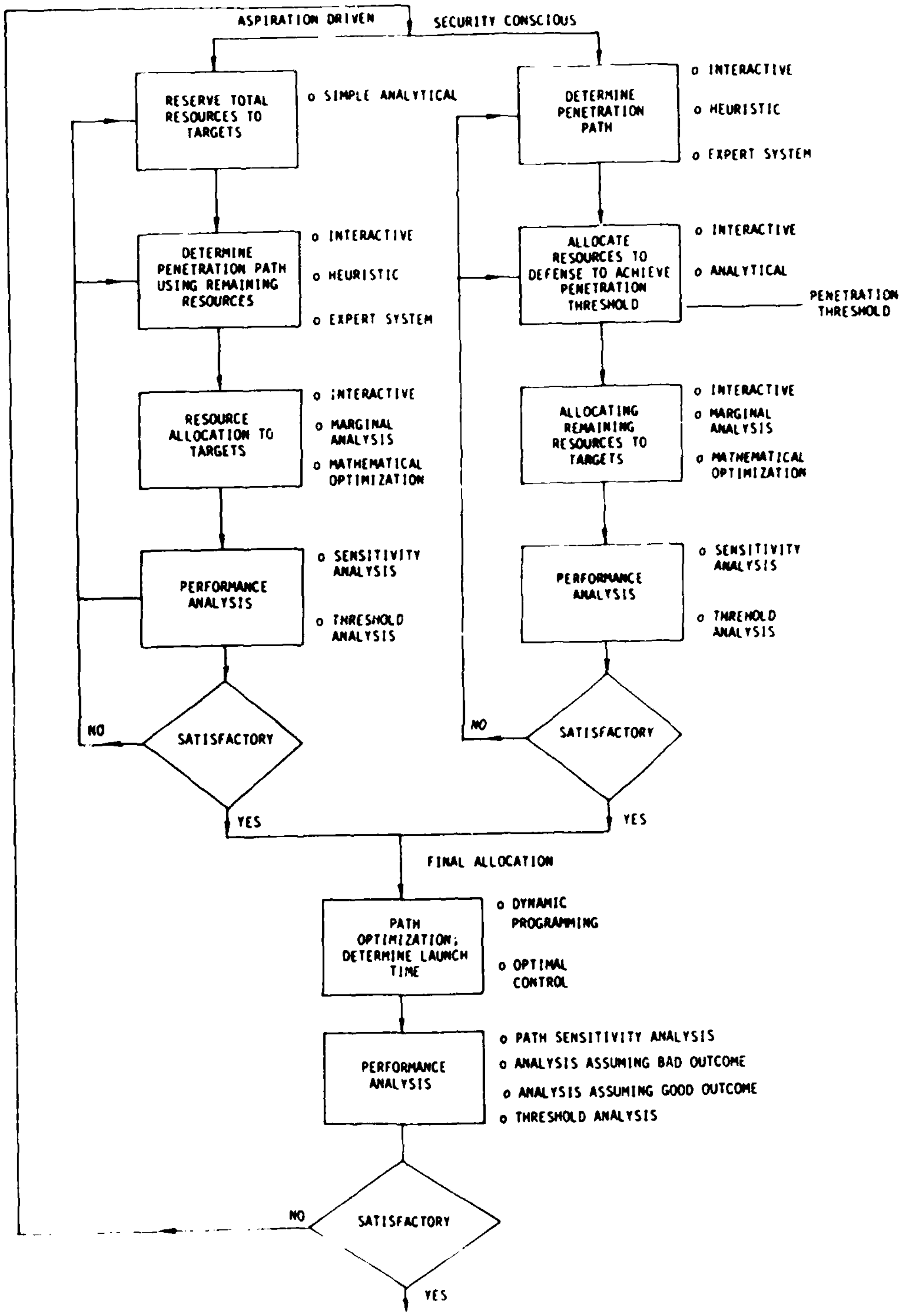


Figure 5-10 Dynamic Sequence Assignment Process

the start of the second wave. The user also must specify how many weapons will be available on each launcher during each wave. In our current research, we assume that weapons available during one wave will be used during that wave and would not be allowed to be "saved" for use in the next wave. Relaxing this constraint certainly would be an interesting and important topic for future research.

The algorithm begins in the same way as the SDFDP approach as described in Section 4.2. The dynamic programming, however, is done over all waves, not just once. The first time it is performed, both waves will yield the same result, as nothing has been attacked. (Thus, to save computing time, results of the first wave DP could simply be copied to get the second wave results.)

The first super-state is again to allow only targets to be attacked. Weapons available during the first wave are used first, followed by those available during the second. Thus, assignment algorithms are used just as in the conventional SDFDP algorithm. The reassignment algorithm, however, only tries to reassign launchers to targets during the same wave. That is, it checks all waves for a "bad" target assignment and decides if another launcher could attack that target during the SAME wave, and get an overall better score. The best plan is then recorded, along with the waves in which assignments were made.

Now, just as in Section 4.2, plans are generated which first attack defenses. The order in which they are generated is very similar to the order generated in conventional SDFDP, with the following exceptions. For every combination of defenses generated, every combination of waves must also be generated, creating even more plans. The heuristic originallv used which constrained defenses to be attacked in increasing order is now replaced by the restriction that a defense attacked during Wave 2 cannot be followed by a defense attacked during Wave 1. This restricts all defenses in a given plan to be in the order in which they are actually attacked. Also, the restriction that defenses can only be attacked twice is removed (or possibly replaced by the constraint that they only be attacked twice during any given wave). Thus, if there are three defenses, two attack waves, and the overall limit-to-defenses is two, then Figure 5-11 shows the sequence of plans that would be generated using temporal partitioning. (Di - Wj means Di attacked during Wave j.)

Plan	Combination of Defenses - Waves
1	no defenses
2	D1 - W1
3	D1 - W2
4	D1 - W1, D1 - W1
5	D1 - W1, D1 - W2
6	D1 - W2, D1 - W2
7	D1 - W1, D2 - W1
8	D1 - W1, D2 - W2
9	D1 - W2, D2 - W2
10	D2 - W1
11	D2 - W2
12	D2 - W1, D1 - W1
13	D2 - W1, D1 - W2
14	D2 - W2, D1 - W2
15	D2 - W1, D2 - W1
16	D2 - W1, D2 - W2
17	D2 - W2, D2 - W2

Figure 5-11 Sequence of Plans Generated by Temporal Partitioning

Once a sequence of defenses is generated, the defense assignment algorithm is used ⁻o determine the best launcher to each defense in the plan during the specified wave. Before the first attack of the second wave, defenses attacked during the first wave of the plan must be checked to determine if they can reconsitute before the start of the second. If so, then the probability of existence of that defense must be reset to 1, and the dynamic programming algorithm must be repeated. Now it may be possible that if the right information has been saved along the way, the dynamic programming may not be necessary and the information needed can just be retrieved. For example, if D1 has been hit during Wave 1 followed by D2 during Wave 1, and now a defense is about to be attacked during the second wave, then if D1 cannot reconstitute but D2 can, the state of the system by the time the second wave begins is as if only D1 were hit (because D2 reconstitutes). Thus, because of the sequence in which the plans are generated, we already know the results of the dynamic programming after hitting D1 only. Thus, the information can be retrieved. This phenomenon will not occur in general, however, and for many cases, it will be necessary to repeat the dynamic programming calculations. This point is a very important one because it means a very large increase in the amount of computing time in order to handle multiple attack waves.

After all the defense assignments have been made, the target assignment algorithm then determines the best launcher-target pair to assign during the first wave. If all the weapons available during the first wave were used to attack defenses, then the best pair during the second wave is found instead. The target reassignment algorithm and the defense-to-target reassignment algorithm are used to try to improve the overall score. The reassignment algorithms are basically the same as described in Section 4.2, except that reassignments must be made within the same wave. The target assignment, reassignment, and defense-to-target reassignment algorithms are implemented repeatedly until all the weapons of all waves are used. Note that weapons available during the first wave must be used before using the weapons available during the second wave.

The generation of plans continues until all plans are processed, and the best is finally outputted.

5.2.2 Implementation Considerations

As mentioned in the previous section, this approach to handling multiple attack waves has a serious drawback that must be considered: computing time. Although the theory of temporally partitioning the problem using the SDFDP algorithm is sound, the implementation of our research so far may cause some serious difficulty. The dynamic programming calculations of the SDFDP algorithm constitute the major bulk of computing time. Adding many more such calculations wil increase the computing time beyond the realm of feasibility. Certainly, more research is needed in this area to find methods of reducing the computational burden.

6. AN INTERACTIVE APPROACH TO DYNAMIC SEQUENCE ASSIGNMENT

6.1. INTRODUCTION

The problem of determining the dynamic sequential assignment of multiple resources against multiple targets, defended by a set of defense sites can be abstractly described as follows. Assume that there are n potential targets $(T_1,\ldots,T_n)$ to be attacked. These targets are located within a net of defense sites $(D_1,\ldots,D_m)$. Assume that all the resources are located at different regions $(R_1,\ldots,R_\ell)$. (See Figure 6-1).

The issues to be considered are

1. how many resources in region R_i should be allocated among $\{D_j\}_{j=1}^{m}$ and the targets $\{T_1,\ldots,T_n\}$?

2. if a certain resource in R_i is allocated to one of the $\{D_j\}$ or $\{T_k\}$, what is the path to be taken by the resource? what should be its initial time?

3. what is the sequential order of assignment?

4. what constitutes an "optimum" assignment plan?

In this section, we shall develop an interactive methodology that directly addresses the main issues of the problems. The approach relies on interaction among planners, who are more concerned with the success of mission; experts who are knowledgeable in the general geographic layout, attack strategy, possible enemy counter-measures; and a mission planning aid which integrates experts′ knowledge and performs mathematical analysis when needed in order to facilitate the planning process. In this section, we shall describe the overall approach, specify the necessary interaction and the different "task" functions that the planning aid needs to have. Some of the detail of the mathematical "tasks" to be performed by the aid will be discussed in the later sections.

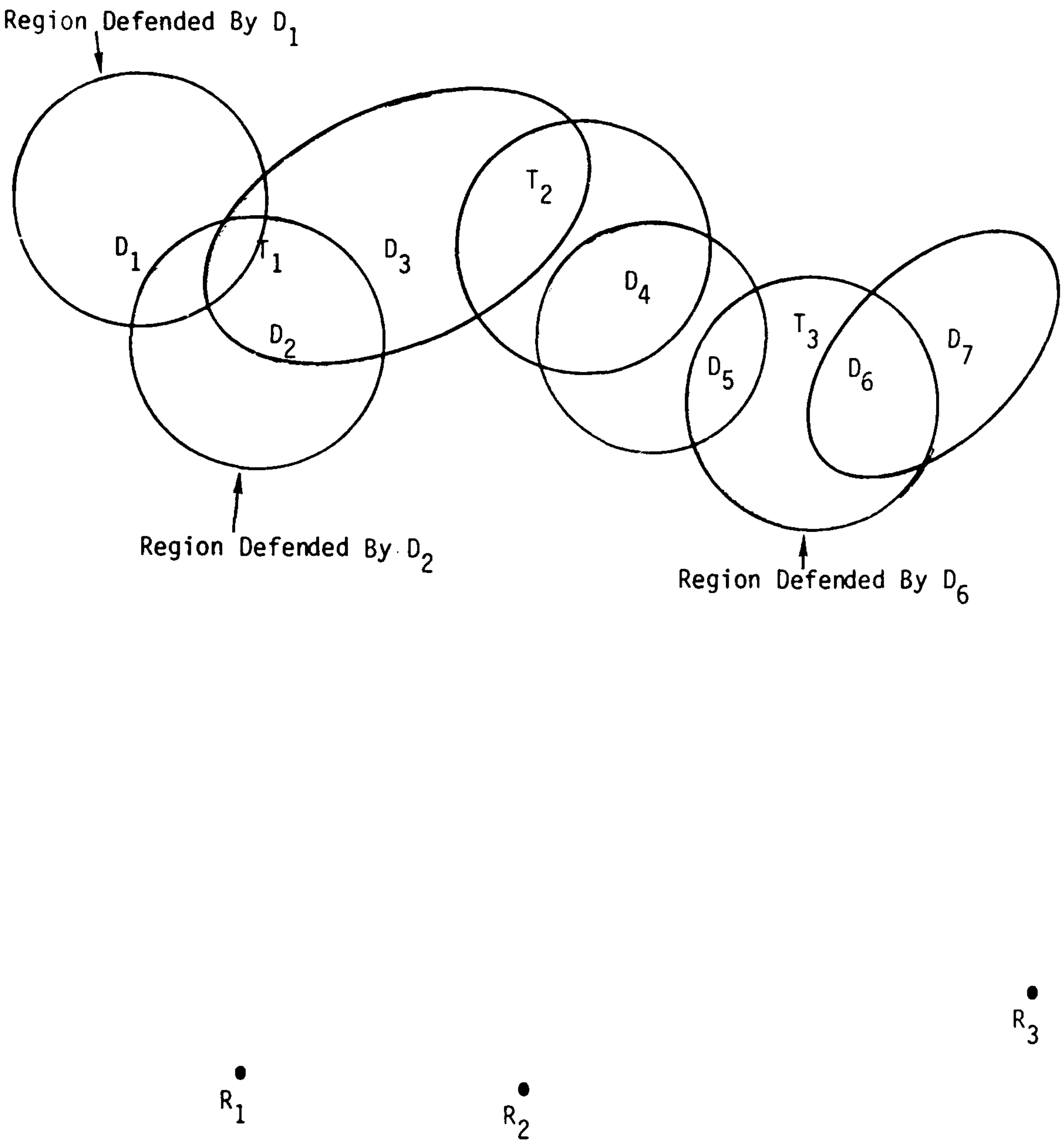


Figure 6-1 High-Level Geographical Representation of the Sequential Assignment Problem

6.1 AN INTERACTIVE APPROACH

The overall dynamic sequence assignment problem can be visualized as a two-level hierarchical optimization problem as described in Figure 6-2. The first level is concerned with the assignment problem. However, to do so, it needs to know the "benefit" associated with an individual assignment. This is to be performed in the second level where it computes the resulting optimal performance for individual assignment by solving the path optimization problem with initial time to be selected optimally, and computing kill probability, etc. The truly optimal assignment is obtained by searching through all of the possible assignments, and determining, for each assignment, the resulting performance.

The above problem description is still not precise enough to allow mathematical formulation: e.g., the performance measure associated with the assignment is not specified, and thus it is not clear what the second level should input to the first level when an assignment is made. This is probably the most difficult part in the problem solution: What should be the performance measure? How should one compare two assignments?

One approach is to construct a scalar measure which "reflects" the system performance; e.g., we can let

$$J= \Sigma_{i=1}^{n} a_i P_i \qquad (6-1)$$

a_i = relative figure of merit associated with target i

P_i = probability that T_i is destroyed

and use

$$E\{J\} \qquad (6-2)$$

as a scalar performance measure for the overall problem. Once a scalar performance measure like that of (6-2) is given, conceptually the problem can be solved via dynamic programming or similar search technique even though the computational requirements may be very large. The issue of the problem, however, is not so much the computational difficulty, but, rather, whether a scalar performance measure always faithfully represents the system goal. As an example, if the assignment problem is an abstraction of aircraft and/or missile assignments for destroying important stationary (or slowly moving)

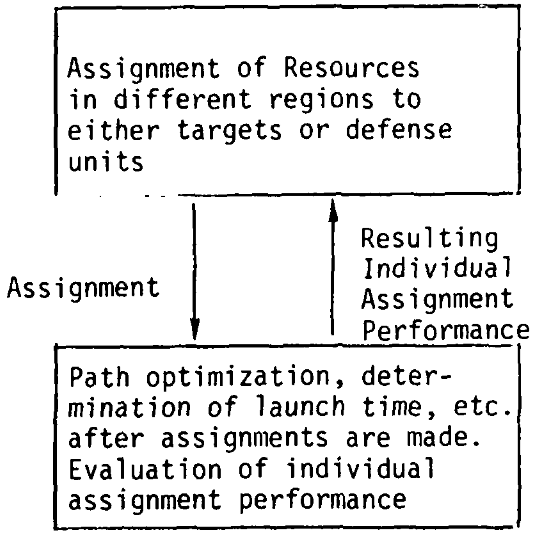

Assignment of Resources in different regions to either targets or defense units
Assignment
Resulting Individual Assignment Performance
Path optimization, determination of launch time, etc. after assignments are made. Evaluation of individual assignment performance

Figure 6-2 Two-Level Hierarchical Optimization Problem

targets, then the goal is to destroy the targets while reducing casualty. Such a goal is rather fuzzy and subject to many interpretations. The following are only a few possible representations.

a. Max $\sum_{i=1}^{n} a_i P_i$ subject to the constraint that $\sum_{j=1}^{\ell} \sum_{i=1}^{\ell j} b_i \tilde{P}_{ij} \geq \beta$; where $\tilde{P}_{ij}$ represents the probability that the i^{th} resource from region j survives after its attack mission is completed (ℓ_j = number of resource units in region j).

b. Max $\sum_{j=1}^{\ell} \sum_{i=1}^{\ell j} b_i \tilde{P}_{ij}$ subject to the constraint that $\sum_{i=1}^{n} a_i P_i \geq \alpha$.

c. Maximize P_i, i=1, ..., n as much as possible without regarding $\tilde{P}_{ij}$, when allocations have been made to assure that $\max_{i=1,\ldots,n} P_i \geq \alpha$, then try to make additional allocation to reduce $\tilde{P}_{ij}$.

d. Select a subset $\tilde{J} \subset \{T_1,\ldots,T_n\}$ of targets to be attacked; the selection is based on the requirement that after allocating all the resources to $\tilde{J}$, $P_i \geq \alpha$, $\forall\, T_i \in \tilde{J}$ and $\tilde{P}_{ij} \geq \beta, i=1,\ldots,\ell_j, j=1,\ldots,\ell$.

e. Similar to (4d) except the requirement that $\tilde{P}_{ij} \geq \beta$ is replaced by P_r {total number of resources survive after their attack mission $\geq K\} \geq \gamma$.

The list can go on and on. A particular choice of goal representation is subject to the decision maker's perception of the specific environment under which the mission is to be carried out. One interesting point that we noticed from the above list of representations is that all representations involve the variables $\{P_i, \tilde{P}_{ij}\}$. We shall call such variables the <u>attributes</u> that are associated with the goal to be accomplished.

In general, specification of goal gives rise to a set of attributes (but not the reverse direction), $\{A_i,\ldots,A_q\}$, which are directly related to assignments made; while individual's subjective interpretation of the goal, which may depend on one's position and interest, give rise to a specific goal representation.

Let J_i $(A_i,\ldots,A_q)$, i=1,...,f be a set of variables which represent the performance associated with the attributes $(A_i,\ldots,A_q)$. A goal representation can be quantified as a tuple $(S, \gg)$ where $S \subset R^{f'}$ is a subset in $K^{f'}$, $f' \leq f$ and $\gg$ is an ordering defined on S.

For example, the representation (a) can be quantified as follows

$$J_1 = \Sigma^{n}_{i=1} a_i P_i \quad , J_2 = \Sigma^{s}_{j=s} \Sigma^{\ell j}_{i=1} b_i \tilde{P}_{ij}$$

$$S = \{J_1 \text{ arbitrary}, J_2 \geq \beta\}$$

$$s \triangleq (J_1, J_2) \in S, \; s' \triangleq (J'_1, J'_2) \in S \quad , s \succ s' \text{ if } J_1 \geq J'_1$$

whereas representation (d) can be quantified as

$$S = \{(P_i;\ i=1, \ldots, n, \tilde{P}_{kj}, k=1, \ldots, \ell_j, j=1, \ldots, S) | P_i \geq \alpha,$$

$$i=1, \ldots, n, \tilde{P}_{kj} \geq \beta, k=1, \ldots, \ell_j, j=1, \ldots, \ell\}$$

for any two elements (s, s′) in S, $s \succ s'$ and $s' \succ s$.

Now, returning to the two-level hierarchical problem as described in Figure 6-2. The dynamic sequence assignment problem can be solved by the following steps.

1. <u>Evaluation</u>: Associate a possible assignment u with specific path and launch time, etc., and assuming certain nominal parameters (e.g., kill probability), evaluate the performance via

   $$J(u,\theta) = \{J_1(A_1(u,\theta), \ldots, A_q(u,\theta)), \ldots, J_f(A_1(u,\theta), \ldots, A_q(u,\theta))\}$$

   The overall evaluation is given by

   $$\mathcal{E}(u) \triangleq \{J(u, \theta);\ C(\theta),\ \theta \in \Omega\}$$

   where Ω represents the set of plausible parameters and $C(\theta)$ represents the confidence on θ.

2. <u>Exhaustion</u>: Exhaust all possible assignments (denote it by U) and evaluate $\mathcal{E}(u), \forall u \in U$.

3. <u>Preferred Assignment</u>: A partial ordering >> is induced on U via (s, α) and $(C(\theta), \theta \in \Omega)$ such that if

$U_1 >> U_2 \Rightarrow U_1$ is preferred over U_2.

The set U_p = {u $\in$ U there exists no $\hat{u} \in$ U with $\hat{u}$ >> u} is called the set of preferred assignment. Note that it is possible that neither $U_1 >> U_2$ nor $U_2 >> U_1$, in which case both assignments cannot be "compared". Given the set of preferred assignments Up, among many other things like individual risk profile, the final choice may depend on many factors which cannot be accounted for quantitatively.

While exhaustive search will give the set of preferable assignments, it is impractical and sometimes infeasible if the number of possible assignments is large. A mission planning aid is a system that facilitates the mission planning process. A planning process can be modeled as a sequence of iterative processes as illustrated in Figure 6-3. It is useful to discuss the main feature of each of the processes within the decision process, and indicates the needs for support in each process.

We started off with the process of identification of issues. This is a process where, by matching goal representation and conditional assignment evaluation (initially exogenous assessment is used instead), it identifies the main issues to be focused on in generating new assignments. This is a process where creativity is needed with support from very simple analysis which is centered around how different combinations of attributes will contribute to the goal satisfaction: why a certain target is more important that the others, why certain defenses should be avoided, etc. The outcome of such a process will help us to focus on generating a new assignment.

The process of assignment generation is to determine a new assignment based on either the issues to be focused on or modification of previous assignment. We shall elaborate on this in the next section. The main point to emphasize here is that the process of assignment generation is heavily dependent on domain expertise and analytical manipulation.

The process of determining optimal path and launch time, etc., is a rather straightforward analytical problem and can be handled by using an optimization technique and/or mathematical programming.

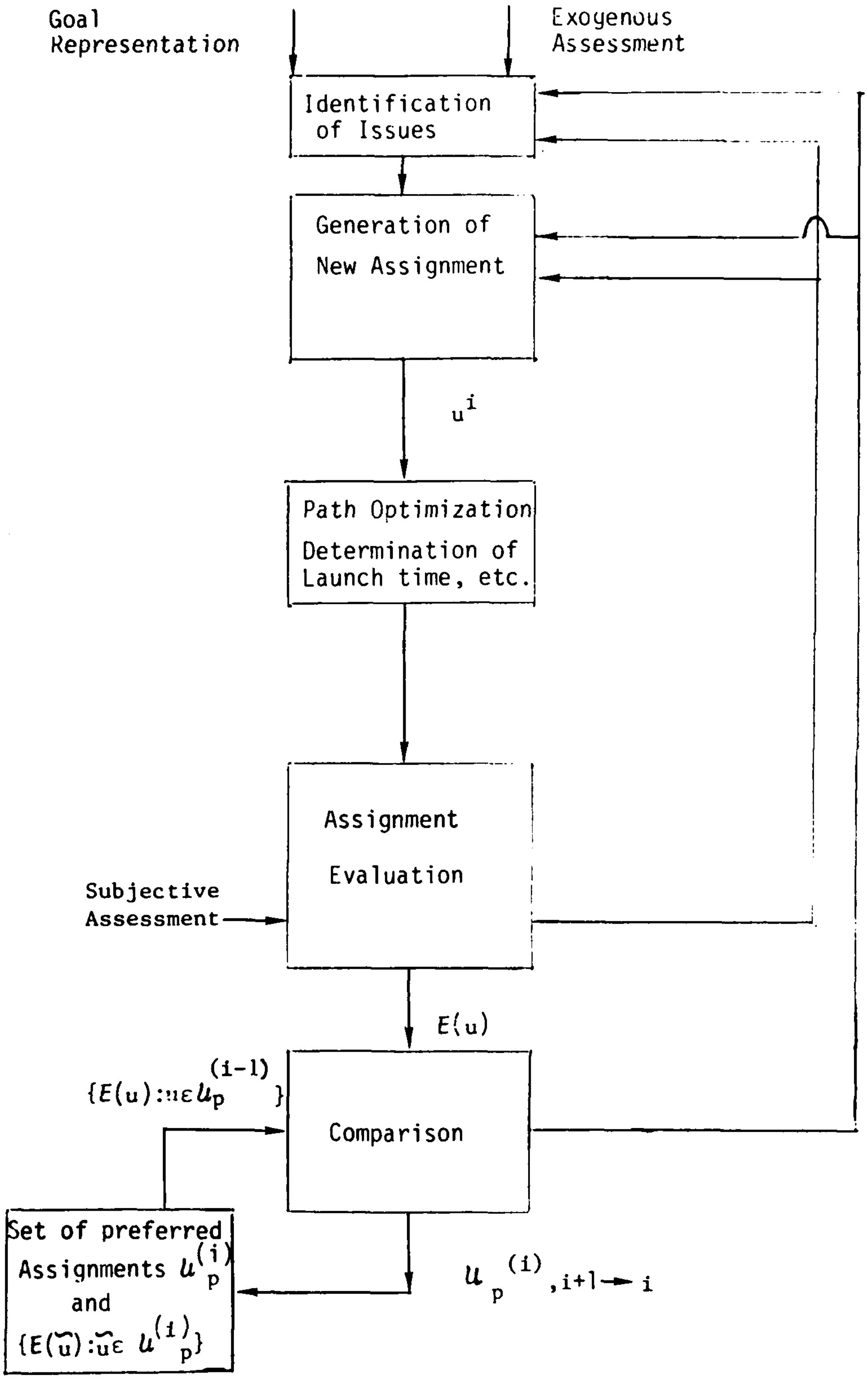

Goal Representation
Exogenous Assessment
Identification of Issues
Generation of New Assignment
u^i
Path Optimization
Determination of Launch time, etc.
Assignment Evaluation
Subjective Assessment
$E(u)$
$\{E(u): u \in \mathcal{U}_p^{(i-1)}\}$
Comparison
Set of preferred Assignments $\mathcal{U}_p^{(i)}$ and $\{E(\tilde{u}): \tilde{u} \in \mathcal{U}_p^{(i)}\}$
$\mathcal{U}_p^{(i)}, i+1 \rightarrow i$

Figure 6-3 A Planning Process

The process of assignment evaluation consists of two stages:

1. Evaluation of $J(U, \theta)$ for a nominal θ. This is a straightforward analytical problem involving mathematical manipulation.

2. Sensitivity evaluation of $\{J(u, \theta), \theta \in \Omega\}$, and the determination of $C(\theta)$, $\theta \in \Omega$. These involve both mathematical manipulation and a subjective view of what possible values $C(\theta)$ can take as well as how likely these values would be. This sub-process requires very tight interaction between experts' evaluation and value assessment.

We shall further elaborate on this in Section 6.4. The output of this process is either that the assignment is satisfactory or that it will provide input to re-generate new assignments either by introducing new issues to be incorporated in assignment re-generation or recommending possible modification. This "filtering" process requires subjective judgement and maybe some simple analysis.

The iterative process will generate a set of preferred assignments, denoted by U_p. The set U_p is built up as follows. Let $U_p^{(i)}$ be the set of preferred assignment at the end of the i^{th} iteration. By definition, $U_p^{(0)}$ = 0. Let u^{i+1} be the new assignment generated at the $i+1^{th}$ iteration. If u^{i+1} is "satisfactory", then

$$U_p^{(i+1)} \subset U_p^{(i)} \cup \{u^{i+1}\}$$

and if u^{i+1} is not satisfactory, $U_p^{(i+1)} = U_p^{(i)}$. The fact that $U_p^{(i+1)}$ is included in $U_p^{(i)} \cup \{u^{i+1}\}$ implies that some of the assignments in $U_p^{(i)}$ may be conceived to be not as good as u^{i+1} and therefore are not included in $U_p^{(i+1)}$.

Depending on time constraint, patience, and satisfaction level, the iterative process will stop after a finite number (say M) of iterations; and the set of preferred assignments would be $U_p \overset{\Delta}{=} U_p^{(M)}$. The set U_p may or may not be singleton, and if it is not singleton, then it either implies that the assignments in U_p are indifferent, or that other exogenous factor other than goal consideration will determine the final assignment to be selected.

Now we can describe the interactions among planners, experts and the planning aid in different stages of the planning process. This is summarized in Figure 6.4.

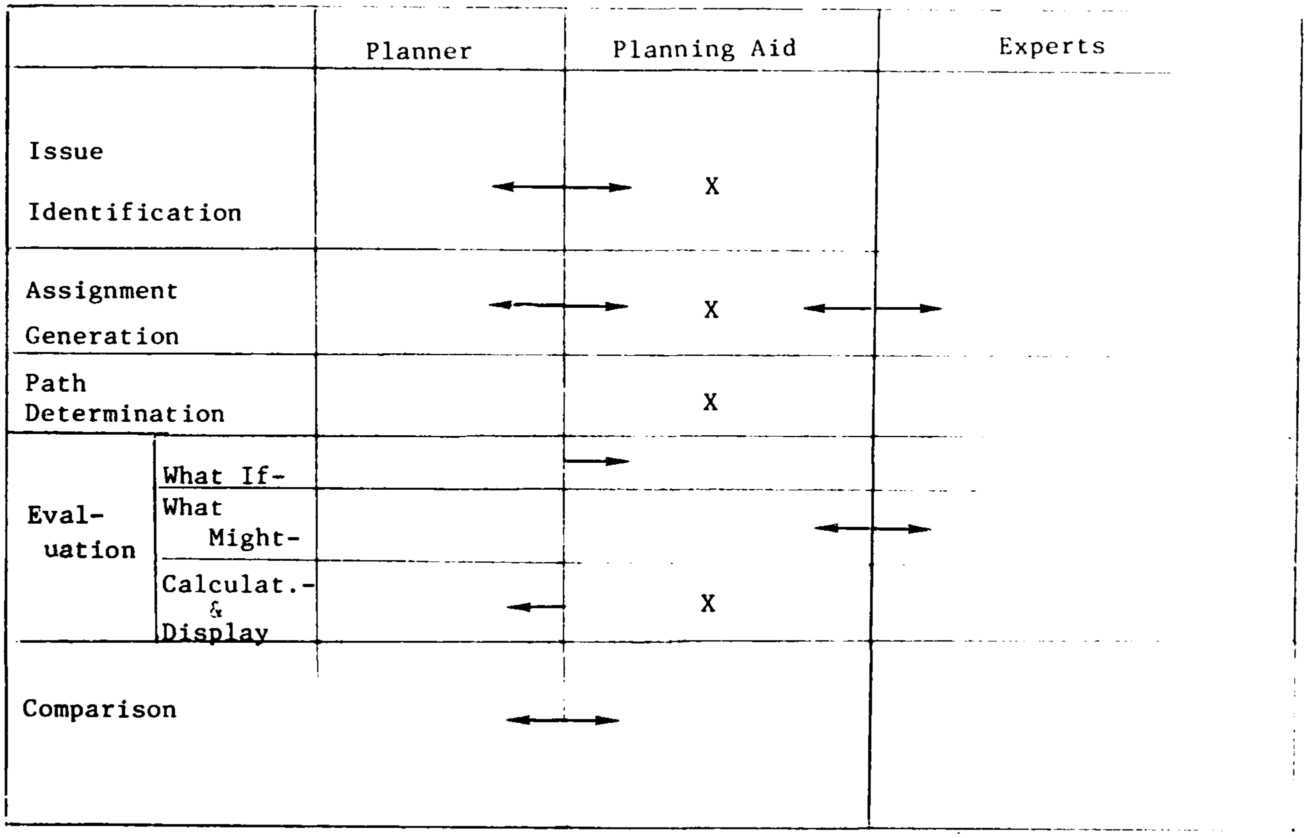


Figure 6-4 Interactions among Planners, Experts and Planning Aid

The arrow indicates input and the cross represents mathematical manipulation. Thus for example, issue identification requires inputs from planner to planning aid so that certain mathematical calculations can be carried out, and then the mathematical solution is displayed to the planner in some appropriate manner.

6.2.1. ASSIGNMENT GENERATION

The inputs to this process are ᴌhe targets to be destroyed and their relative values. The problem is that of assignment of resources to these targets and defenses so as to accomplish the mission which is reflected by the goal rɔpresentation.

We shall break the assignment generation into two problems: the first problem is to determine which targets are to be attacked and the nominal penetration paths; the second problem is allocating resources to the targets and their related defense units, assuming that the specified penetration paths are to be used. The first problem relies more heavily on heuristic and experts' knowledge while the second problem is a well specified resource allocation problem once a mathematical performance measure is specified. We shall discuss these two sub-problems separately.

6.2.2. Nominal Penetration Paths

The determination of nominal penetration paths is based on experts' knowledge of the tactical situation. Sometimes simple analysis can help in such determination. It is also stressed that interactions between planner and planning aid, planning aid and experts would facilitate this process. We shall discuss some of these interactions in this subsection.

A threat contour and terrain masking display by the planning aid to the planner would greatly help the planner to identify plausible avenues for penetration. A plausible penetration path is selected and appropriate experts are consulted to comment on the feasibility of the path (whether this can be done within the physical constraint of the resource capability and, if the resource is to be transported by human, the physical limitation of the human), the likelihood of encountering counter-measure of the enemy, etc. The decision aid can facilitate the interaction between planner and the appropriate experts; e.g., it can guide to locate the appropriate experts, extract relevant knowledge from experts and present the knowledge to the planner in an appropriate representation form. Some simple analysis based on experts' information can be employed to determine the desirability of the proposed plausible path. Several iterations between the planner and the experts may be necessary to arrive at a reasonably "good" penetration path. It is possible, also, to allow multiple penetration paths to one target.

For a more detailed discussion of determining penetration paths, see Section 5.1.2.4.

6.2.3. Resource Allocation

Given the penetration path to the targets, the next question is how resources should be allocated among the targets and defenses. In this subsection, we shall discuss a possible mathematical formulation to deal with such problems.

Let $\{T_1,\dots, T_t\}$ be the subset of targets to be attacked. Associated with each T_i is a set of penetration paths, denoted by L^T_{ik}, $k=1,\dots,t_i$. We shall say that T_i is defended by D_j if there exists L^T_{ik} which passes through the defense region of D_j, and we shall represent it by the symbol $L^T_{ik} \varepsilon \bar{D}_j$. For a given set of penetration paths $\{L^T_{ij}\}$, we can identify a subset $\bar{D}_1,\dots, \bar{D}_r$ such that $\bar{D}_j$ is nonempty, $j=1,\dots, r$. Let x_{ik} be the resource units allocated to take path L^T_{ik}. Define the following symbols

$X=\{x_{ik} \ ; \ k=1,\dots,t_i \ ; \ i=1,\dots, t\}$

$\{D_j = 1\}$: the event that D_j is operational

$\{D_j = 0\}$: the event that D_j is non-operational

$a = \{a_1,\dots, a_r\}$; a_i can be either 1 or 0

$D(a) = \{D_1 = a_1,\dots, D_r = a_r\}$ (thus if $a_1 = 1$, $a_i = 0$, $i=2,\dots, r$, then $D(a)$ represents the event that D_1 is operational and the rest of D_i are non-operational).

Using a defense network model, we can evaluate

$P(\bar{X}|D(a); X)$ = Probability that $\bar{x}_{ik}$ of x_{ik} leak through the defense network, $k=1, \dots, t_i$ $i=1,\dots, s$ conditional on the event $D(a)$.

By analyzing the hardness and the geographical outlay of T_i, one can evaluate

P^{ik}_k = probability that T_i will be destroyed by a resource unit delivered to it via L^T_{ik}.

Now to generate an assignment, we introduce a <u>mathematical performance measure</u> which is in-line with our goal. One possible choice is

J = Expected (relative) military value destroyed

$$= \sum_{i=1}^{s} w_i P_k^i$$

where w_i represents relative weights on the military value for each site representε the probability that T_i is destroyed. J is a function of X and $P(a)$:

$$J(X;P(a)) = \Sigma_{i=1}^{s} w_i \Sigma a \Sigma_{\overline{X} \leq X} [1- \pi_{k=1}^{t_i} (1-P_k^{ik})^{\overline{x}_{ik}}] P(\bar{X}|D(a);x)P(a)$$

$P(a)$ can be influenced by assignment resource y_j to D_j, via path L_j^D, $j=1, \ldots, r$. The selection of L_j^D can be carried out by path optimization or penetration discussed earlier. Let us define this by

$$P(a) = P(a|Y) \; ; \; Y = \{y_i, \ldots, y_r\}$$

and thus

$$J(X;Y) = \Sigma_{i=1}^{s} w_i \Sigma_a \Sigma_{\bar{X} \leq X} (1- \pi_{k=1}^{t_i} (1-P_k^{ik})^{\bar{x}_{ik}}) P(\bar{X}|D(a);X)P(a|Y)$$

$$= \Sigma_{i=1}^{s} f(X_i;Y) \quad , X_i \overset{\Delta}{=} \{x_{i1}, \ldots, x_{it_i}\}$$

where $f(X_i;Y) = w_i \Sigma_a \Sigma_{\bar{X}_i \leq X_1} (1- \pi_{k=1}^{t_i} (1-P_k^{ik})^{\bar{x}_{ik}}) P(X_i|D(a);X)P(a|Y)$

To generate a reasonably good assignment, we may want to solve the following optimization problem:

$$\text{Max} \sum_{i=1}^{s} f(X_i;Y)$$

such that $X_i \geq 0$, $Y \geq 0$; and sum of all resource units originated from a source must equal to the total resource units in that source.

The problem can be solved via dynamic programming; or, if the ıunction $f(X_i;Y)$ has certain :oncavity conditions, can be handled by extended marginal analysis as discussed in Appendix C. The decision aid should be capable of solving the optimization problem very quickly so that the user does not have to wait for a long time to have one assignment generated.

6.3. ASSIGNMENT EVALUATION

This process is input by an assignment generated via solving the above mathematical optimization problem. To evaluate such an assignment, we first evaluate the set of attributes associated with an assignment. Possible attributes are

(1) P_k^i, i=1,..., s (related success of mission)

(2) $P(\bar{X}|X) = \Sigma_a P(\tilde{X}|D(a);X)P(a)$ (related cost if resource itself has value)

The evaluations of these attributes are first carried out using nominal kill probability (e.g., P_k^{ik}), defense contour, and relative weightings $\{w_i\}$. Next, we carry out sensitivity analysis by varying P_k^{ik}, defense contour and investigate their impact on the deviation in the attributes. The following issues are to be considered in order to decide whether the assignment is desirable.

1. Would one of the attributes fall below a certain security level with variation in either P_k^{ik}, defense contour which is considered to be reasonably plausible (i.e., $\exists\, \theta \in \Omega$ such that $J(U,\theta) \in S$)?

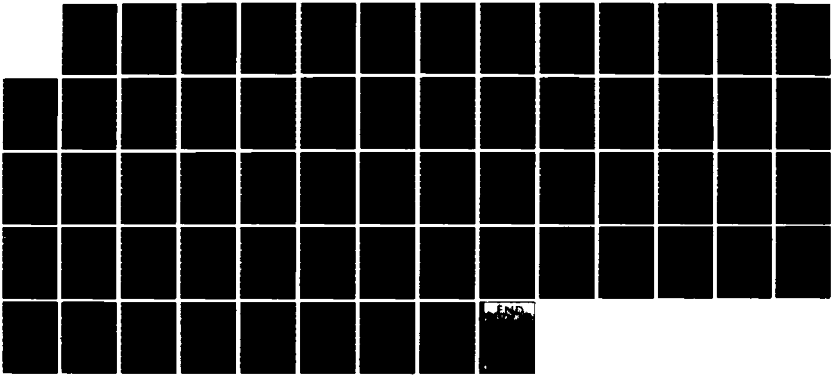
2. What would the scenario be to give a "bad" outcome? How likely would such a scenario be? Can one modify plan as such situation arises?

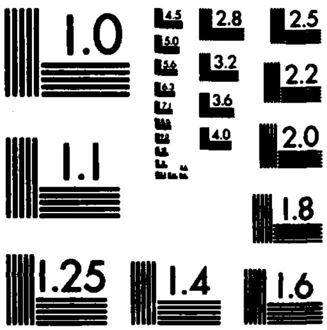
3. What would the scenario be for the "best" outcome? Can we promote the likelihood for such scenario to happen? How good is this "best" outcome? Does it achieve over the aspiration level?

Note that the evaluation is not based on one single numerical measure, but rather, a set of measures under different "what if" conditions. This also implies that evaluation must also be carried out in an interactive manner (see Figure 6-4). The above dialogue can identify the weakness of present

AD-A136 548 DYNAMIC SEQUENCE ASSIGNMENT(U) ADVANCED INFORMATION AND DECISION SYSTEMS MOUNTAIN VIEW CA C A O'REILLY ET AL. DEC 83 AI/DS-TR-1026-1 N00014-82-C-0085 2/2

UNCLASSIFIED F/G 1/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

assignment; and if the assignment is considered to be unsatisfactory*, such identification would guide in the next iteration in generating new assignment. If the assignment generated is satisfactory, it is compared with the set of preferred assignments $U_p^{(i)}$ that had already been generated.

6.4. COMPARISON

Let $u^{(i+1)}$ be the newly generated assignment and $U_p^{(i)}$ be the set of previous assignment. If $u^{(i+1)} >> u_j$ for some $u_j \in U_p^{(i)}$, then $U_p^{(i+1)} = U_p^{(i)} \cup \{u^{(i+1)}\} - \{u_j\}$; whereas if there exists some $u_j \in U_p^{(i)}$ such that $u_j >> u^{(i+1)}$, then $U_p^{(i+1)} = U_p^{(i)}$; and finally if neither $u^{(i+1)} >> u_j$ nor $u_j >> u^{(i+1)}$ for all $u_j \in U_p^{(i)}$, then $U_p^{(i+1)} = \{u^{(i+1)}\} \cup u_p^{(i)}$. Thus the construction of $U_p^{(i+1)}$ boils down to pairwise comparison between $u^{(i+1)}$ and u_j for each $u_j \in U_p^{(i)}$.

It is felt that the comparison issue should be resolved by the planner. However, since there are many dimensions to be considered in a multi-attribute situation, the planning aid can be used to help in setting up structure for priority assessment for the different attributes which would facilitate the comparison process. One possible tool is one based on the analytical hierarchical process discussed by Saaty [Saaty, 1980], yet another is one based on multi-attribute utility theory discussed by Keeny and Raiffa.

*Each assignment has a weakness, regardless of whether it is a satisfactory assignment or not.

7. EXTENSIONS OF BASIC SCENARIOS

In this section we discuss ways of extending the basic scenarios presented in this report to reflect many of the remaining issues involved in real life mission planning. We have already resolved some of these issues, while others merit further research.

7.1. TIME-COMPRESSION OF SEQUENTIAL SOLUTIONS

Our algorithms generate mission plans in the form of a sequence, which corresponds to the assumption that each new sortie has to wait for the completion of all the previous ones before even starting its flight. In real life situations one usually doesn't have this luxury and needs to send all the sorties out as soon as possible. Fortunately, there exists an efficient procedure to derive an equivalent time-compressed, concurrent mission plan from a given sequential one.

The reason that makes this possible lies in the fact that many sorties do not depend for their success on completion of their predecessors. For example, all attacks on distinct targets are completely independent of each other. Also, certain targets can be attacked without waiting for some defenses to be destroyed. Going back to Example 4 (see Figure 2-1, Figure 3-2, and Figure 7-1), we can see that the attack (A9 -> T6) can be executed without waiting for A3 and A4 to attack D3, since the trajectory to T6 does not come into contact with the area of coverage of D3.

If we consider the amount of completion of each sortie as an "event," then we can construct a "delay" graph G on these events. Let V represent the set of vertices (events) and E represent the set of edges of G. If V_1 and V_2 are two events, then (V_1,V_2) belongs to E if event V_2 has to occur after event V_1.

With each such ordered pair (V_1,V_2) we can associate the delay function $d(V_1,V_2)$, which is equal to the minimal necessary delay between events V_1 and V_2. This delay can be easily computed from the available optimal flight paths. Going back to Example 4, let V_1 = (A2 -> D2) and V_2 = (A9 -> T6). The delay function can be computed as follows: before the moment when A2 attacks D2, airplane A9 should be outside of the zone of coverage of D2. Thus, A9 cannot be any further along its path (see Figure 7-1) than square (4, 6). After A2 attacks D2, it will require an additional 12 units of time for A9 to travel the rest of its path to reach T6. Therefore, $d(V_{1,}V_2) = 12$.

Directed graph G satisfies the monotonicity (no loops) relation, as well as the weak triangle inequality $[d(V_{1,}V_2) < d(V_{1,}V_3) + d(V_{3,}V_2)]$. This allows for a simple reduction of this graph into a directed tree (arborescence), with the starting point as the root and attacks against targets as leaves. Figure 7-2 depicts such arborescence for the optimal sequential plan of Figure 3-2. Notice that the longest path within this tree is that between the start and (A12 -> T2) , which is 6+ 3 + 7 + 3 + 7 = 26 units of time long. If we subtract the maximal time/fuel allowance of 17 units from 26, we see that A12 can lift off any time after time 9 and complete its mission at time 26. The earliest times for lift-off of all the other airplanes can be computed in the same way. The numbers in brackets underneath each assignment in Figure 7-2 represent these times. The above procedure can be applied to any consistent sequential plan and will produce a time-compressed version of it. In particular, the plan in Figure 7-2 can be executed in 26 units of time, as opposed to approximately 180 units of time needed for its sequential prototype in Figure 3-2.

7.2. VARIATION IN ENEMY DEFENSE TYPES

Just like airplanes, enemy defenses can differ from each other. These differences can be manifested in terms of the sizes and shapes of their areas of coverage, their kill strengths within those areas, and their survival capabilities against aircraft attacks. As in the case of airplanes,

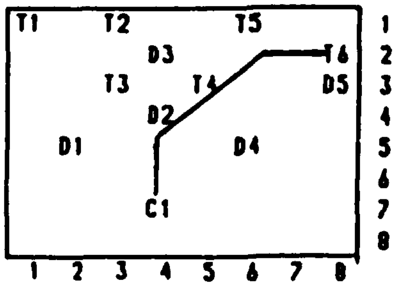


Figure 7-1 Optimal Path from C1 to T6

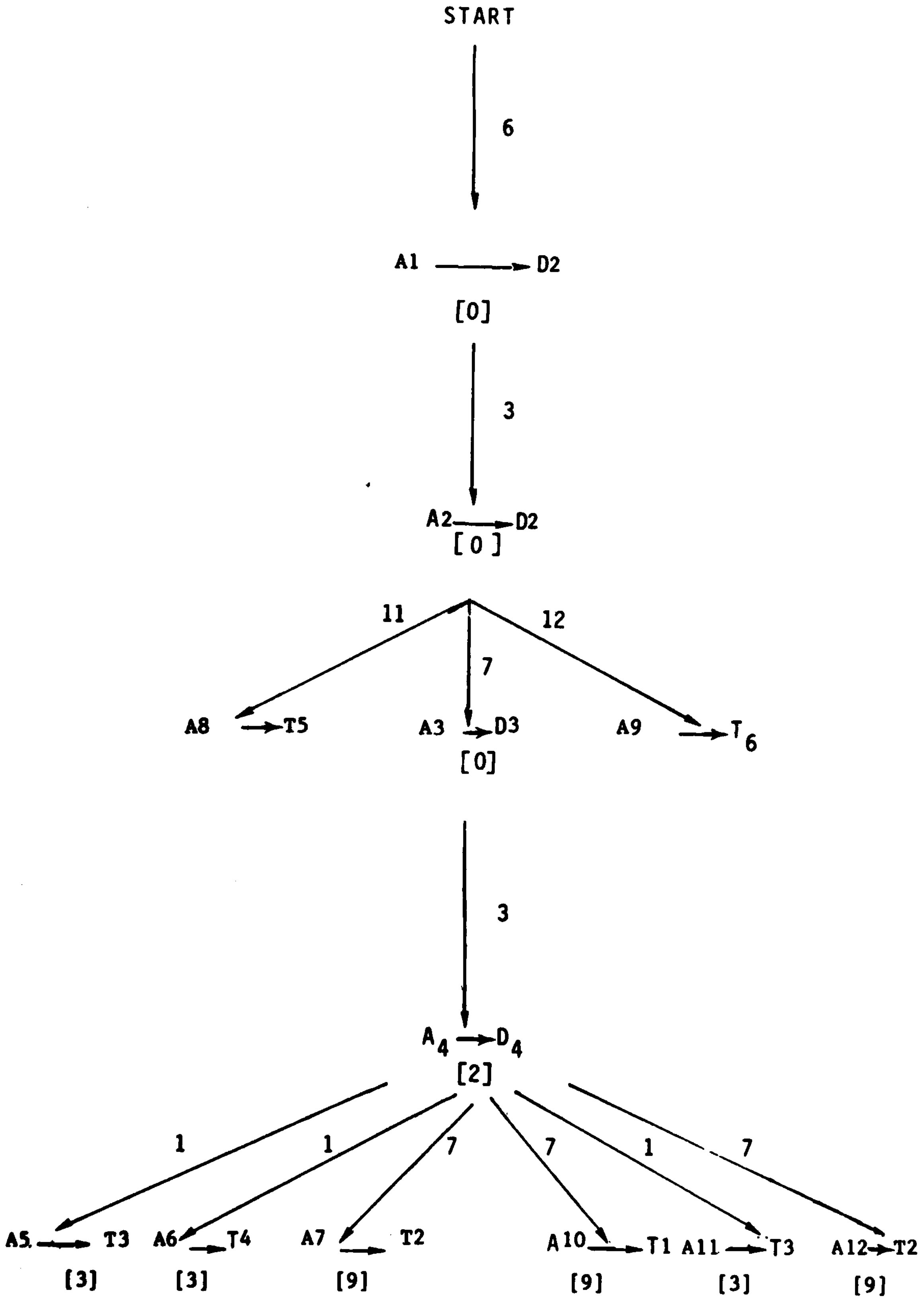


Figure 7-2 Optimal Compressed Plan

these differences are accommodated by our models and algorithms in a straightforward manner.

7.3. UNCERTAINTY IN ENEMY LOCATIONS

All the battle models that we discussed so far have presumed near perfect knowledge of the number of enemy forces and their positions and strengths. The case of reliable knowledge of the number of enemy defenses and their approximate locations and strengths, can be accommodated by blurring, i.e., spreading out their "probabilistic" area of coverage. On the other hand, the case of unknown enemy defenses (pop-up threats) at unknown locations presents a challenge for future research. Both of these ideas are important topics for further research.

7.4. DEFENSE SUPPRESSION

When several sorties have some part of their trajectories in common, it is often advantageous for them to travel together. In particular, when many airplanes fly over a defensive site, it is harder for that defense to handle all of them simultaneously. Thus, the airplanes will have, on the average, a better probability of survival by flying together than by flying separately.

This possibility presents a new attack option, and could be modeled by our planning system. One way to deal with this issue would be through a "doubling up" procedure, which can be viewed as a generalization of the Plan Compression procedure. It converts a sequential plan into its time-compressed counterpart with the maximum number of sorties executed together.

7.5. ATTACKS WITH MULTIPLE WEAPON LOADS

In our models we presumed that each aircraft was carrying a single weapon load and thus was carrying out an attack against a single objective. This is the case in most real life missions. However, multiple loading is an important option and can be addressed in future research.

One possible way to model this issue is to allow for one-against-many assignments to be included as single controls in our DP formulation. Introduction of multiple objectives will bring an extra wrinkle to the path optimization problem because there will be a new problem of allocating the available fuel between several legs of the flight mission. This, however, can be easily taken care of by an iterative procedure which computes optimal returns for the last leg of the trip, starting at the location of the second-to-last way point for a spectrum of feasible fuel allocations. These returns then become a part of the pay-off function for the second-to-last leg of the trip, which permits the computation of optimal returns for that leg for a spectrum of feasible fuel allocations. This defines a backward induction, which will solve the fuel allocation problem.

The solution proposed above has, however, a drawback in the sense that this type of modeling will increase the already large decision space, and it remains to be seen how much this would slow down the computation speed.

8. CONCLUSIONS

In this research contract we have made significant progress in several areas of the mission planning problem.

We have developed basic mathematical models of battle forces and control options. We have modeled a variety of issues dealing with multiple carriers, multiple aircraft types, aircraft fuel constraints, multiple enemy targets with variable importance, and multiple enemy defenses of variable strength. Electronic warfare, defense suppression, multiple weapon loads, weapon selection, recognizance and pop-up threats are issues that warrant further research.

Our major effort involved designing algorithms for finding optimal solutions to our battle scenarios. We have developed dynamic programming (DP) algorithms for low-level path optimization and for high-level mission planning scenarios. In the process we discovered a new version of DP, called DFDP, that provides a number of advantages over the standard DP.

Because of the extensive computational requirements of the Dynamic Sequence Assignment problem, we found it essential to modify optimal DP algorithms with some heuristic procedures designed for reducing the feasible state space. For that purpose, we have developed, implemented and compared a number of heuristic algorithms. These algorithms proved to be most effective when incorporated into the DFDP algorithm. We have also researched the practice and methodology of combining human guidance into our algorithms in order to achieve search space reduction.

In order to be able to study more realistic scenarios, we examined two ways of partitioning the problem. Spatial partitioning led to a decrease in computing time, so that larger scenarios could be studied. By partitioning the problem in time, we developed an approach that would enable multiple attack waves to be handled. Future research is necessary to improve this approach so that it can be actually implemented.

We have also looked at the methodology of dynamic sequence assignment, including such issues as hierarchical problem decomposition, multi-planner approach, selection of proper objective functions, and plan evaluation.

The interactive approach developed in Sections 6.2 and 6.3 focuses on solving the dynamic sequence assignment problem via a decision unit which consists of man and computer. As such, it avoids as much as possible the use of the computer to solve complex mathematical optimization problems; instsad it promotes the use of human judgment to break the complex problems into simple optimization subproblems that can be solved without much computational requirement. The decision aid enhances the decomposition process and provides analytical capability to solve the simple subproblems. Therefore, it is quite feasible to develop a decision aid system as discussed in Sections 6.2 and 6.3 for use on practical sized boards. The decomposition process allows one to break a very large assignment problem into a sequence of smaller sub-assignment problems and thus the requirement of the machine capability does not grow exponentially, but only linearly with the number of aircrafts, launchers, targets and air defenses.

Our research has made significant progress into the mathematical algorithms and techniques necessary to an aircraft mission planning aid. There are still many issues that warrant future effort and research.

REFERENCES

Bellman, R., (1957). Dynamic Programming. Princeton University Press, Princeton, NJ.

Callero, M., Jamison, L., Waterman, D.A., (1982). TATR: An Expert Aid for Tactical Air Targeting. Rand Corporation, Santa Monica, CA.

Case, K.E., and Thibault, H.C.,(1977). A Heuristic Allocation Algorithm with Extensions for Conventional Weapons for the Marine Integrated Fire and Air Support System. School of Industrial Engineering and Management, Oklahoma State University, Stillwater.

Engelman, C., Berg, C., Bischoff, M., (1979). KNOBS: An Experimental Knowledge Based Tactical Air Mission Planning System. Proc. of VI, International Conference on Artificial Intelligence., pp. 247-249.

Larson, R.E., and Casti, J.L., (1978). Principles of Dynamic Programming, Part 1. Marcel Dekker, Inc.

Marsh, J.P., and Grossberg, M., (1978). Research Report for the Advanced Weapons Management System (AWMS). Systems Control, Inc.

Nemhauser, G., and Garfinkel, R., (1972). Integer Programming. John Wiley, New York.

Nilsson, N.J., (1980). Principles of Artificial Intelligence. Tioga Publishing Company, Palo Alto, CA.

Rutenburg, V., (1982a). Depth-first Dynamic Programming Algorithm. Technical Memorandu TM-1026-2, AI&DS, Mountain View, CA.

Rutenburg, V., (1982b). Dynamic Sequence Assignment: Scenarios and Algorithmic Solutions. Technical Memorandum TM-1026-3, AI&DS, Mountain View, CA.

Saaty, T. (1980). The Analytic Hierarchy Process, McGraw Hill, New York.

Slagle, J., Cansone, R., Halpern, E., (1982). BATTLE - An Expert Decision Aid for Fire Support Command and Control. NRL Report 4847, Washington, D.C.

Winston, P.H. (1979). Artificial Intelligence. Addison-Wesley, Reading, MA.

Wishner, R.P., and Payne, J.R., (1979). Resource Allocation for Naval Platforms and Weapons. MIT/ ONR Workshop on distributed Information and Decision Systems.

APPENDIX A

Single Resource Unit Assignment

Appendix A
Single Resource Unit Assignment

This Appendix gives a method of approach and example results for a single assignment resource allocation problem. This approach was described in (Wishner, 1979) and in (Marsh, 1978) of the foregoing text.

The example utilized will be that of an aircraft on an interdiction mission but the methodology will be applicable to a variety of single force unit resource allocation problems. The performance measure utilized is "current military value" defined as

$$V(R) = V_T P_M + V_A P_S$$

where:

R = Vector of applied resources

V_T = Value of the target (objective)

P_M = Probability of accomplishing the objective (which is euqal to the probability the aircraft survives to the target times the probability of killing the target given that it survives to the target).

V_A = Value of the aircraft for future missions

P_S = Probability that an aircraft survives this mission.

For two targets the above formula becomes

$$V = V_{T_1} P_{M_1} + V_{T_2} P_{M_2} + V_A P_S$$

The above formulas can be normalized by dividing them by the future value of the aircraft for future missions. The resulting performance measure is used in the examples below. Note that P_M and P_S and therefore V depend on the trajectory flown and the resources utilized. Thus, we can define our optimization problem as

$$\underset{R}{\text{maximize}}\ V(R)$$

where R is the vector of resources available including

flight path,

RF jamming power used against given threat,

the number of decoys used at a given time, and

the number of ARMS used against a given threat

subject to the constraint that: (1) the fuel used be less than or equal to the available fuel, and (2) the allocated instantaneous jamming power, number of decoys, number of chaff packages, and number of ARMS be less than or equal to that available.

We first explain the solution approach with only flight path optimization. Consider Figure A.1. Each cell in Figure A.1 contains the probability of survival of the aircraft from the enemy threats that can attack that cell. The ground is represented by probability of survival equal to zero.

To determine the optimal trajectory, we must effectively examine all trajectories to a target and back to a landing point that satisfy the constraints, calculate the probability of survival ($P_{S_{xyz}}$) at each grid point (x,y,z) on the trajectory, and multiply all the $P_{S_{xyz}}$ together to obtain an overall P_S. To accomplish this, the volume in which the aircraft can fly is divided into cubes (see Figure A.1). The cube dimension depends on the aircraft maneuverability and the degree of variation in the aircraft's probability of survival, P_S, across the cell. A backward dynamic programming algorithm is then used to find the optimal flight path.

Dynamic programming provides an approach for solving optimization problems involving multi-stage decision processes. These problems are characterized by the fact that the control decision taken at the present time affects the behavior of the system at future times, and hence the solution is a sequence of decisions over the entire duration of control, not just a decision at the present time. Basically, dynamic programming converts the simultaneous determination of the entire optimal control sequence into a tractable sequential solution of vastly simpler intermediate optimization problems. The resulting solution is precisely the one obtained by exhaustively searching all possible control combinations without actually performing such a computationally prohibitive search.

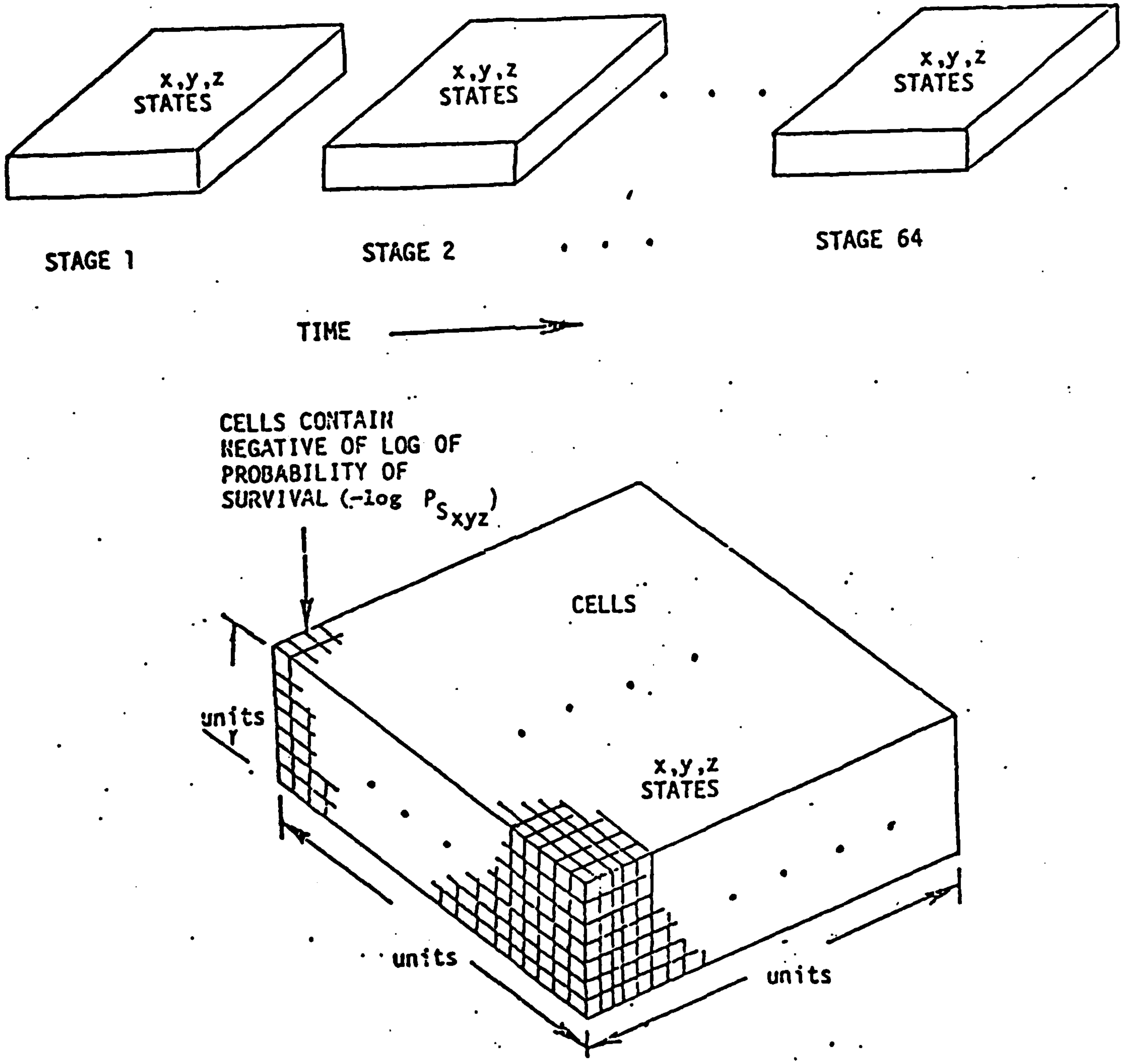


Figure A.1 Three-Dimensional Dynamic Programming Space Description

With the dynamic programming formulation, the motion of the aircraft is described by a Markovian state equation, where the state contains the position of the aircraft as a function of an independent variable known as the stage variable. The stage variable, represents motion of the aircraft through the quantized state space. Since the length of time the aircraft stays in each quantized cell is dependent on its velocity and path through the cell (e.g., along a diagonal or straight through), the stage variable is only loosely related to time.

Each cell in Figure A.1 has an associated probability of aircraft survival obtained from threat models. The negative logarithm of the probability is used in practice so that the overall path survival probability can be obtained by summation of positive numbers (logs of probabilities are negative) rather than multiplication. Maximization of the probability of survival, therefore, results from minimizing the sum of the negative logs. The target is located at some specified cell in the state space. At the last stage, the state space is loaded with values pertaining to the probability of accomplishing the mission dependent upon arriving at each location. For the case of an overflight requirement, this boundary condition amounts to placing a zero probability of survival at every location except the target's location, where the threat model dependent value of probability of survival is used. This forces all solutions to end at the target.

The problem is to find the path and associated velocities through the space which maximize the performance measure (i.e., net military value) constrained by available fuel. The aircraft is constrained to always move at least one cell (quantized position) from stage to stage. Permissible movements (transitions) are to any of the cells adjacent to the one containing the aircraft. Thus, for a space partitioned into cubes, 26 different transitions are possible from each cell because the space containing the cell under consideration is 3 x 3 x 3 = 27 total cells in size and the solution is constrained to move to a new cell.

The dynamic programming solution starts out at the next-to-the last stage (the performance measure at the last stage is the negative log of the probability of accomplishing the mission conditioned upon arriving at each position). For each state in the $(n-1)^{th}$ stage (with n total stages), the transition to an adjacent state in the n^{th} stage that minimizes the performance measure is determined and stored, along with the performance measure itself. The optimal transition includes a direction and a velocity. Now that a performance measure has been determined for each state in the $(n-1)^{th}$ stage, similar calculations are performed for each state in the $(n-2)^{th}$ stage. This process continues to move back through successive stages until the calculations have been performed for the first stage, which contains only the initial position of the aircraft. The overall optimal trajectory can now be found by tracing through the optimal state transitions from stage to stage. The solution vector of positions and velocities is then mapped into a time-ordered control vector which is the desired solution form.

In order to include a fuel constraint, the performance measure is augmented by a LaGrange multiplier which multiplies time and the fuel rate. Then a search over the value of the LaGrange multiplier occurs. In practice, only a few iterations are required for this search.

The optimization over trajectories and EW resources such as chaff, jamming, decoys and anti-radiation missiles can be performed in a sequential fashion. With a given onboard resource assignment to air defense units a dynamic programming algorithm was utilized to find the optimal trajectory and thence for the given trajectory the optimal EW resources were formed using a maximum marginal return algorithm.

Iteration over the trajectory for fixed EW and EW for a fixed trajectory continued until convergence. Conceptually both the trajectory and the EW resource optimization problem can be embedded in one dynamic programming problem, but there are some computational issues. Note that if a slow computer is to be used, it is always possible to reduce the computational problem by enlarging the grid size.

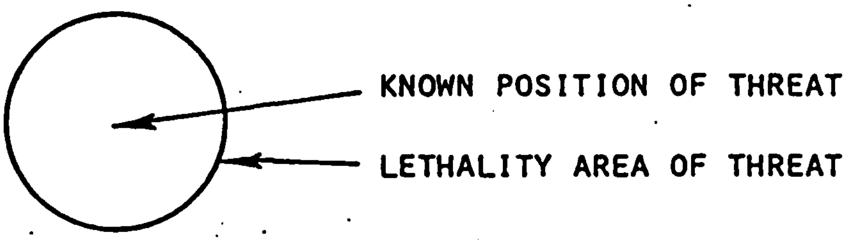
A few example results are now given [1,3]. Figure A.2 shows the symbols used in describing the scenarios and results. Figure A.3 shows a set of enemy threats, two targets, the optimal trajectory, and the optimal EW resources allocation. The threat models used, albeit crude, are three-dimensional models. Although only a two-dimensional aircraft trajectory is shown, in fact, the computer program took into account fuel consumption and probability of survival, P_S, variations with altitude and velocity, and determined an optimal three-dimensional trajectory.

Note in Figure A.3 that the aircraft flies around the threat C1 whose location is uncertain (see Figure A.2) and uses chaff and jamming against A1. The aircraft uses decoys against B1, but does not jam B1. The aircraft uses chaff and jamming against C2.

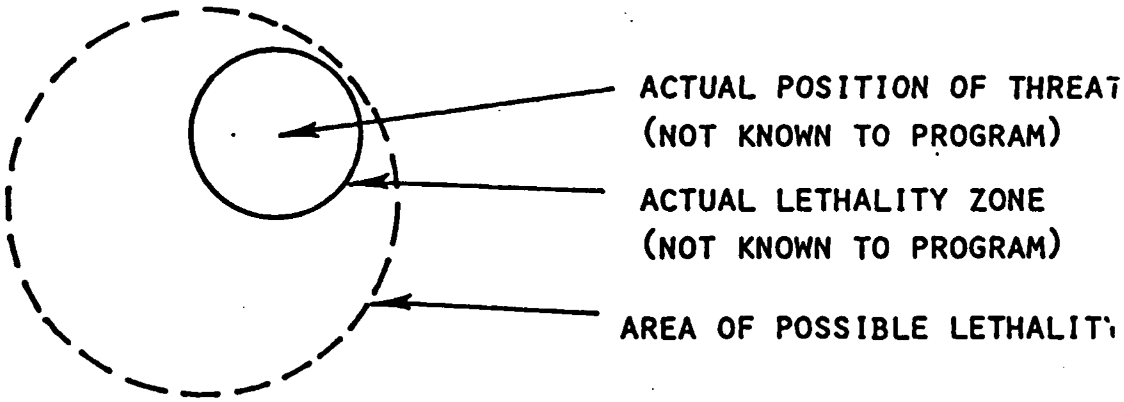
The case where the location of C1 is known is shown in Figure A.4. Here we assume that the aircraft discovers C1 at the end of the dotted line and then computes the optimal trajectory subject to his remaining fuel (14,600 pounds). The new optimal trajectory flies around C1, uses chaff and jamming against A1, and jams B1. Since the aircraft trajectory is shorter in the first leg, it has more fuel remaining and uses it to avoid C2 and now does not use chaff or jamming against C2. The performance measure, current military value, and P_S are, of course, higher in this second case where we know the location of C1.

SYMBOLS USED IN SCENARIOS

- THREATS KNOWN WITH CERTAINTY



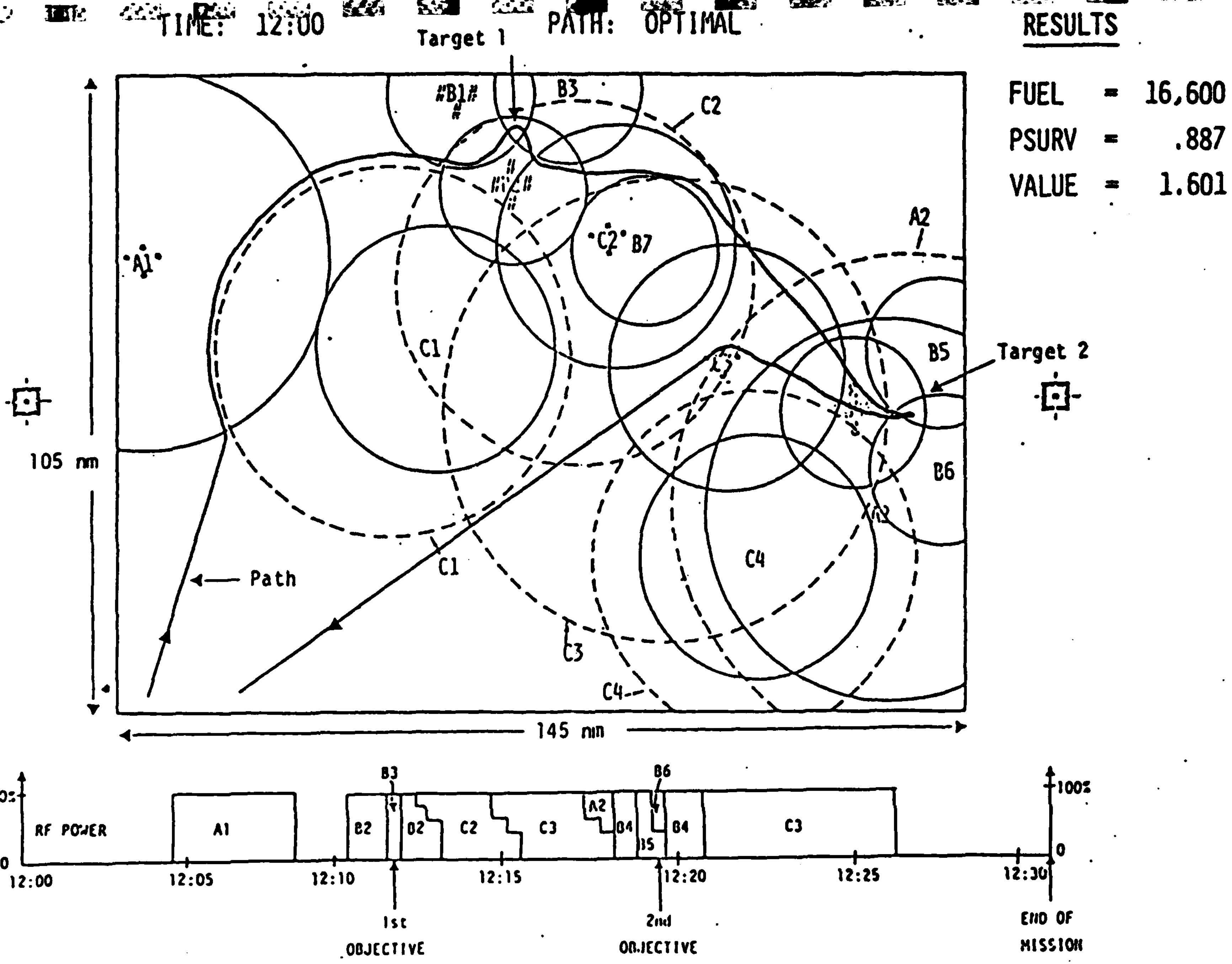
- THREATS WITH UNCERTAINTY IN LOCATION



- EXPENDABLES

```
  #              *               &
# A #  =  DECOY  * A *  = CHAFF  & A &  = ARM
  #              *               &
```

Figure A.2

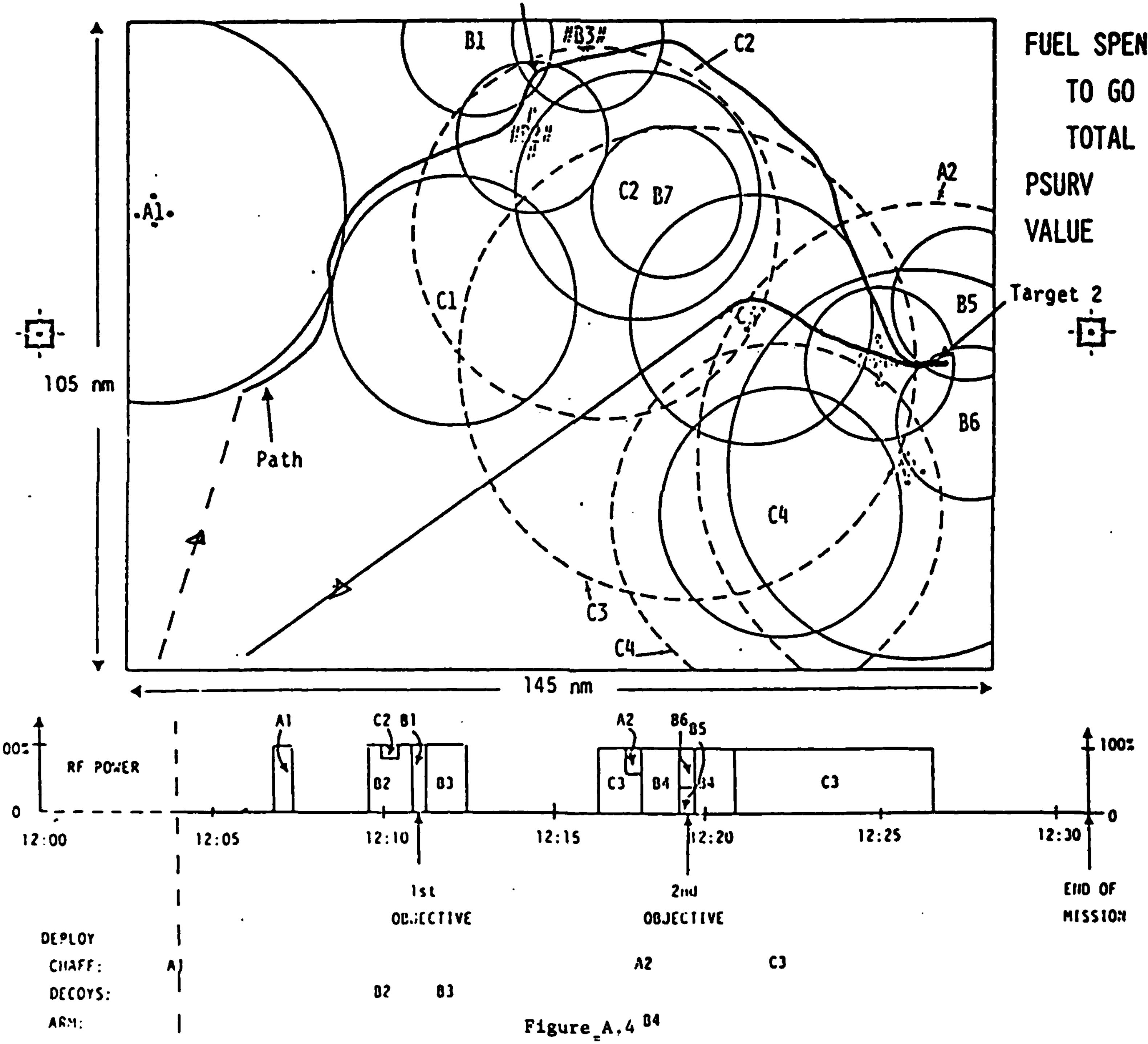


DEPLOY

CHAFF:	A1	C2	C3
DECOYS:		D2 D1	
ARMS:			B4

Figure A.3

TIME: 12:04 PATH: OPTIMAL



RESULTS

FUEL SPENT	= 1,950
TO GO	=14,600
TOTAL	=15,550
PSURV	= .897
VALUE	= 1.611

Figure A.4

APPENDIX B

Depth-First Dynamic Programming Algorithm

INTRODUCTION

In this Appendix, we present a new dynamic programming algorithm. This algorithm is a symbiosis of the traditional DP algorithm and the depth-first branch-and-bound search method. Thus, it can be called Depth-First Dynamic Programming (DFDP).

While guaranteed to have complexity of the same order of magnitude as the traditional DP algorithm in the worst case, DFDP has many advantages over the latter. Among them are:

- Ability to incorporate heuristic, AI guidance methods.
- Ability to rigorously reduce the state space using "branch-and-bound" pruning techniques.
- Suitability for satisficing and for real-time on board computations
- Applicability to the analysis of robustness as an alternative to the "colored corridor" algorithm.

Therefore, DFDP algorithm promises to be helpful in our Dynamic Sequence Assignment research (for both path optimization and mission assignment).

This Appendix is organized as follows:

The first section provides some intuition about the main idea behind the DFDP algorithm.

The next section provides the mathematical background and definitions of dynamic programming systems.

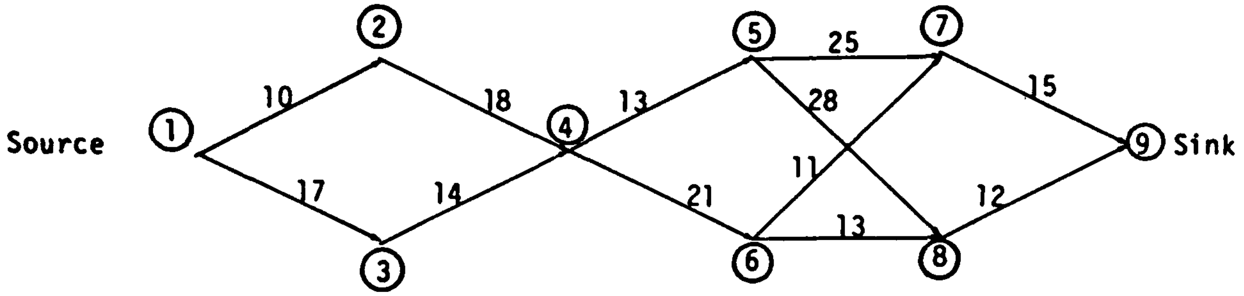
The third section gives a rigorous mathematical definition of the DFDP algorithm in the form of a program flow-chart.

The last section walks the reader through a fairly large example, illustrating the workings of the new algorithm.

1. Intuitive Example

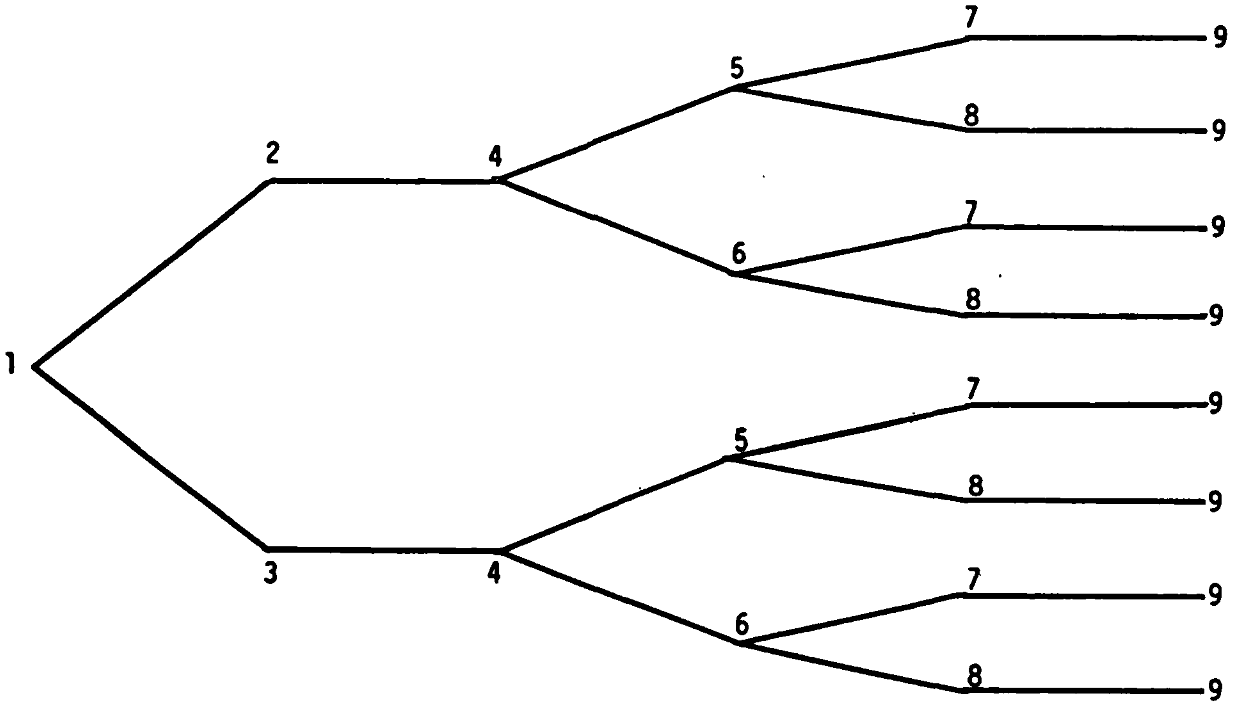
Before we give a rigorous description of this new depth-first dynamic programming (DFDP) algorithm and go through a large example illustrating this algorithm, let us consider a small example which illustrates an advantage that DFDP has over classical tree search.

Consider the problem of finding the shortest path in the following network:



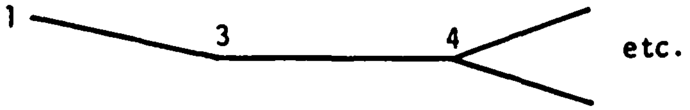
This simple problem can be easily solved by inspection, or by classical DP. But suppose that we didn´t know about DP and tried to solve this problem by enumerating all possible paths and choosing the shortest.

The search tree would look like this:

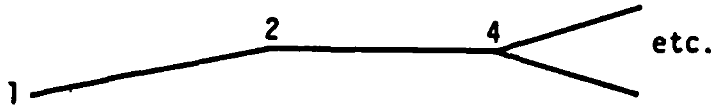


Even for this tiny example, the search tree is pretty large, and it will grow exponentially with the number of stages. However,there is much redundancy and repetition in this search.

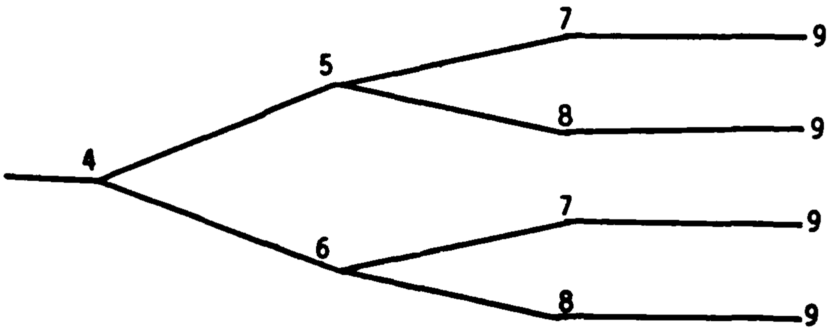
For example, the upper branch of the tree visits Node 4:



but the lower branch also visits Node 4:



and, therefore, the trees from Node 4 on in both cases look the same:



Since the cost function is separable and thus satisfies the optimality principle, we can cut down on computation by first exploring the upper branch and tree, and later, when arriving at Node 4 in the lower branch, just use the optimal cost to go from Node 4, already computed by the first branch.

Of course, the upper branch itself is not free of redundancy, since both Nodes 7 and 8 were visited in two separate places, and, thus, we can achieve even more computational reduction. Both DFDP and classical DP exploit this computational reduction.

2. Mathematical Background

Let us use the traditional definition of Dynamic Programming problem, (see [Larson], for example).

Let X_i, i=0, 1, 2, ..., N be the state-space at stage i.

$X_i \subset R^n$, i=0, 1, 2, ..., N, where i represents the stage variable.

Let U_i (x) be the set of available actions in the state $x \in X_i$ at stage i.

Let the system equation be

(1) $X(k + 1) = g_k(x(k), u(k))$, k=0, 1, ..., N-1

where u(k) is the action chosen at stage k, stage x(k).

Without loss of generality (W.L.O.G.), let us assume that x(0) is specified, and so is x(N), i.e., initial and terminal conditions are specified. Let the cost function

(2) $J(x(0), u(0), x(1), u(1), \ldots, x(n))$

be given. The problem is to find the optimal controls {u(0), u(1),...,u(N-1)}, which minimize the function J within the system defined by

(3)
$x(i) \in X_i$

$u(i) \in U_i\ (x(i))$

x(0) = c - Constant

x(n) = d - Constant

and by the system equation (1).

For the following discussion, let us make the traditional assumption that the
(4) sets $U_i(x)$ are finite for all x and all i (otherwise the problem would be computationally infeasible).

(5) Let $m_i(x)$ be the cardinality of set $U_i(x)$.

Notice that the state-spaces X_i(i ∈ {0, 1, ..., N}) need not be restricted to be finite. In fact, if the assumption (4) is true, then set $R_i(R_i \subset X_i)$ of reachable states is finite for all i. One of the strong points of the proposed algorithm is its ability to handle such situations with ease and without having to pre-generate all reachable state-spaces R .

Nevertheless, for the simplicity of notation, let us assume that, unless
(6) otherwise specified, sets I_i are finite and

(7) let B_i be the cardinality of X_i.

(8) Also define T_i as the cardinality of the set $R_i \subset X_i$.

One of the features of DFDP algorithm is that it can comfortably handle <u>any</u> cost function that satisfies the Markovian, "optimality principle" conditions. But for reasons of simplicity, we shall follow the common practice (see [Larson] [Bellman], etc.) of assuming that J is a separable function:

$$J(x(0), u(0), x(1), u(1), \ldots, x(N)) = \sum_{i=0}^{N-1} L_i(x(i), u(i)). \tag{9}$$

Since the action space $U = \bigcup_{i=0}^{N-1} U_i$ is finite (by (3)), we can assume W.L.O.G.

that L_i is non-negative for any i.

Also, W.L.O.G., we can assume that for a given i and given x, the set of actions

$$U_i(x) = \{u_i^1(x), u_i^2(x) \ . \ . \ u_i^m(x)\}$$

is in the increasing order of L_i, i.e., if $1 \leq j \leq k \leq m_i(x)$, then

$$L_i(x, u_i^j(x)) \leq L_i(x, u_i^k(x)).$$

(Actually, if that is not the case, the algorithm will work just as well; so this is assumed purely for notational convenience).

There is one more assumption: W.L.O.G. we can safely assume that the final state is a sink, i.e., $X_N = \{d\}$. (See (3)).

3. The Algorithm Flow-Chart

Let us define the meanings of the variables used in the flow-chart (Fig. B-0).

s - current state

i - current stage

$I_i(s)$ - the currently shortest path length (cost-to-go) from state s to the final state d

$BC_i(s)$ - the currently best control at state s

$F_i(s)$ - the currently shortest path length from the starting state c to state s

Expl. [s, i] - this boolean variable is equal to 1 if the state s has been explored completely, (i.e., if the optimal cost-to-go from stage s is known), and 0 otherwise

Predecessor [s] - the latest state from which we arrived at state s

Cur.Opt. - the currently shortest path between the starting state (c, i) and terminal state (d, i)

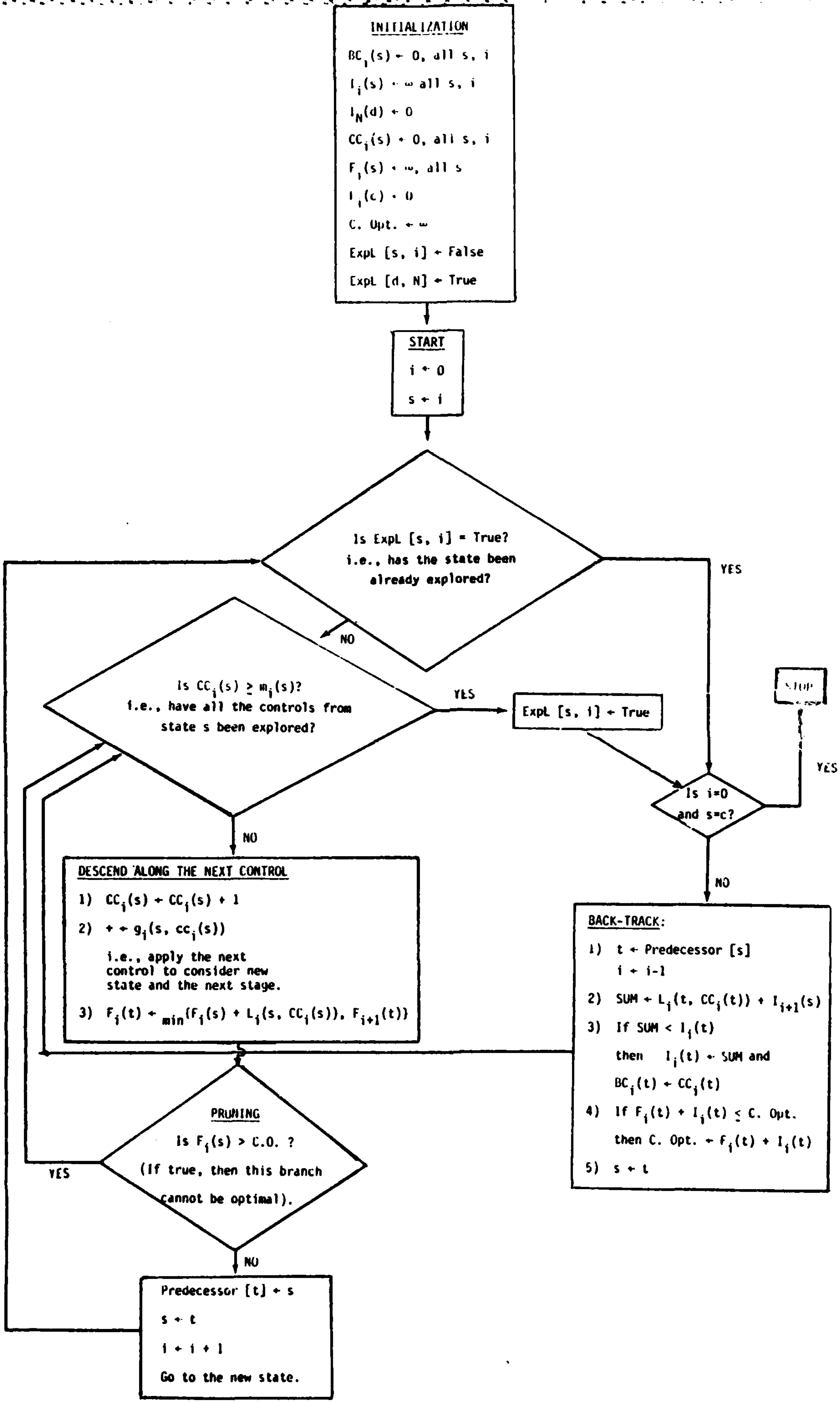

INITIALIZATION
$BC_i(s) \leftarrow 0$, all s, i
$I_i(s) \leftarrow \infty$ all s, i
$I_N(d) \leftarrow 0$
$CC_i(s) \leftarrow 0$, all s, i
$F_i(s) \leftarrow \infty$, all s
$I_i(c) \leftarrow 0$
C. Opt. $\leftarrow \infty$
ExpL [s, i] ← False
ExpL [d, N] ← True
START
$i \leftarrow 0$
$s \leftarrow i$
Is ExpL [s, i] = True? i.e., has the state been already explored?
YES
NO
Is $CC_i(s) \geq m_i(s)$? i.e., have all the controls from state s been explored?
YES
ExpL [s, i] ← True
STOP
YES
Is i=0 and s=c?
NO
NO
DESCEND ALONG THE NEXT CONTROL
1) $CC_i(s) \leftarrow CC_i(s) + 1$
2) $t \leftarrow g_i(s, cc_i(s))$
i.e., apply the next control to consider new state and the next stage.
3) $F_i(t) \leftarrow \min(F_i(s) + L_i(s, CC_i(s)), F_{i+1}(t))$
BACK-TRACK:
1) t ← Predecessor [s]
$i \leftarrow i-1$
2) $SUM \leftarrow L_i(t, CC_i(t)) + I_{i+1}(s)$
3) If $SUM < I_i(t)$
then $I_i(t) \leftarrow SUM$ and
$BC_i(t) \leftarrow CC_i(t)$
4) If $F_i(t) + I_i(t) \leq$ C. Opt.
then C. Opt. $\leftarrow F_i(t) + I_i(t)$
5) $s \leftarrow t$
PRUNING
Is $F_i(s) >$ C.O. ?
(If true, then this branch cannot be optimal).
YES
NO
Predecessor [t] ← s
$s \leftarrow t$
$i \leftarrow i + 1$
Go to the new state.

Fig. B-0. High-level Flow-chart of the DFDP Algorithm

C $C_i(s)$ - the currently considered control at state s

All the other variables and constants in the flow chart have been defined in the previous section.

4. A detailed Example

Let us illustrate the DFDP algorithm with a detailed example. Please refer to Fig. B-1 through B-17.

1) The problem is finding a minimum cost path through the network in Fig. B-1. The node numbers are given above each node, the transition cost for each arc is written above it.

The starting state is 1, the terminal state is 22.

2) Fig. B-2. At state 1, we chose arc (1, 2) because it has the lowest cost of the arcs originating from state 1.

Thus we come to state 2. The circle above each state represents the data stored for this state. The top number (in this case it is 6) represents the value of F(s), the shortest known (at present) path length from the starting state.

The bottom number (in this case it is infinity) represents the value of I(s), the currently shortest known path length to the terminal state. At present it is infinite because no path to the terminal has been found yet.

At state 2 we chose to transition to state 7 because that is closer to 2 than state 8.

The next transition is to state 12.

3) Figure B-3. Here we followed our path all the way to state 22, the terminal state. Note that the arrows originating from the visited states represent the best currently known control at that state.

Notice the circles above state 22. The number 31 is, of course, the length of the path we've traveled.

The check mark below represents the fact that all paths from that state have been explored (there aren't any). The number 0 below the check represents the cost-to-go.

4) Figure B-4. Here we see that we have found a path of length 31 from start to finish. Thus the value 31 gets placed on the bottom of the circle of state 1 as the current optimum.

Now we back-track to state 17. We put 10 as the currently best cost-to-go, and looked for another arc to take from state 17. There are none. Thus state 17 has been explored. We put a check on it, and 10 becomes the official cost-to-go from that state.

Now we need to back-track more.

5) Figure B-5. We back-tracked to state 12. The current best cost-to-go from 12 is equal to the cost-to-go from state 17 plus the cost of arc between 12 and 17, that is 10 + 2 = 12.

From state 12, we chose another arc to traverse, and visited state 18. The number in the circle there is equal to the "length-from-start" to state 12 plus the arc length of arc (12, 18), which is 19 + 5 = 24.

From state 18, we took arc (18, 22) and came to state 22 with the total length traveled being 29.

6) Figure B-6. Since 29 is better than 31, we put 29 as the current optimum from state 1.

Since there are no other arcs originating from state 18, we marked it as checked, with cost-to-go being 5.

Then we back-tracked to state 12. Notice that the total cost-to-go from state 12 following arc (12, 18) is 5 + 5 = 10, which is better than the 12 currently there. Thus, we should substitute 10 for 12 on the bottom of the circle, and also point to arc (12, 18) as the new best known arc-to- take.

7) Figure B-7. As you can see, we did that. Since there are no other arcs to try from state 12, we back-tracked to state 7. The cost-to-go from there becomes 10 + 10 = 20. Then we try another arc, which takes us to state 13.

8) Figure B-8. From state 13 we go to state 17. Since it has a check mark, we don´t have to do anything with it, just copy the cost-to-go from it (which is 10) and back-track.

9) Figure B-9. We back-tracked and came back with 10 + 4 = 14 as the cost-to-go from state 13.

Since the two numbers in the circle above that state add up to 35, which is larger than 29, no change to the current-best-solution need to be made.

10) Figure B-10. Then we explored arc (13, 19), and found that taking it costs a total of 12, which is better than 14, the previous best.

Then we wanted to visit state 19. But because the total length is already 38, and we are not "there" yet, we know that this path cannot be optimal, so we back-tracked.

11) Figure B-11. We back-tracked to state 13, put a check on it with 12 as the minimal cost-to-go.

Then we back-tracked to state 7 to discover that 12 + 12 = 24 is not as good a cost-to-go as 20, so no changes need to be made.

State 7 was done, we put a check and back-tracked to state 2 and put 20 + 3 as the current cost-to-go. Then we went to state 8 and put 6 + 6 = 12 as the current cost-to-get-there.

12) Figure B-12. From state 8 we went to state 12. Since it has a check on it, we just recorded its cost-to-go 10 and are ready to back-track.

13) Figure B-13. The cost-to-go from state 8 becomes 10 + 4 = 14. Notice that the two numbers in the circle above state 8 add up to 26, which is smaller than 29 - the currently best known total cost.

14) Figure B-14. Thus we put 26 as the new current optimum.

Then from state 8 we went to state 13. Since it had a check on it we back-tracked with 9 + 14 = 23 as the cost-to-go, which is much larger than 14 that we currently had.

Then we wanted to go to state 14, only to discover that the cost of arc (8, 14) is 21 which in itself is larger than the current cost-to-go 14 + 6 = 20, which is better than 23 that is currently there.

15) Figure B-15. So we changed that 23 to 20, put a check mark on state 2 and back-tracked to state 1. Then we took arc (1, 3) with cost 11.

From state 3, we first went to state 7, and since it has a check mark on it, we didn´t have to go any farther, we recorded 20 as the cost-to-go, added it to 2 (the cost of arc (3, 7) and put 22 as the current cost-to-go from state 3.

Then we went to state 8, recorded its cost-to-go 14, added it to 5, and came up with a better cost-to-go: 19.

Then we visited state 9 on the arc of length 18, which brought the total cost-to-get-there to 29, which is larger than the current best total cost 26. No further exploration is needed.

16) Figure B-16. Thus we were done with state 3, back-tracked to state 1 and tried state 4 next on an arc of length 14.

From state 4 we went to state 8 which had a check and gave us 14 + 9 = 23 as the cost-to-go. Then we tried state 9, but the cost-to-come added up to 27, which is too large already. Then we tried state 10, but that gave us 29, no good either.

Thus state 4 was explored, we put a check next to it, and back-tracked.

We then tried going to states 5 and 6, but both of them required more than 26 units of cost, so we didn´t have to explore them.

Thus the exploration has been done. The optimal solution is of length 26.

17) Figure B-17. Following the arrows, which represent the best-arc-to-take from each explored state, we can now reconstruct the optimal solution path.

If we look back at Figure B-16, we shall see that we needed to explore only 26 out of the total of 49 arcs in this network.

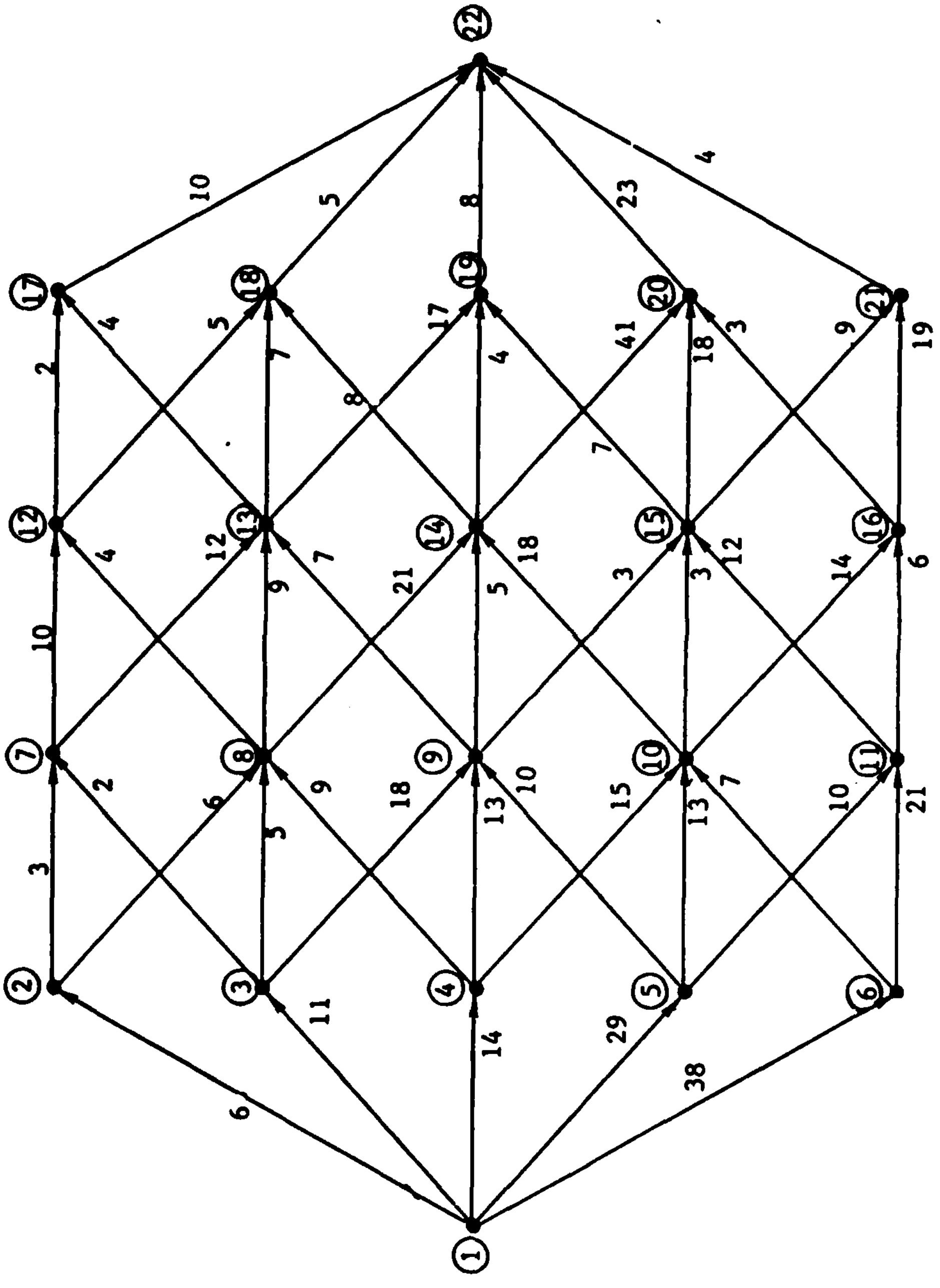

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
6
11
14
29
38
3
2
6
5
9
18
13
10
15
13
7
10
21
10
4
12
9
7
21
5
18
3
3
12
14
6
2
4
5
7
8
17
4
41
7
18
3
9
19
10
5
8
23
4

Figure B-1

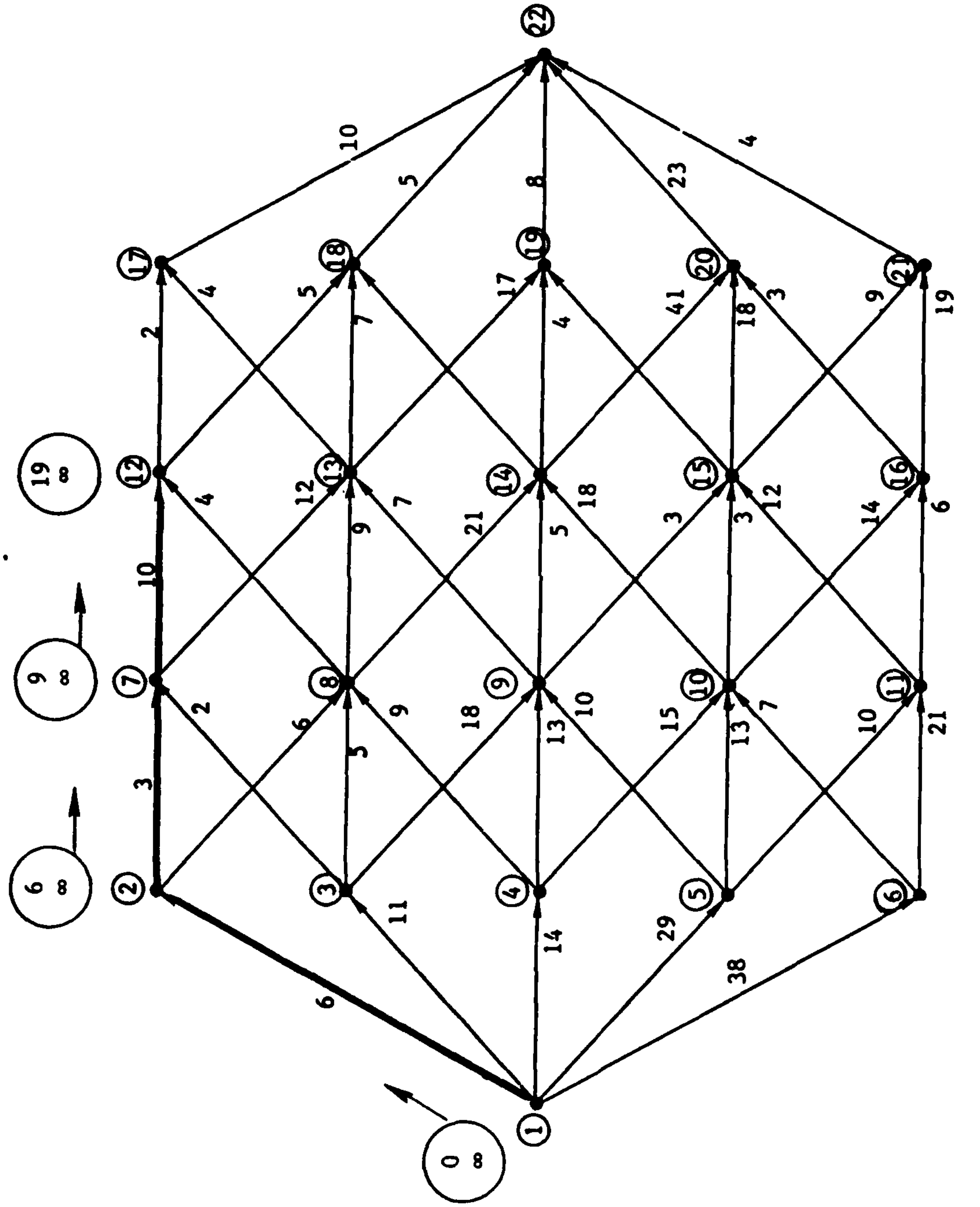

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
0
∞
6
∞
9
∞
19
∞
6
11
14
29
38
3
2
6
5
9
18
13
10
15
13
7
10
21
10
4
12
9
7
21
5
18
3
3
12
14
6
2
4
5
7
17
4
41
18
3
9
19
10
5
8
23
4

Figure B-2

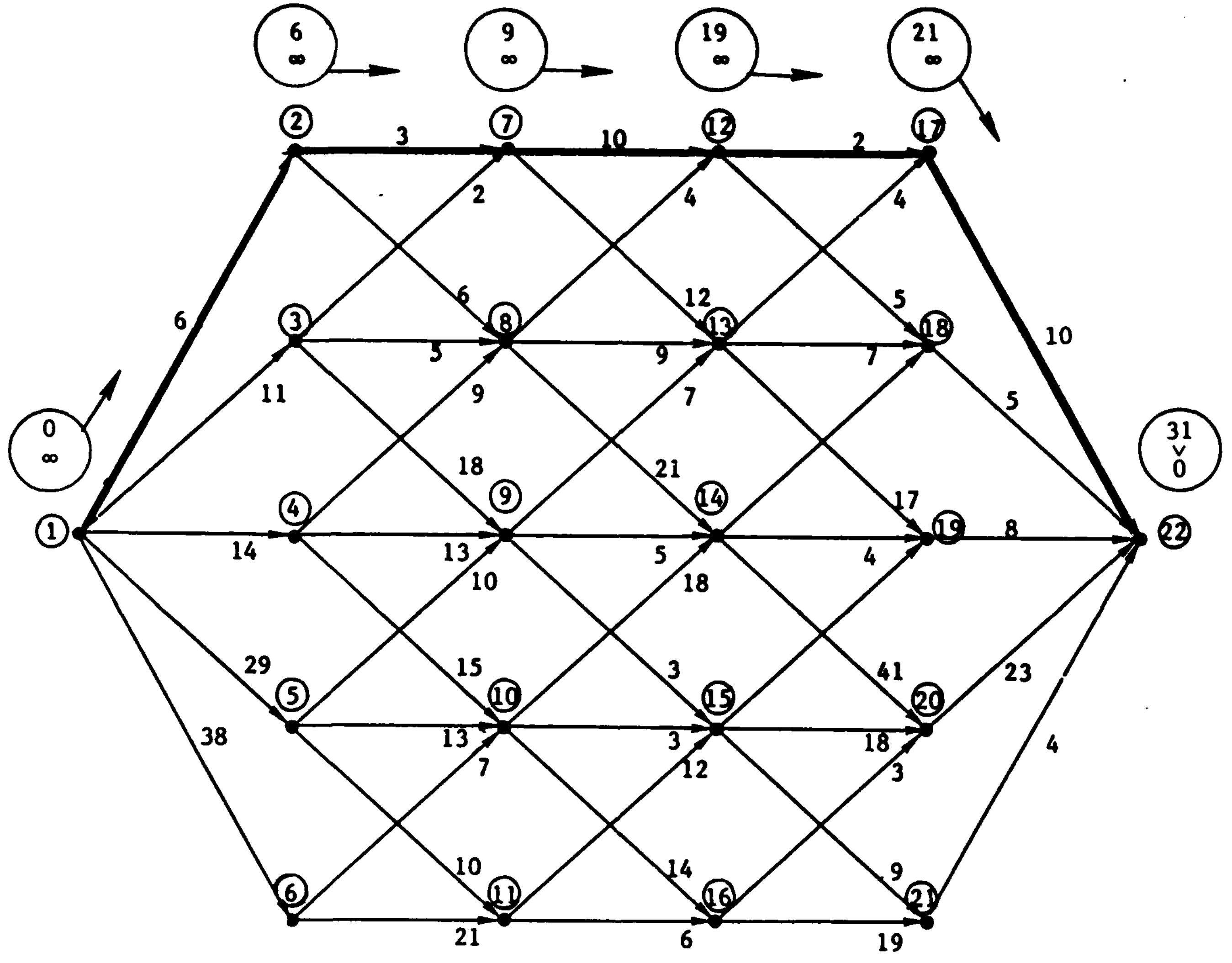
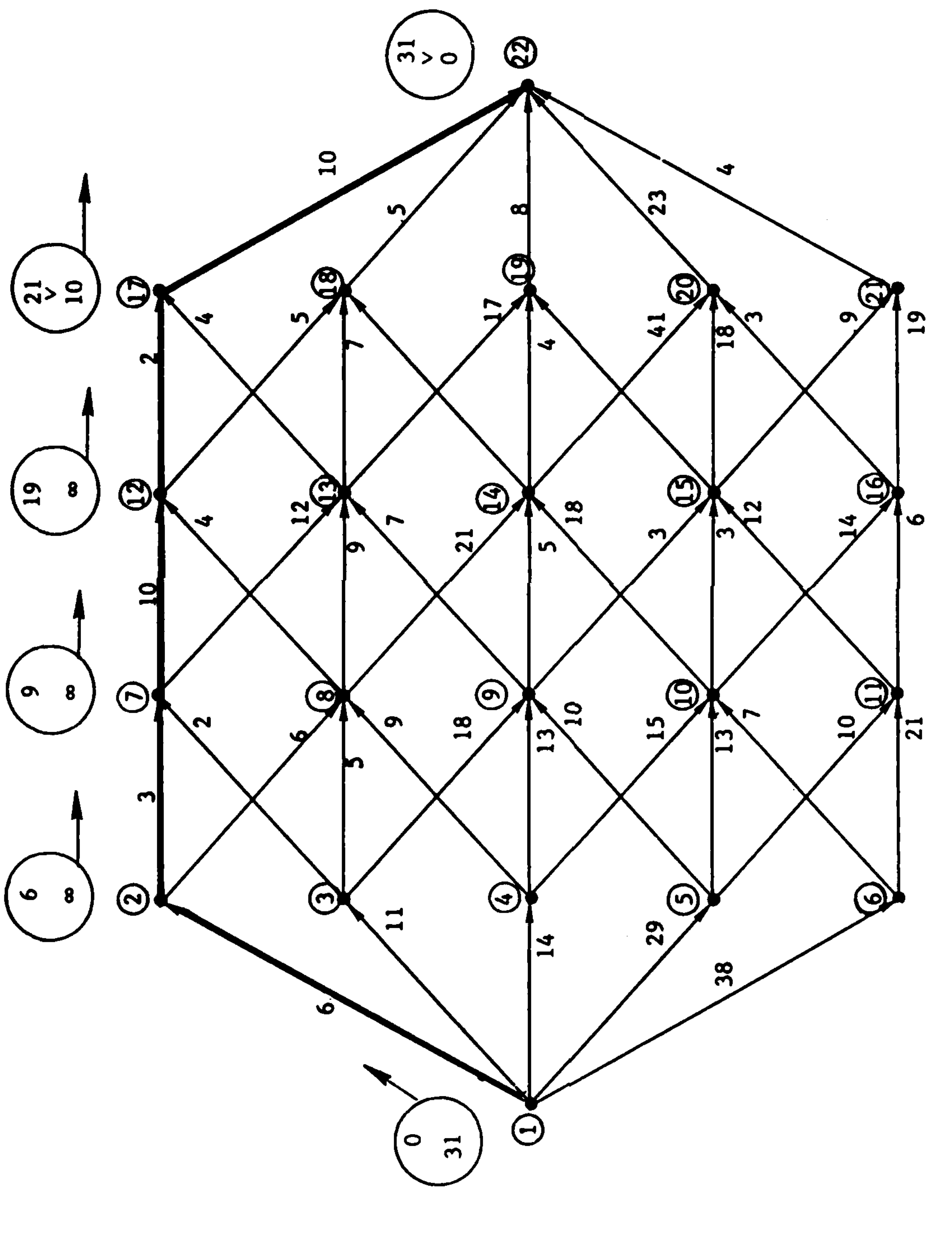

6
∞
9
∞
19
∞
21
∞
0
∞
31
>
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
6
11
14
29
38
3
10
2
2
6
5
9
18
13
10
15
13
7
10
21
4
12
9
7
21
5
18
3
3
12
14
6
4
5
7
17
4
41
18
3
9
19
10
5
8
23
4

Figure B-3



0
31
6
∞
9
∞
19
∞
21
>
10
31
>
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
6
11
14
29
38
3
5
13
13
21
2
6
9
18
10
15
7
10
10
9
5
3
6
4
12
7
21
18
3
12
14
2
7
4
18
19
4
5
17
41
3
9
10
5
8
23
4

Figure B-4

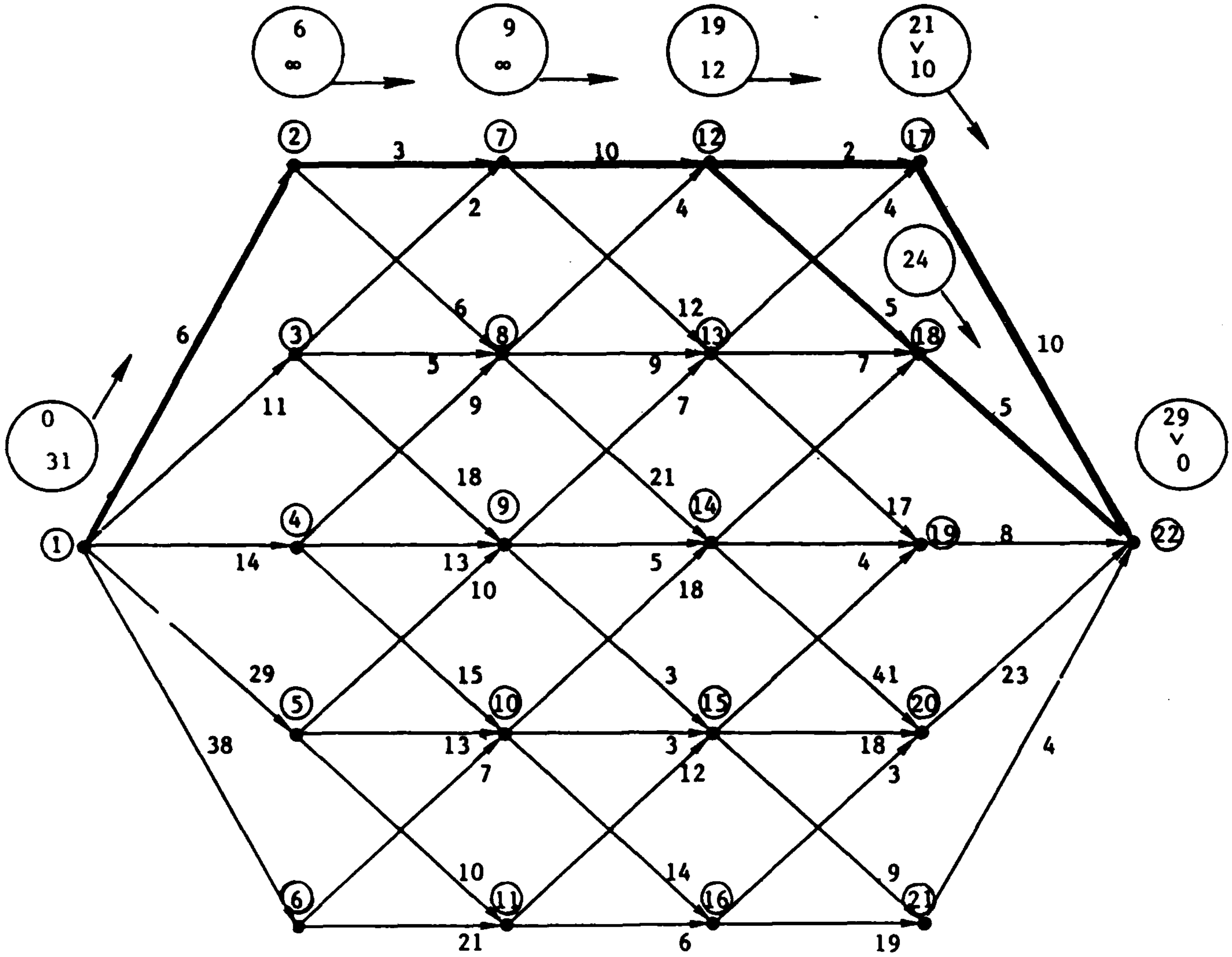

6 ∞
9 ∞
19 12
21 > 10
24
0 31
29 > 0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
6
11
14
29
38
3
10
2
2
4
4
6
12
5
5
9
7
10
5
9
7
18
21
17
8
13
5
4
10
18
15
3
41
23
13
3
18
7
12
3
10
14
9
4
21
6
19

Figure B-5

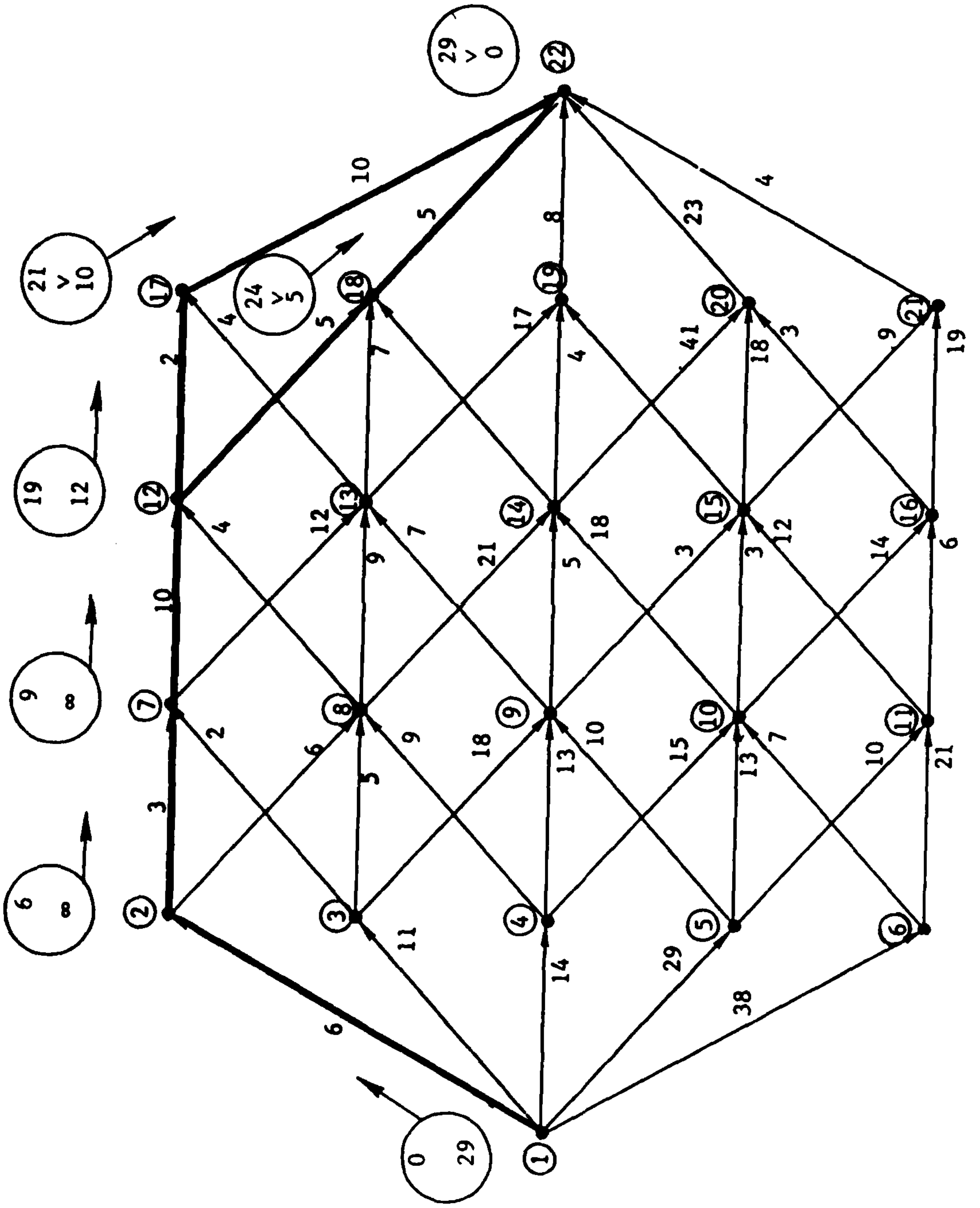
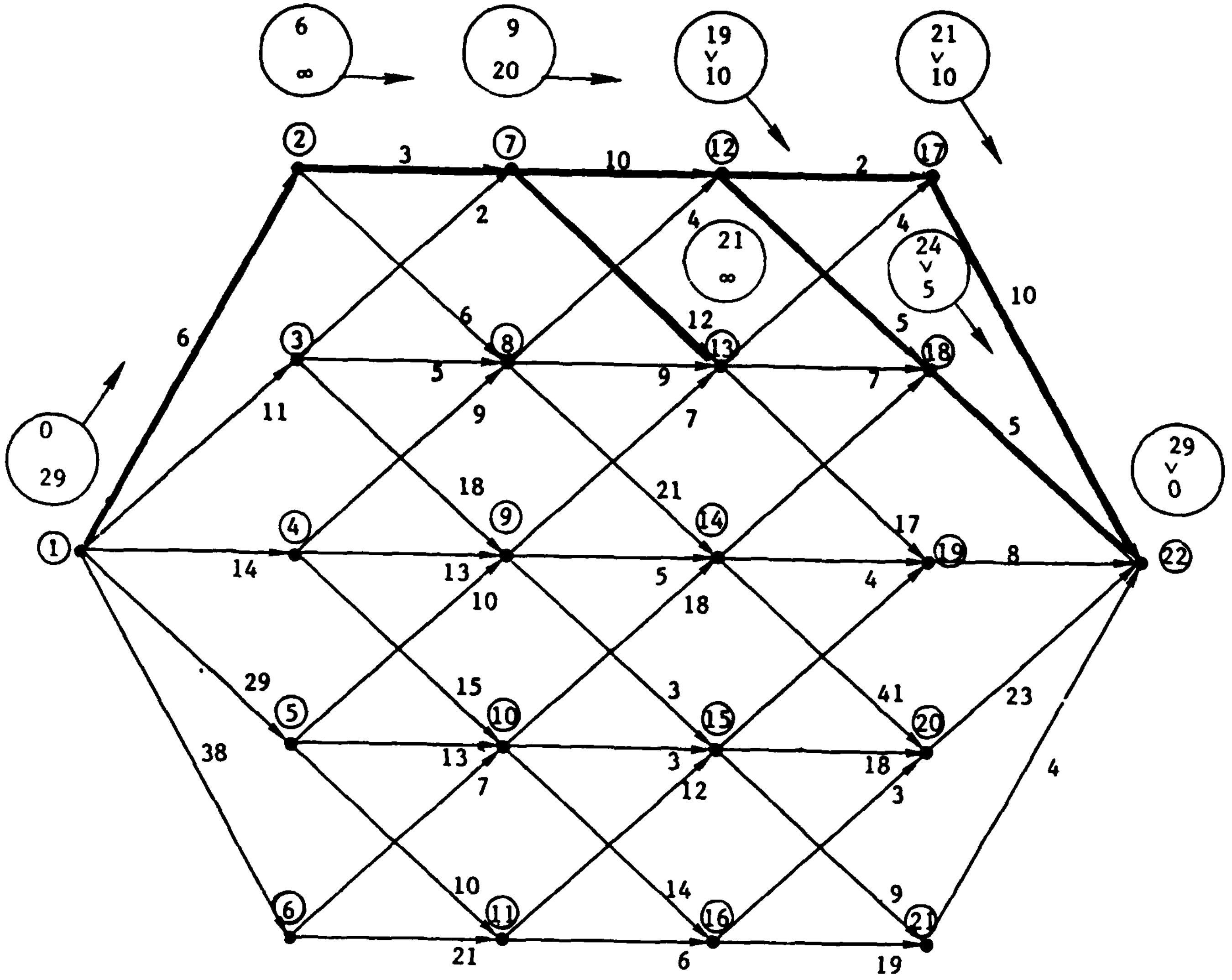

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
0
29
6
8
9
8
19
12
21
>
10
24
>
5
29
>
0
6
11
14
29
38
3
5
13
13
21
2
6
9
18
10
15
7
10
10
9
5
3
6
4
12
7
21
18
3
12
14
2
7
4
18
19
4
5
17
41
3
9
10
5
8
23
4

Figure B-6



0
29
6
∞
9
20
19
v
10
21
v
10
21
∞
24
v
5
29
v
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
6
11
14
29
38
3
2
6
5
9
18
13
10
15
13
7
10
21
10
4
12
9
7
21
5
18
3
3
12
14
6
2
4
5
7
17
4
41
18
3
9
19
10
5
8
23
4

Figure B-7

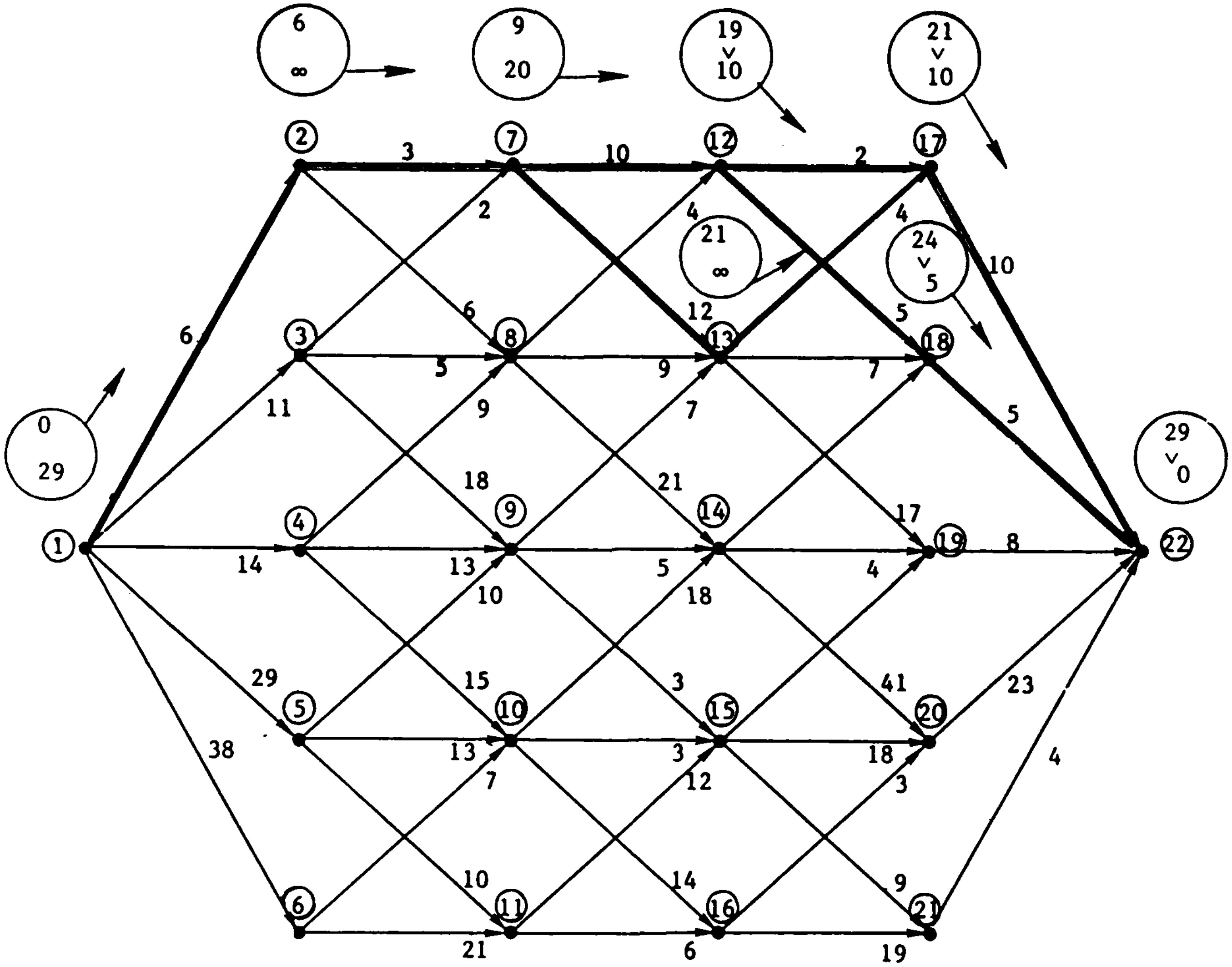
6
∞
9
20
19
>
10
21
>
10
21
∞
24
>
5
0
29
29
>
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
6
11
14
29
38
3
10
2
2
6
5
9
18
13
10
15
13
7
10
21
4
12
9
7
21
5
18
3
3
12
14
6
4
5
7
17
4
41
18
3
9
19
10
5
8
23
4

Figure B-8

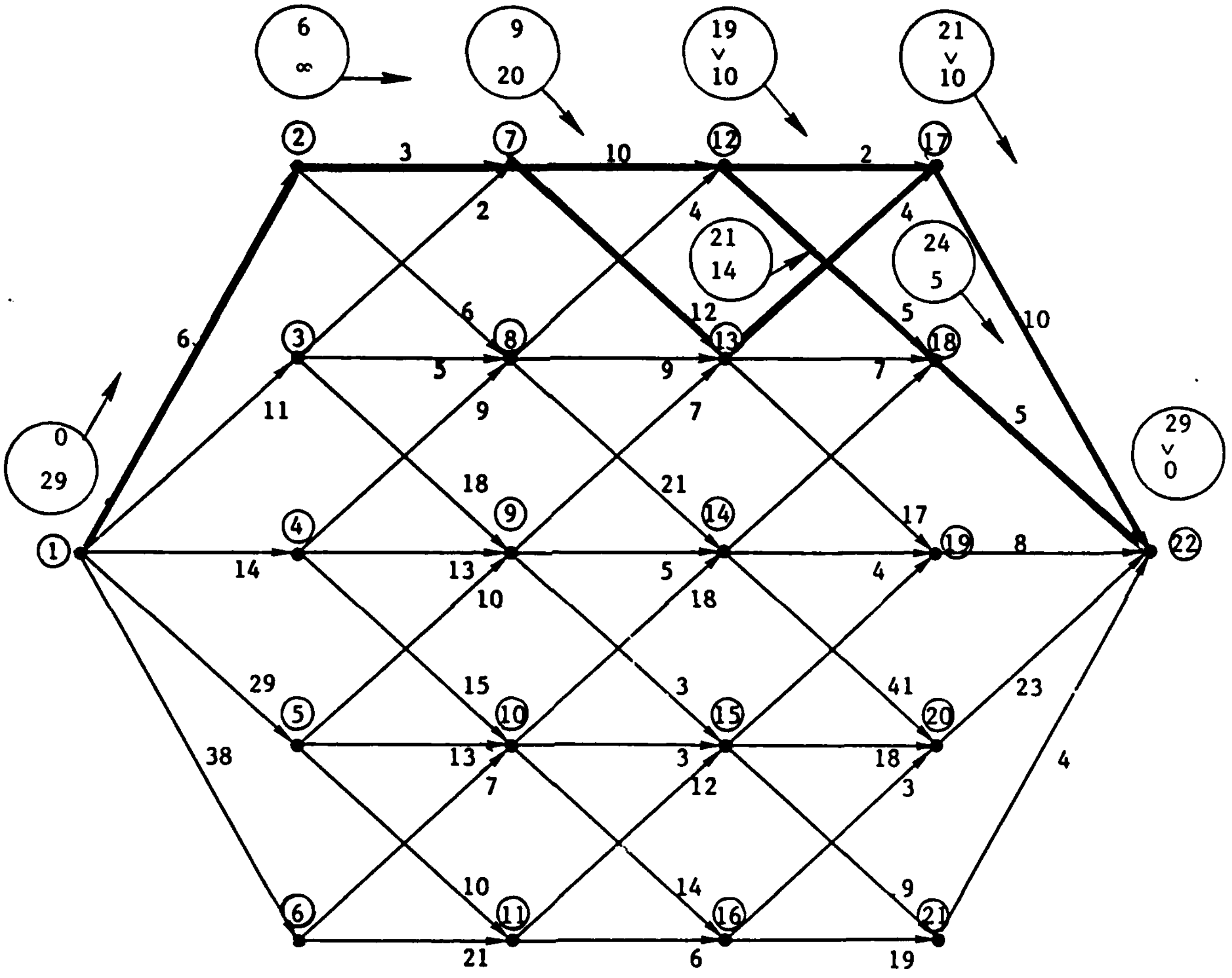

6
∞
9
20
19
>
10
21
>
10
21
14
24
5
0
29
29
>
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
6
11
14
29
38
3
10
2
2
6
5
9
18
4
12
9
7
21
4
5
7
17
10
13
5
18
4
8
15
13
7
3
3
12
41
18
3
10
21
14
6
9
19
23
4
10
5

Figure B-9

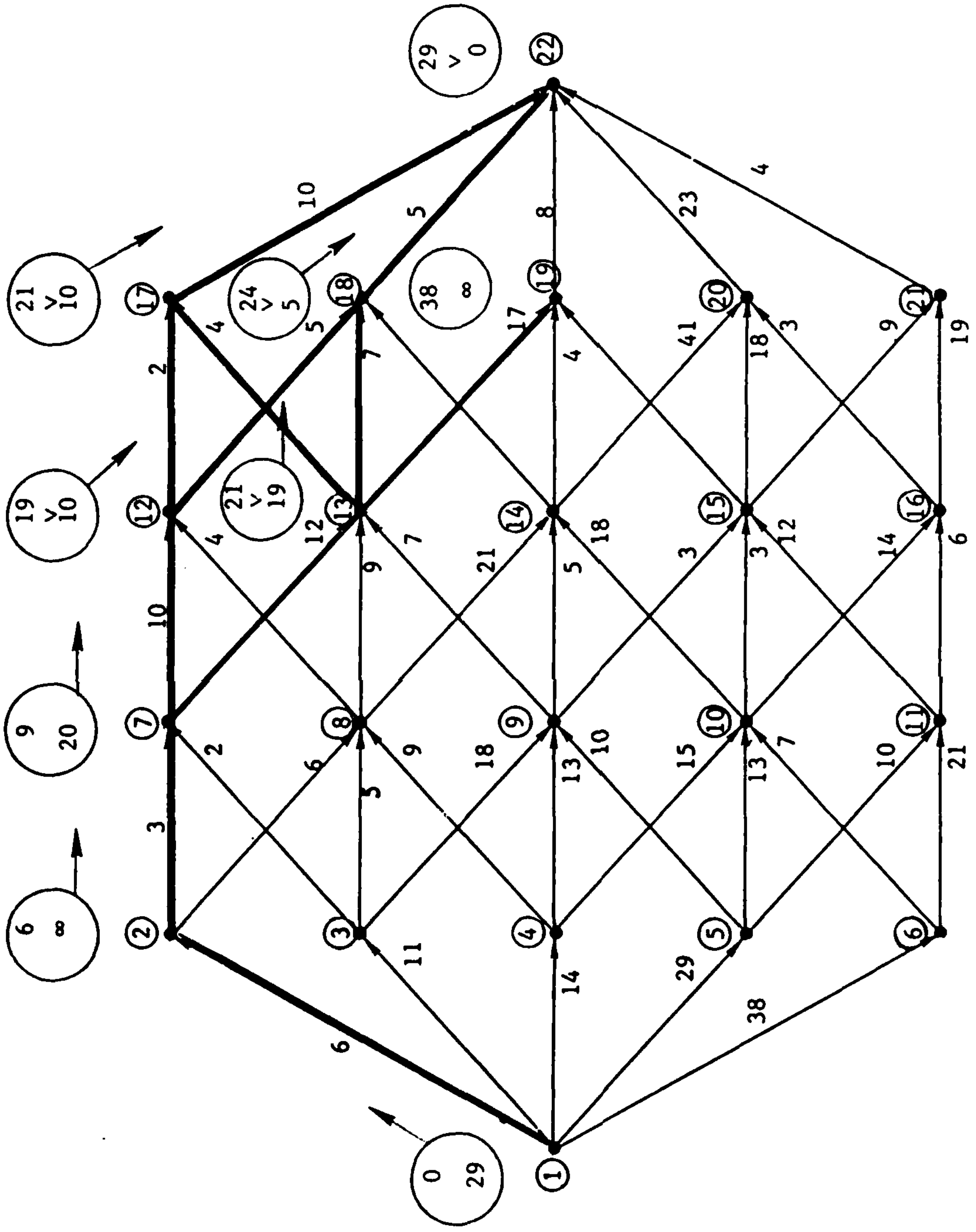

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
0
29
6
∞
9
20
19
>
10
21
>
19
21
>
10
24
>
5
38
∞
29
>
0
6
11
14
29
38
3
2
6
5
9
18
13
10
15
13
7
10
21
10
4
12
9
7
21
5
18
3
3
12
14
6
2
4
5
7
17
4
41
18
3
9
19
10
5
8
23
4

Figure B-10

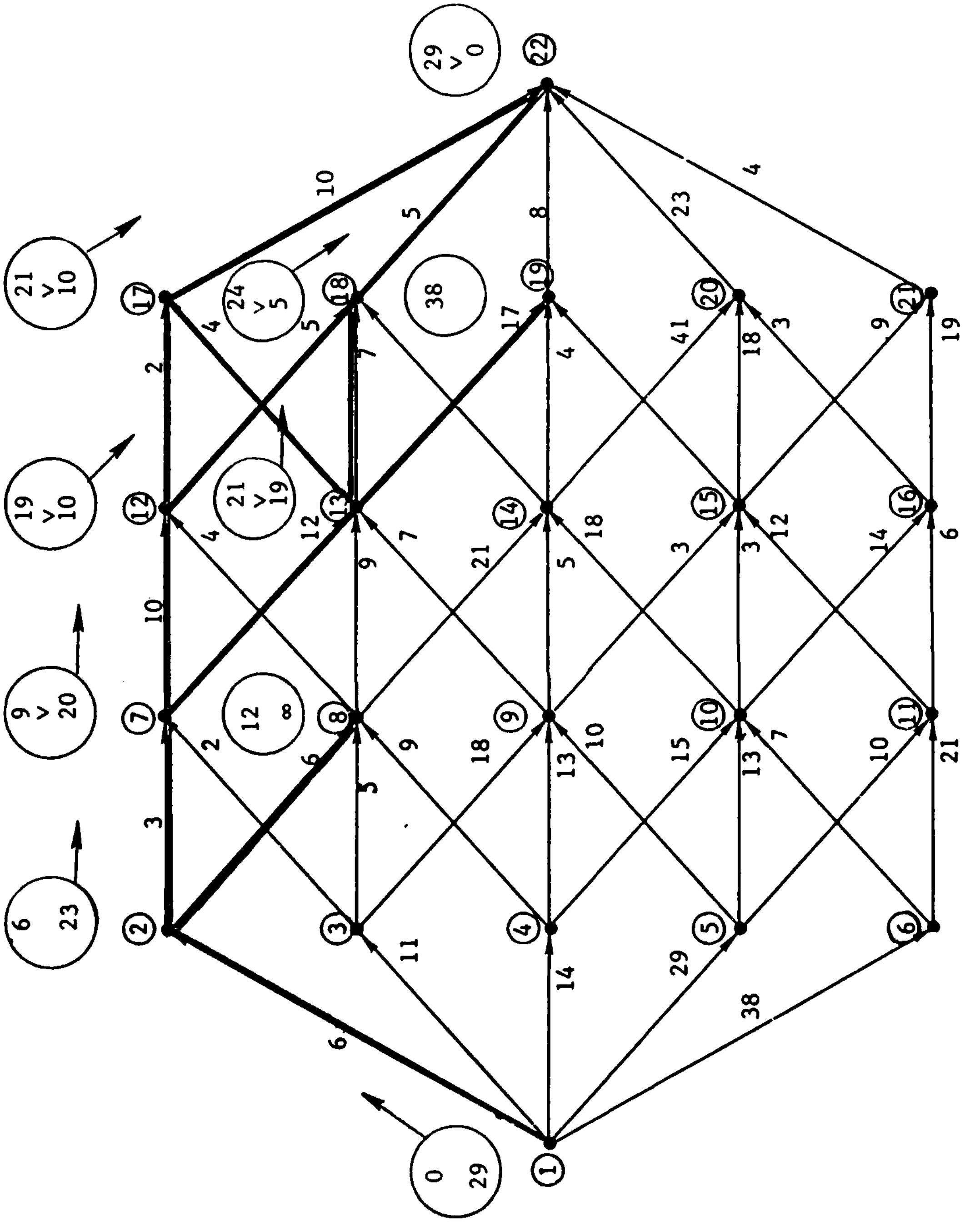

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
0
29
6
23
9
>
20
19
>
10
21
>
10
12
∞
21
>
19
24
>
5
38
29
>
0
6
11
14
29
38
3
2
5
6
9
18
13
10
15
13
7
10
21
10
4
12
9
7
21
5
18
3
3
12
14
6
2
4
5
7
17
4
41
18
3
9
19
10
5
8
23
4

Figure B-11

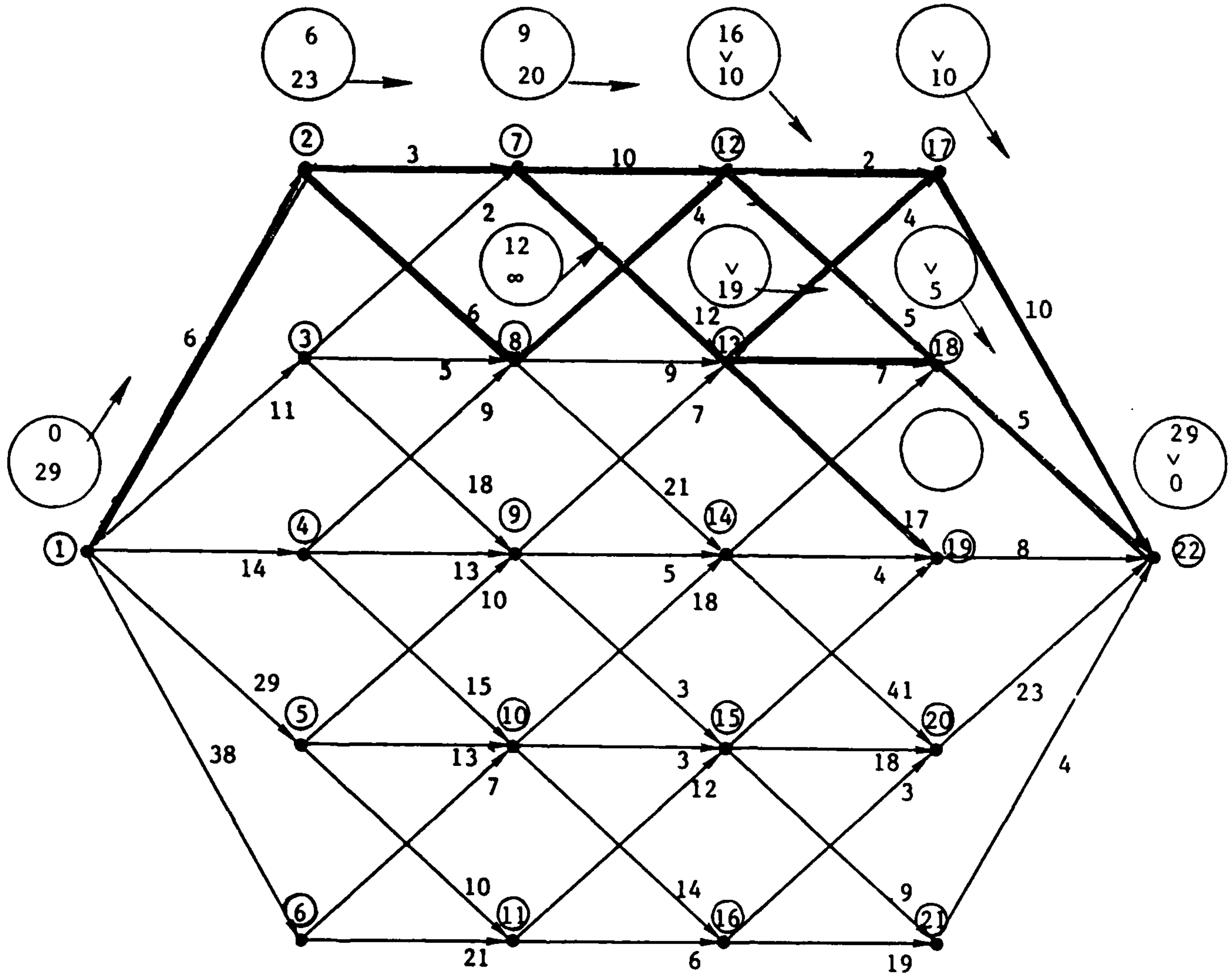
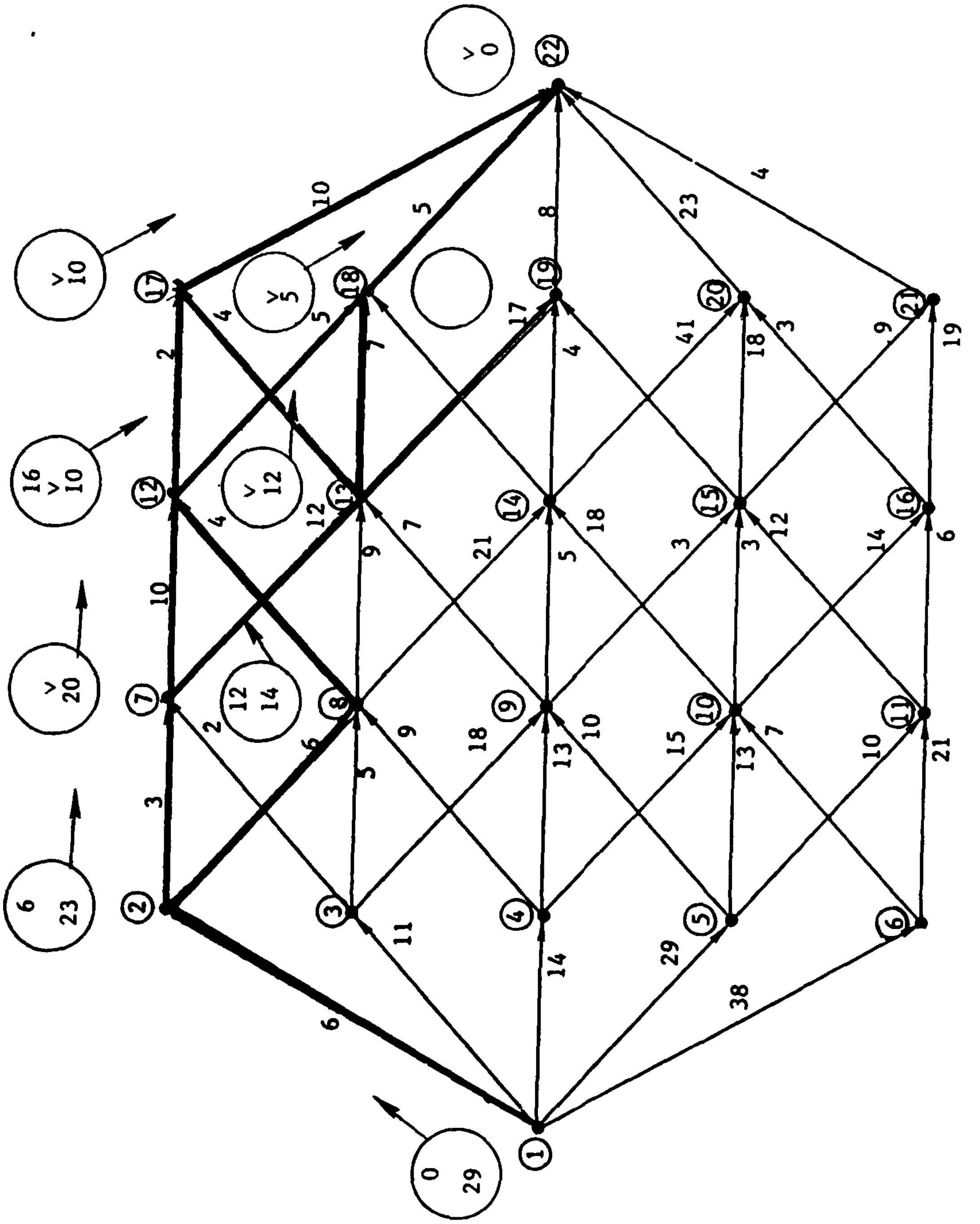

0
29
6
23
9
20
16
>
10
>
10
12
∞
>
19
>
5
29
>
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
6
11
14
29
38
3
2
5
9
18
10
15
7
10
21
13
13
6
4
12
9
7
21
5
18
3
3
12
14
6
2
4
5
7
17
4
41
18
3
9
19
10
5
8
23
4

Figure B-12



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
0
29
6
23
>
20
12
14
16
>
10
>
12
>
10
>
5
>
0
6
11
14
29
38
3
2
5
6
9
18
13
10
15
13
7
10
21
10
4
9
7
21
5
18
3
3
12
14
6
2
4
5
7
17
4
41
18
3
9
19
10
5
8
23
4

Figure B-13

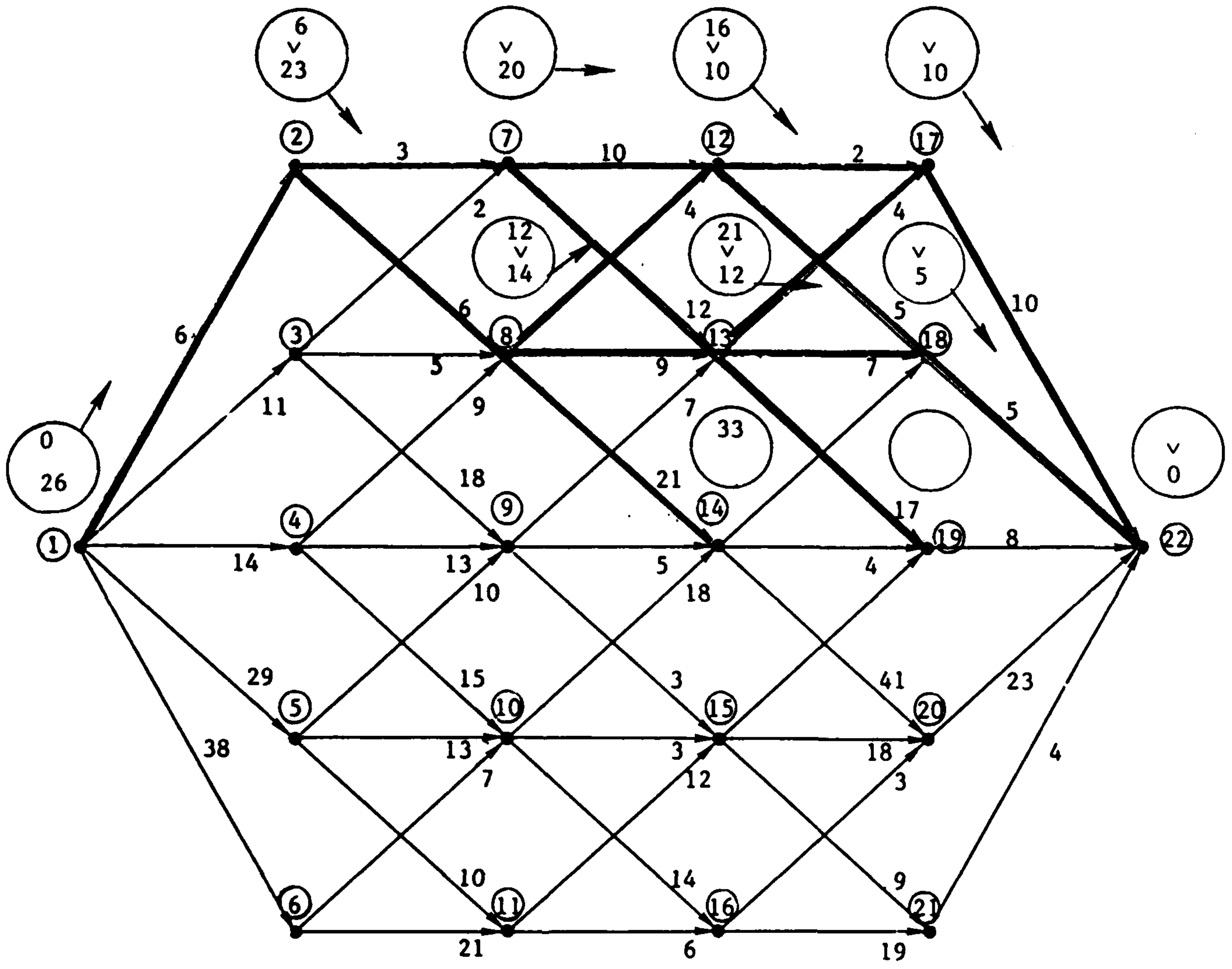
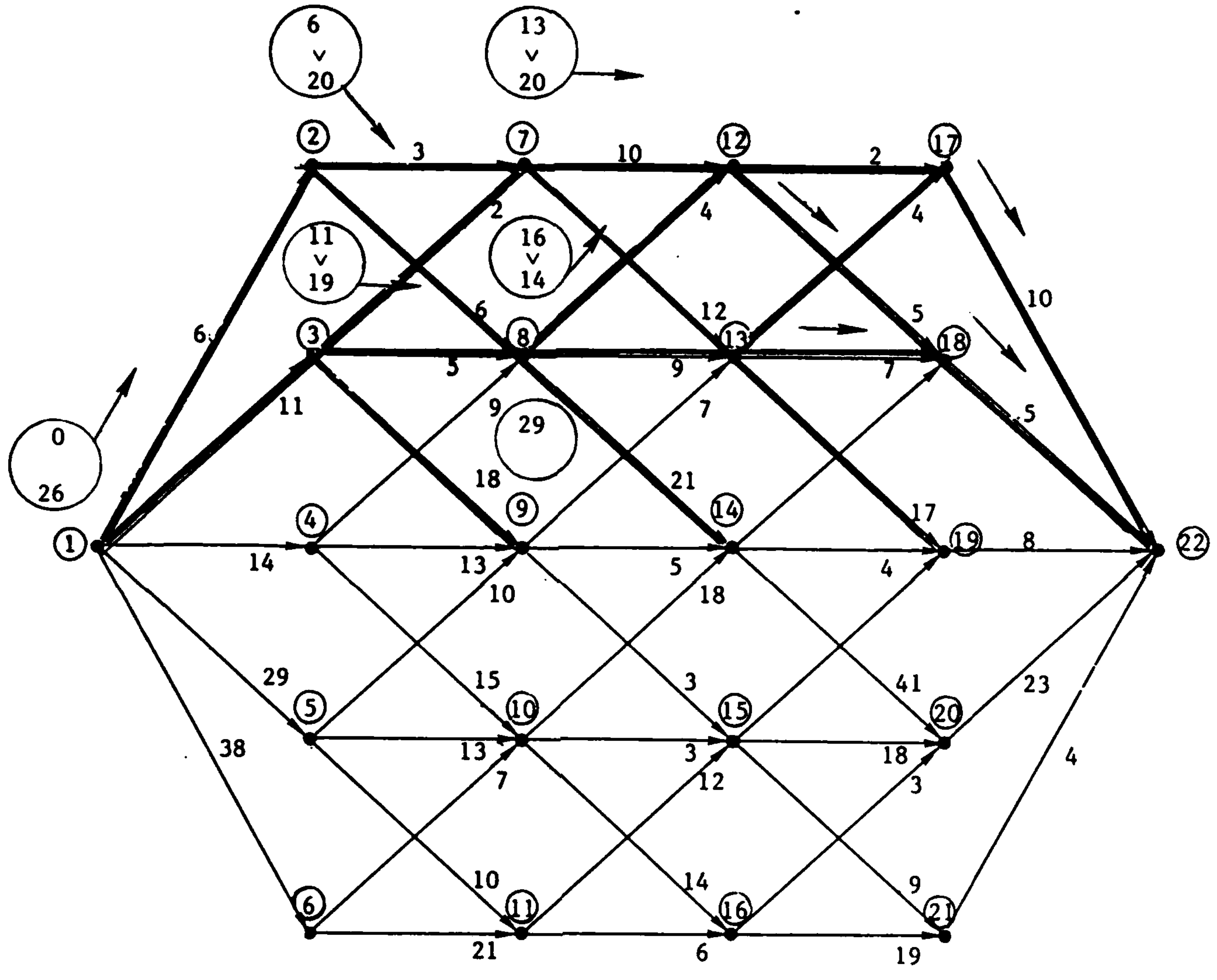

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
0 26
6 > 23
> 20
16 > 10
> 10
12 > 14
21 > 12
> 5
33
> 0
6
11
14
29
38
3
2
6
5
9
18
13
10
15
13
7
10
21
10
4
12
9
7
21
5
18
3
3
12
14
6
2
4
5
7
17
4
41
18
3
9
19
10
5
8
23
4

Figure B-14



6 > 20
13 > 20
11 > 19
16 > 14
29
0
26
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
3
10
2
6
11
14
29
38
2
4
4
10
6
12
5
5
9
7
5
9
7
18
21
17
8
13
5
4
10
18
15
3
41
23
13
3
18
7
12
3
4
10
14
9
21
6
19

Figure B-15

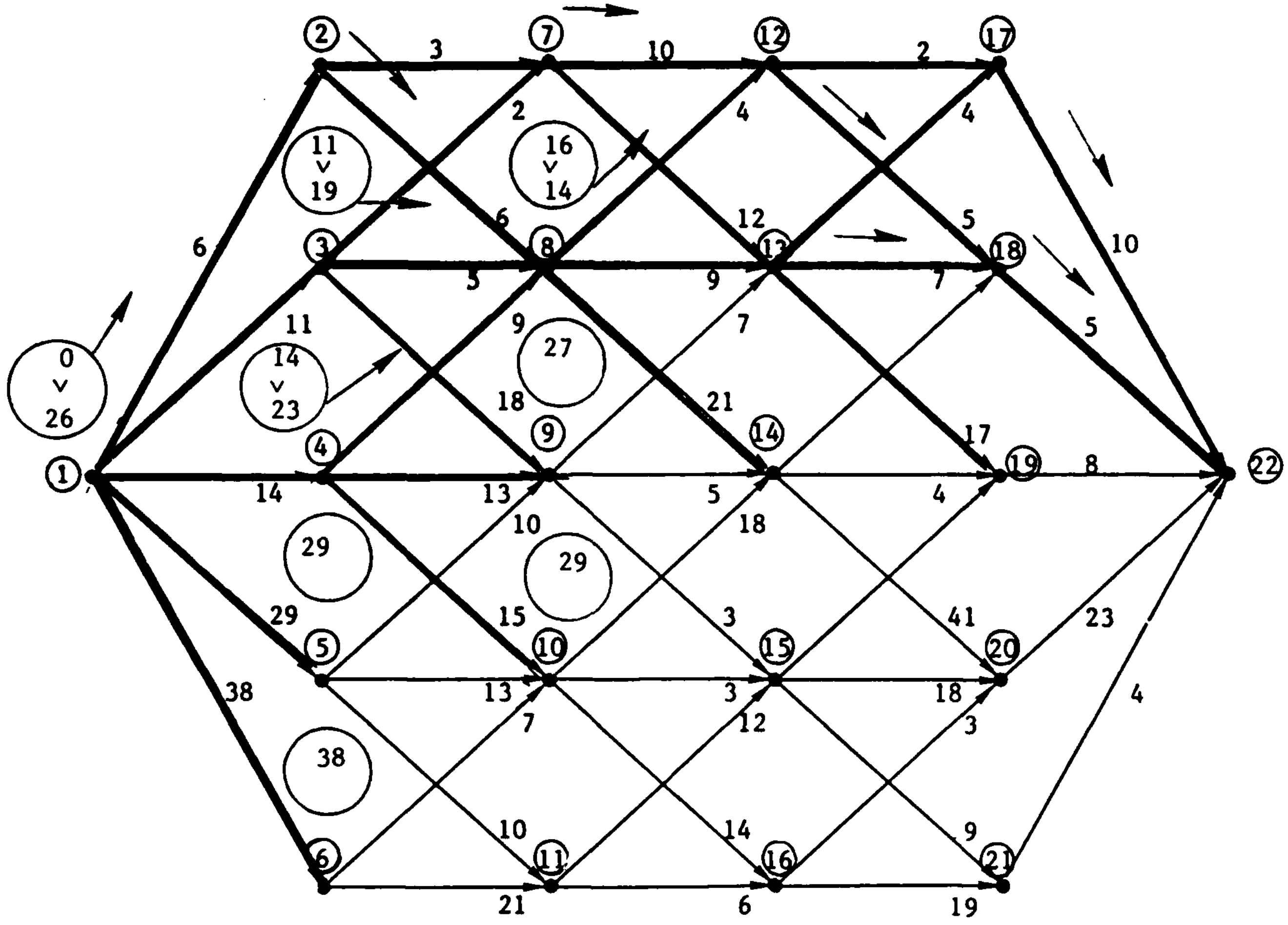

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
0
v
26
11
v
19
14
v
23
16
v
14
27
29
29
38
6
3
10
2
2
4
4
6
12
5
5
9
7
10
11
9
7
5
18
21
17
14
13
5
4
8
10
18
29
15
3
41
23
38
13
3
18
7
12
3
4
10
14
9
21
6
19

Figure B-16

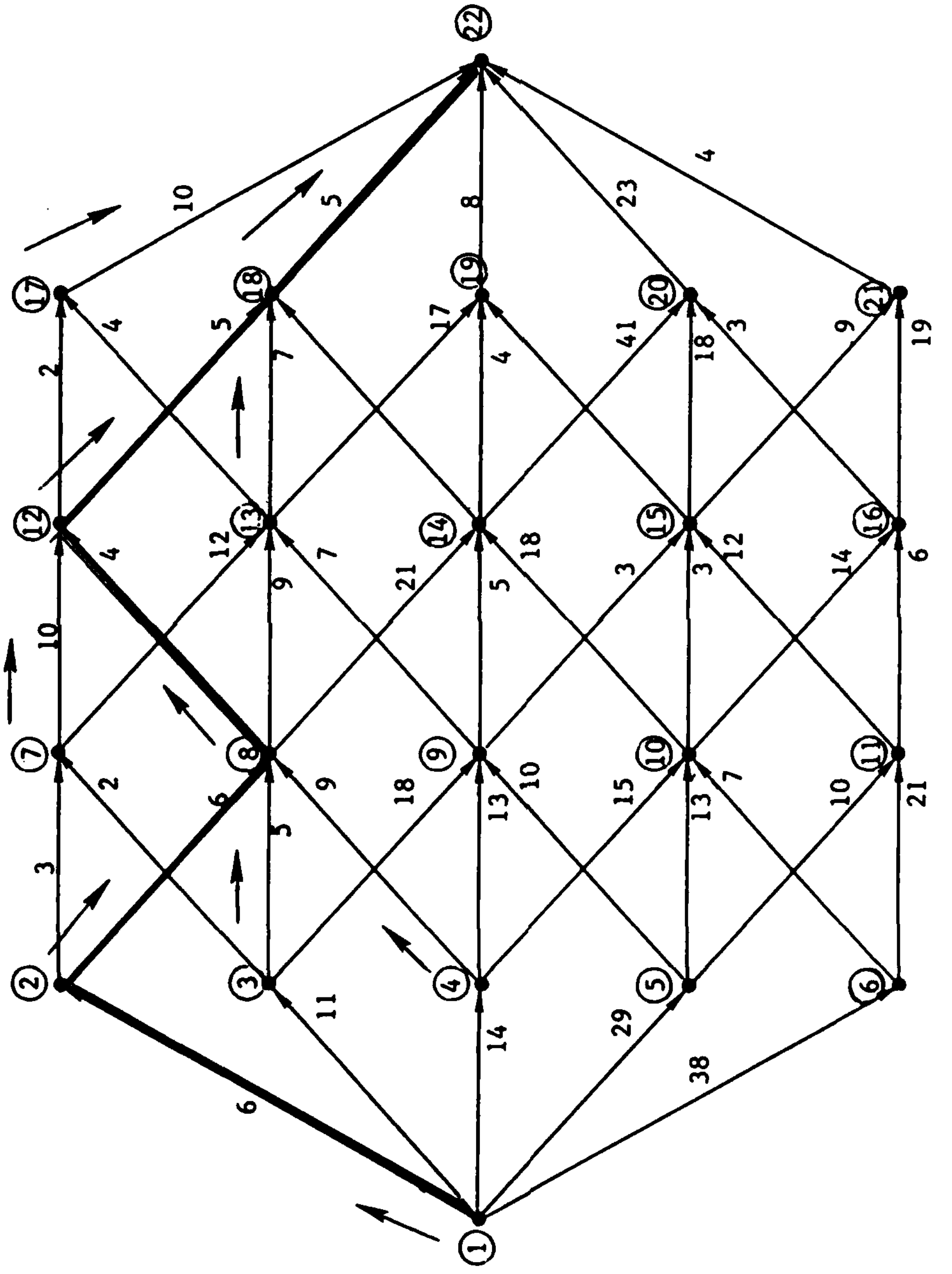

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
6
11
14
29
38
3
5
13
13
21
2
6
9
18
10
15
7
10
10
9
5
3
6
4
12
7
21
18
3
12
14
2
7
4
18
19
4
5
17
41
3
9
10
5
8
23
4

Figure B-17

Appendix C

A DISTRIBUTED RESOURCE ALLOCATION ALGORITHM BASED ON EXTENDED MARGINAL ANALYSIS

1. INTRODUCTION

Marginal analysis is commonly applied to resource allocation problems with separable return functions. However, in many applications, the return function is nonseparable, and thus the classical marginal analysis is not applicable. In this section, we shall extend the marginal analysis to a certain case where the return function is nonseparable. A set of necessary and sufficient conditions for optimal allocation is given. The analysis easily lends itself to distributed implementation.

2. PROBLEM STATEMENT

The allocation problem is given as follows:

$$\left.\begin{aligned} &\max \quad J = \sum_{i=1}^{n} f_i(x_i; y_j, j\ S_i) \\ &\text{s.t.} \sum_{i=1}^{n} x_i + \sum_{j=k}^{s} y_j = R \\ &\qquad\qquad x_i, y_j \varepsilon I_+ = \{0,1,2,\ldots\} \\ &\qquad\qquad S_i \subset \{1,\ldots,s\} \qquad ;i=1,\ldots,n \end{aligned}\right\} \qquad (P)$$

where f_i has the following properties:

(a) $f_i(0; y_j, j\varepsilon S_i) = 0$ for all $y_j \varepsilon I_+$

(b) f_i is nondecreasing in each component while holding the rest of the components fixed

(c) For all $i=1,\ldots.n$ $(x_i > 0, y_t > 0, y_{t'} > 0)$

$$f_i(x_i+1; y_j, j\varepsilon S_i) + f_i(x_i-1; y_j, j\varepsilon S_i) \leqslant 2f_i(x_i; y_j, j\varepsilon S_i)$$

$$f_i(x_i+1; y_t-1, y_j, j\varepsilon S_i - t) + f_i(x_i-1; y_t+1, y_j, j\varepsilon S_i - t) \leqslant 2f_i(x_i; y_j, j\varepsilon S_i); \forall t\varepsilon S_i$$

$$f_i(x_i; y_t+1, y_j, j\varepsilon S_i - t) + f_i(x_i; y_t-1, y_j, j\varepsilon S_i - t) \leqslant 2f_i(x_i; y_j, j\varepsilon S_i) \forall t\varepsilon S_i$$

$$f_i(x_i; y_t+1, y_{t'}-1, j\varepsilon S_i - \{t,t'\}) + f_i(x_i; y_t-1, y_{t'}+1, j\varepsilon S_i - \{t,t'\})$$
$$\leqslant 2f_i(x_i; y_j; j\varepsilon S_i) \forall t,t'\varepsilon S_i$$

Note that this corresponds to "concavity" of f_i.

Note that (P) is more general than the usual resource allocation problem where the return function is separable. One class of problems that has the above formulation is in mission planning where we are to allocate R resource units to n targets which are defended by s defense sites. S_i represents the subset of defense sites that can "protect" target i. The function $f_i(x_i;y_j,j\varepsilon S_i)$ represents the probability of destroying the i^{th} target, and its functional form implies that the success of destroying target i depends on (1) allocation of x_i units to target i and (2) allocation of y_j units to those defense sites which protect target i.

3. A TRADING MODEL

Let us define a concept of price which will be utilized in the later discussions. For a given allocation $\{x_i\},\{y_j\}$, define for $x_i > 0$

$$\lambda_i^-(x_i;y_j,j\varepsilon S_i) \overset{\Delta}{=} f_i(x_i;y_j,j\varepsilon S_i) - f_i(x_i-1;y_j,j\varepsilon S_i)$$

and for $y_j>0$, $G_j \overset{\Delta}{=} \{i|j\varepsilon S_i\}$, $y^j \overset{\Delta}{=} \{y_1,\ldots,y_{j-1},y_{j+1},\ldots,y_s\}$

$$\rho_j^-(y_j,y^j;x_i,i\varepsilon G_j) \overset{\Delta}{=} \sum_{i\varepsilon G_j} \{f_i(x_i;y_\ell,\ell\varepsilon S_i) - f_i(x_i;y_j-1,y_\ell,\ell\varepsilon S_i-j)\}$$

For completeness, we define $\lambda_i^-(0;\cdot) = \infty$, $\rho_j^-(0,y^j;\cdot) = \infty$.

Note that λ_i^- is equal to the marginal decrease (in absolute value) in return if one resource unit is taken away from the i^{th} target from its nominal allocation. Now if we imagine that there are n agents, each of them controlling resource units $\{x_i\}$ assigned to the i^{th} target, then for agent i to give up one resource unit, he must charge a price equal to the marginal decrease in return due to the reduction of one resource unit. Therefore, one can interpret λ_i^- as the "selling price" for one resource unit asked by agent i who controls x_i.

Similarly, if we imagine that there are s agents controlling the resource units $\{y_j\}$ assigned to the j^{th} defense site, then ρ_j^- has the interpretation of "selling price" for one resource unit asked by the agent controlling y_j.

Analogous to the concept of selling price, we have a concept of "buying" price." Define

$$\lambda_i^+(x_i;y_j,j\varepsilon S_i) = f_i(x_i + 1;\ y_j,j\varepsilon S_i) - f_i(x_i;y_j,j\varepsilon S_i)$$

This has the interpretation of the buying price that the agent controlling x_i is willing to <u>pay</u> for an additional resource unit from an outside source. Similarly, one can define

$$\rho_j^+(y_j,y^j;x_i,i\varepsilon G_j) = \sum_{i\varepsilon G_j} \{f_i(x_i;y_j+1,y_\ell,\ell\varepsilon S_i-j) - f_i(x_i;y_\ell,\ell\varepsilon S_i)\}$$

With the above notion of prices, we have easily the following properties:

(1) $\lambda_i^-(x_i;y_j,j\varepsilon S_i) = \lambda_i^+(x_i - 1;\ y_j,j\varepsilon S_i)$

(2) $\lambda_i^-(x_i;y_j,j\varepsilon S_i) \geq \lambda_i^+(x_i;y_j,j\varepsilon S_i)$ [from property (c)]

(3) $\rho_j^-(y_j,y^j;x_i,i\varepsilon G_j) = \rho_j^+(y_j - 1,y^j;x_i,i\varepsilon G_j)$

(4) $\rho_j^-(y_j,y^j;x_i,i\ \varepsilon\ G_j) \geq \rho_j^+(y_j,y^j;x_i,i\varepsilon G_j)$ [from (c)]

Properties (1) and (3) come from our definition and the implication of properties (2) and (4) is that at each agent, the selling price for one unit resource is greater than the buying price for one unit resource.

With the above concept, we can investigate the"trading" pattern among agents, where the result of a trading yields a better overall performance; i.e., J increases. Let us divide the agents into two groups: X-group and Y-group. The X-group controls the resource units allocated to the targets and the Y-group controls the resource units allocated to the defenses. We have the following trading patterns:

(1) agents in X-group trade among themselves

(2) agents in Y-group trade among themselves

(3) agents in X-group trade with agents in Y-group

We can also have hierarchical trading patterns as illustrated in Fig. 1.

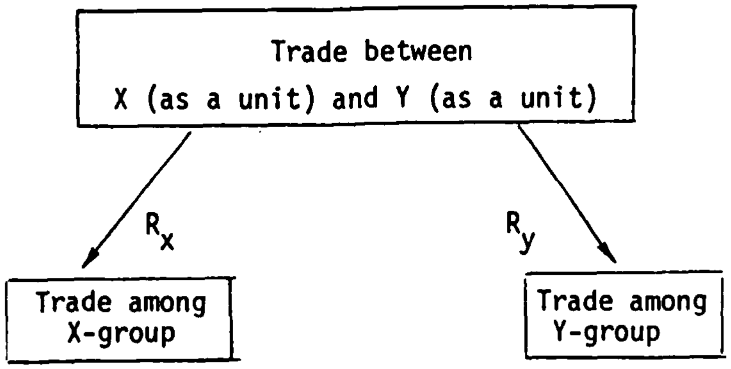


Figure A-1 Hierarchical Trading Pattern

4. TRADING EQUILIBRIUM AND OPTIMAL ALLOCATION

In this section we shall investigate the above trading patterns, and relate the concept of trading equilibrium to optimal allocation.

Theorem 1: Let $\{x_i\}$,$\{y_j\}$ be a given feasible allocation. Assume now trading takes place among X-group with $\{y_j\}$ remaining the same. If there exist some i,i' such that

$$\lambda_i^+(x_i;y_j,j\epsilon S_i) > \lambda_{i'}^-(x_{i'};y_j,j\epsilon S_i), \qquad (1)$$

then trading will take place between agents i and i' where at least one resource unit will be transferred from i' to i.

Proof: Equation (1) implies that

$$f_i(x_i + 1;\ y_j,\ j\varepsilon S_i) - f_i(x_i;y_j,j\varepsilon S_i) > f_{i'}(x_{i'};y_j,j\varepsilon S_{i'}) - f_{i'}(x_{i'}-1;y_j,j\varepsilon S_{i'})$$

and thus for the new allocation $\{x_i^o\},\{y_j\}$ where

$$x_i^o = x_i + 1 \quad ;x_{i'}^o = x_{i'} - 1 \qquad x_t^o = x_t \quad t \neq i,i'$$

we have

$$\sum_{i=1}^{n} f_i(x_i^o;y_j,j\varepsilon S_i) > \sum_{i=1}^{n} f_i(x_i;y_j,j\varepsilon S_i)$$

and thus trading will take place between agents i and i' which results in a better overall performance.

Consider the situation where $\{y_j\}$ is held fixed, then trading among agents in group X continues as long as (1) is satisfied for some i, i'; and every time a unit resource transaction is completed, the overall performance is improved. The trading sequence will result in a final allocation where no more trading will take place. We shall call such an allocation an equilibrium allocation among the X-group while $\{y_j\}$ is held fixed.

<u>Theorem 2</u>: Let $\{y_j\}$ be given with $\sum_{j=1}^{s} y_j < R$. The equilibrium allocation among the X-group, $\{\hat{X}_i\}$, is characterized by

$$\lambda_i^+(\hat{x}_i;y_j,j\varepsilon S_i) \leqslant \lambda_{i'}^-(\hat{x}_{i'};y_j,j\varepsilon S_{i'}) \qquad \begin{array}{l} i = 1,\ldots,n \\ i' = 1,\ldots,n \\ i \neq i' \end{array} \qquad (2)$$

Moreover, $\{x_i\}$ solves the optimization problem

$$\left.\begin{array}{ll} \max & J = \sum_{i-1}^{n} f_i(x_i;y_j,j\varepsilon S_i) \\ & \text{s.t.} \ \sum_{i-1}^{n} x_i = R - \sum_{j-1}^{s} y_j \qquad ; x_i\varepsilon I^+ \end{array}\right\} P(y)$$

Proof: From theorem 1, we see that (2) must be satisfied if no trading occurs. All we need to establish is that if (2) is satisfied, then no trading occurs. Let us consider an allocation $\{x_i^o\},\{y_i\}$ where

$$x_i^o = \hat{x}_i + k_i \quad ; \quad \sum_{i=1} k_i = 0$$

One can imagine $\{x_i^o\}$ as the resulting allocation, as deviated from $\{\hat{x}_i\}$, if a trade has occurred among agents in X-group who are having initial allocation $\{\hat{x}_i\}$. For any i, k_i can either be positive or negative. Suppose $k_i > 0$, then

$$\begin{aligned} f_i(x_i^o;y_j,j\varepsilon S_i) &= f_i(x_i + k_i;\ y_j,j\varepsilon S_i) \\ &= \sum_{\gamma=0}^{k_i-1} \lambda_i^+(x_i+\gamma;y_j,j\varepsilon S_i) + f_i(\hat{x}_i;y_j,j\varepsilon S_i) \qquad (3) \end{aligned}$$

By the first inequality in assumption (c), we have

$$f_i(x_i;y_j,j\varepsilon S_i) > \frac{1}{2} f_i(x_i+1;y_j,j\varepsilon S_i) + \frac{1}{2} f_i(x_i-1;y_j,j\varepsilon S_i)$$

or for all $x_i > 0$

$$\lambda_i^+(x_i-1;y_j,j\varepsilon S_i) \geq \lambda_i^+(x_i;y_j,j\varepsilon S_i)$$

which implies that $\lambda_i^+(x_i;\cdot)$ is decreasing in x_i. Using this fact, (3) becomes

$$f_i(x_i^o;y_j,j\varepsilon S_i) \leq k_i\lambda_i^+(\hat{x}_i;y_j,j\varepsilon S_i) + f_i(\hat{x}_i;y_j,j\varepsilon S_i)$$

If $k_i < 0$, then using the similar argument, we have

$$\begin{aligned} f_i(x_i^o;y_j,j\varepsilon S_i) &= -\sum_{\gamma=0}^{|k_i|-1} \lambda_i^-(\hat{x}_i;\ ;y_j,j\varepsilon S_i) + f_i(\hat{x}_i;y_j,j\varepsilon S_i) \\ &\leq k_i\lambda_i^-(\hat{x}_i;y_j,j\varepsilon S_i) + f_i(\hat{x}_i;y_j,j\varepsilon S_i)\,. \qquad (4) \end{aligned}$$

Let $\Pi = \min_{i=1,\dots n} \lambda_i^-(x_i;y_j,j\varepsilon S_i)$, then (1),(3) and (4) gives

$$f_i(x_i^o;y_j,j\varepsilon S_i) \leqslant k_i \Pi + f_i(\hat{x}_i;y_j,j\varepsilon S_i) \tag{5}$$

since $\sum_{i=1}^{n} k_i = 0$, thus

$$\sum_{i=1}^{n} f_i(x_i^o;y_j,j\varepsilon S_i) \leqslant \sum_{i=1}^{n} f_i(\hat{x}_i;y_j,j\varepsilon S_i) \tag{6}$$

and therefore no incentive for such trading to occur. Note also that (6) implies x_i is an optimal allocation for P(y).

Next, we shall consider trading among agents in Y-group. We shall say that j and j' are <u>related</u> if $G_j \cap G_{j'} \neq 0$; otherwise they are unrelated.

Theorem 3: Let $\{x_i\},\{y_i\}$ be a feasible allocation. Assume trading takes place among agents in Y-group with $\{x_i\}$ remaining the same. If j, j' are related and if $(y^{(j,j')} = \{y_i,\ldots,y_s\}-\{y_j,y_{j'}\})$

$$\rho_j^+(y_j,y_{j'}-1,y^{(j,j')};x_i,i\varepsilon G_j) > \rho_{j'}^-(y_{j'},y^{j'};x_i,i\varepsilon G_j) \tag{7}$$

then trading occurs between j and j' where at least one resource unit is transferred from agent j' to agent j. On the other hand, if j, j' are unrelated, trading occurs by transferring one unit from j' to j if we have

$$\rho_j^+(y_j,y^j;x_i,i\varepsilon G_j) > \rho_{j'}^-(y_{j'},y^{j'};x_i,i\varepsilon G_{j'}) \tag{7'}$$

Proof: The unrelated case is similar to the case of trading in X-group and the proof is similiar to that for Theorem 1. Therefore we shall concentrate only on the related case.

Let us partition the set $I_n \overset{\Delta}{=} \{1,2,\ldots n\}$ into $G_j \cap G_{j'}$, $G_j|G_{j'}$, $G_{j'}|G_j$ and $I_n|G_j \cup G_{j'}$.[†] If j, j' are related, $G_j \cap G_{j'}$ is nonempty. Consider a new allocation $\{y_j^o\}$ with

$$y_j^o = y_j + 1 \quad ;y_{j'}^o = y_{j'} - 1 \quad ;y_t^o = y_t \quad t \neq j,j'$$

For $i\varepsilon G_j \cap G_{j'}$, we have

$$\begin{aligned} f_i(x_i;y_\ell^o,\ell\varepsilon S_i) &= f_i(x_i;y_j+1,y_{j'}-1,\ y_\ell,\ell\varepsilon\ S_i-\{j,j'\}) \\ &= f_i(x_i;y_j+1,\ y_{j'}-1,\ y_\ell,\ell\varepsilon S_i-\{j,j'\}) \\ &\quad - f_i(x_i;y_{j'}-1,y_\ell,\ell\varepsilon S_i-j') + f_i(x;y_{j'}-1,y_\ell,\ell\varepsilon S_i-j') \\ &\quad - f_i(x_i;y_j,j\varepsilon S_i) + f_i(x_i;y_j,j\varepsilon S_i) \end{aligned} \tag{8}$$

for $i\varepsilon G_j|G_{j'}$, we have

$$\begin{aligned} f_i(x_i;y_\ell^o,\ell\varepsilon S_i) &= f_i(x_i;y_j+1,y_\ell,\ell\varepsilon S_i-j) - f(x_i;y_\ell,\ell\varepsilon S_i) \\ &\quad + f_i(x_i;y_\ell,\ell\varepsilon S_i) \end{aligned} \tag{9}$$

for $i\varepsilon G_{j'}|G_j$, we have

$$\begin{aligned} f_i(x_i;y_\ell^o,\ell\varepsilon S_i) &= f_i(x_i;y_{j'}-1,y_\ell,\ell\varepsilon S_i-j')-f_i(x_i;y_\ell,\ell\varepsilon S_i) \\ &\quad + f_i(x_i;y_\ell,\ell\varepsilon\ S_i) \end{aligned} \tag{10}$$

and for $i\varepsilon\ I_n|G_j \cup G_{j'}$, we have

$$f_i(x_i;y_\ell^o,\ell\varepsilon S_i) = f_i(x_i;y_\ell,\ell\varepsilon S_i) \tag{11}$$

[†] $A|B = \{a\varepsilon A | a\notin B\}$

Combining (8)-(11) and using the price definitions, we have

$$\sum_{i=1}^{n} f_i(x_i;y_\ell^o,\ell\varepsilon S_i) = \rho_j^+(y_j,y_{j'}-1,y^{(j,j')};x_i,i\varepsilon G_j)$$
$$- \rho_{j'}^-(y_{j'},y^{j'};x_i\varepsilon G_{j'}) + \sum_{i=1}^{n} f_i(x_i;y_\ell,\ell\varepsilon S_i) \qquad (12)$$

therefore if (7) is true, trading will occur between j' and j which results in a better allocation.

We can define equilibrium allocation among Y-group as we did for X-group.

<u>Theorem 4</u>: Let $\{x_i\}$ be given with $\sum_{i=1}^{n} x_i < R$. The equilibrium allocation among the Y-group, $\{\hat{y}_j\}$, is characterized by

$$\rho_j^+(\hat{y}_j,\hat{y}_{j'}-1,\hat{y}^{(j,j')};x_i,i\varepsilon G_j) \leqslant \rho_{j'}^-(\hat{y}_{j'},\hat{y}^{j'};x_i,i\varepsilon G_{j'}) \qquad (13)$$

if j and j' are related, and

$$\rho_j^+(\hat{y}_j,\hat{y}^j;x_i,i\varepsilon G_j) \leqslant \rho_{j'}^-(\hat{y}_{j'},\hat{y}^{j'};x_i,i\varepsilon G_{j'}) \qquad (13')$$

if j and j' are unrelated. Moreover $\{\hat{y}_j\}$ solves

$$\left.\begin{array}{ll} \max & \sum_{i=1}^{n} f_i(x_i;y_j,j\varepsilon S_i) \\ \text{s.t.} & \sum_{j=1}^{s} y_j = R - \sum_{i=1}^{n} x_i \quad ;y_j\varepsilon I^+ \end{array}\right\} P(x)$$

Proof: The proof is similar to that for Theorem 2 while utilizing (8), (9),(10), (11) and the third and fourth inequalities in assumption (c); but algebrically is more complicated because of the "relationship" among the agents in Y-group.

Finally, we shall consider trading between agents in X and Y groups. We shall say that an agent i in X-group and an agent j in Y-group are dependent if $j\varepsilon S_i$ (or equivalently $i\varepsilon G_j$).

Theorem 5: Let $\{x_i\},\{y_j\}$ be a feasible allocation. If i in X-group and j in Y-group are dependent, and if

$$\rho_j^+(y_j,y^j,x_i-1,x_t,t\varepsilon G_j-i) > \lambda_i^-(x_i;y_\ell,\ell\varepsilon S_i) \tag{14}$$

trading occurs between these two agents with at least one resource unit being transferred from i to j; and if

$$\lambda_i^+(x_i;y_j-1,y_\ell,\ell\varepsilon S_i-j) > \rho_j^-(y_j,y^j;x_t,t\varepsilon G_j) \tag{15}$$

then trading occurs with at least one resource unit being transferred from j to i.

If, however i and j are not dependent, then trading occurs with at least one unit being transferred from i to j if

$$\rho_j^+(y_j,y^j;x_t,t\varepsilon G_j) > \lambda_i^-(x;y_\ell,\ell\varepsilon S_i) \tag{14'}$$

and one unit being transferred from j to i if

$$\lambda_i^+(x_i;y_\ell,\ell\varepsilon S_i) > \rho_j^-(y_j,y^j;x_t,t\varepsilon G_j) \tag{15'}$$

Proof: This proof is similar to those for Theorem 1 and Theorem 3. In the proof, the second inequality in assumption (c) is utilized.

We shall say that a certain allocation $\{x_i^*\},\{y_j^*\}$ is in global equilibrium if there is no incentive for any agents, either in X or Y group, to trade with each other.

Theorem 6: A global equilibrium allocation $\{x_i^*\},\{y_j^*\}$ is characterized by the following conditions

$$
(C^*)\begin{cases}
(1) & \lambda_i^+(x_i^*;y_j^*,j\epsilon S_i) \leqslant \lambda_{i'}^-(x_{i'}^*;y_j^*,j\epsilon S_i);\ \begin{matrix} i'=1,\dots,n \\ i=1,\dots,n \end{matrix} \\[2ex]
(2) & \rho_j^+(y_j^*,y_{j'}^*-1,y^{*(j,j')};x_i^*,i\epsilon Gj) \leqslant \rho_{j'}^-(y_{j'}^*,y^{*j'};x_i^*,i\epsilon G_j) \\
 & \qquad\qquad\qquad\qquad \text{for all } (j,j') \text{ which are related} \\[2ex]
 & \rho_j^+(y_j^*,y^{*j};x_i^*,i\epsilon G_j) \leqslant \rho_{j'}^-(y_{j'}^*,y^{*j'};x_i^*,i\epsilon G_{j'}) \\
 & \qquad\qquad\qquad\qquad \text{for all } (j,j') \text{ which are not related} \\[2ex]
(3) & \rho_j^+(y_j^*,y^{*j};x_i^*-1,x_t^*,t\epsilon G_j-i) \leqslant \lambda_i^-(x_i^*;y_\ell^*,\ell\epsilon S_i);\ \forall i\epsilon G_j,\ j=1,\dots.s \\[2ex]
 & \rho_j^+(y_j^*,y^{*j},x_t^*,t\epsilon G_j) \leqslant \lambda_i^-(x_i^*;y_\ell^*,\ell\epsilon S_i)\quad \forall i\notin G_j,\ j=1,\dots s \\[2ex]
 & \lambda_i^+(x_i^*;y_j^*-1,y_\ell^*,\ell\epsilon S_i-j) \leqslant \rho_j^-(y_j^*,y^{*j};x_t^*,t\epsilon G_j)\quad \forall j\epsilon S_i,\ i=1,\dots.n \\[2ex]
 & \lambda_i^+(x_i^*;\ y_\ell^*,\ell\epsilon S_i) \leqslant \ell_j^-(y_j^*,y^{*j};x_t^*,t\epsilon G_j)\quad \forall j\notin S_i,\ i=1,\dots.n
\end{cases}
$$

Moreover, if $\{x_i^*\},\{y_j^*\}$ is a global equilibrium allocation, it also solves (P).

Proof: This comes directly from combination of Theorem 1 to Theorem 5.

C* is the necessary and sufficient condition for both the global equilibrium allocation for the trading problem and the optimal solution for the allocation problem (P). An interesting interpretation of C* is possible.

Let us interpret

$\rho_j^+(y_j, y_j-1, y^{(j,j')}; x_i \varepsilon G_j)$ -- buying price j will offer for one additional resource from a related agent j' in Y-group

$\rho_j^+(y_j; x_i-1, x_t, x_t \varepsilon G_j - i)$ -- buying price j will offer for one additional resource from a dependent agent i in group X

$\lambda_i^+(x_i; y_j-1, y_\ell, \ell \varepsilon S_i - j)$ -- buying price that agent i in X-group will offer for one additional resource from a dependent agent j in group G

Then each agent has a selling price for one unit of resource; but dependent on where he buys his resource, has different buying prices for one more unit of resource. The global equilibrium is achieved when all the buying prices each agent is willing to give is lower than all the selling prices offered by the agents.

5. A DISTRIBUTED ALGORITHM

In this section, we shall describe a distributed algorithm based on Theorem 6. The algorithm is based on a sequence of "distributed trading" which leads to the trading equilibrium. Each trading cycle consists of two phases:

Phase 1: information exchange

Phase 2: unit resource trading

The purpose of Phase 1 is for each trading agent to compute his selling price and his set of buying prices (from different agents) for additional resources; in Phase 2, trading is to be carried out among agents in a distributed manner such that J is increased.

Phase 1: Information Exchange

Let us assume that, in order to evaluate

$$\{f_i(x_i+\alpha; y_j+\beta, j\varepsilon S_i)\} \quad \alpha = -1,0,1; \quad \beta = -1,0,1$$

information exchange between agents i in X-group and j in Y-group ($j \varepsilon S_i$)

must be carried out; which will, in turn, determine

$$\lambda_i^+(x_i;y_j,j\varepsilon S_i),\ \lambda_i^+(x_i;y_j-1,y_\ell,\ell\varepsilon S_i),\ \forall\, j\varepsilon S_i$$

and

$$\lambda_i^-(x_i;y_j,j\varepsilon Si).$$

After

$$\{f_i(x_i+\alpha;y_j+\beta,j\varepsilon S_i)\}\quad \alpha = -1,0,1;\ \beta=-1,0,1$$

are determined for all i = 1,...n; then

$$\rho_j^+(y_j,y^j;x_i,i\varepsilon G_j),\quad \rho_j^+(y_j,y_{j'}-1,y^{(j,j')};x_i,i\varepsilon G_j)$$

and

$$\rho_j^-(y_j,y^j;x^i,i\varepsilon G_j)$$

are determined for each agent j in Y-group.

Phase 2: Unit Resource Trading

We shall distinguish between three trading patterns:

(a) Trading among X-group

(b) Trading among Y-group

(c) Trading between X and Y groups

To describe how trading is to be carried out, we need to specify the set of agents that each agent is allowed to trade with. For trading pattern (a), each agent in X-group is allowed to trade with any one or more agents in X-group. For trading pattern (b), each agent in Y-group is allowed to trade with all related agents and one or more unrelated agents in Y-group. For trading pattern (c), each agent in X-group is allowed to trade with all its dependent agents and one or more independent agents in Y-group.

From Phase 1, selling and buying (dependent on agent to buy from) prices are computed; then depending on the trading pattern each agent trades with the allowable trading agent for one unit of resource. The trading is to be carried out in the following sequence.

(1) Each agent determines, from the set of allowable trading agents, a subset of agents who offer a buying price higher than the agent's selling price (denote this subset as a list of buyers). An agent with a nonempty list of buyers is denoted as a selling agent.

(2) Each selling agent goes through his list of buyers and offers to the highest "bidder" a unit of resource at his selling price.

(3) A "buyer" who receives multiple offers chooses to receive one unit of resource from the seller who has the lowest selling price.

(4) Those selling agents whose offers are not accepted will form new lists by deleting the buyer who offered the highest price but chose to buy from other selling agents; these will be the new set of selling agents.

(5) If the set of selling agents is empty, Phase 2 is terminated, otherwise go back to (2) and iterate.

With the imposed "concavity conditions", one can show that successively iterating between Phase 1 and 2 will yield a sequence of monotonic improving resource allocations that will converge to the optimal allocation.