AD-A256 205

NAVAL POSTGRADUATE SCHOOL

Monterey, California

OCT 21 1992

THESIS

SCHEDULING UNDERWAY REPLENISHMENT AS A GENERALIZED ORIENTEERING PROBLEM

by

Jeffrey Scott Dunn

June 1992

Thesis Advisor: Robert F. Dell

Approved for public release; distribution is unlimited

92-27424

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

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

11. TITLE (Including Security Classification)
Scheduling Underway Replenishment as a Generalized Orienteering Problem

12 PERSONAL AUTHOR(S)
DUNN, Jeffrey Scott

13 TYPE OF REPORT	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Year, Month, Day)	15. Page Count
Master's thesis		1992, June	44

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

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Dynamic Programming, Underway Replenishment

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

The replenishment of a dispersed battle group requires logistic ships to travel long distances between ships in the battle group. When operational requirements limit the amount of time that can be spent conducting replenishment, decision makers must select which ships to replenish based on the amount of time needed to transit between ships, and the combat value added to the battle group by replenishment. With proper assumptions, this problem is analagous to the Generalized Orienteering Problem. A dynamic programming algorithm is developed using this approach and tested against a set of test problems. The algorithm is capable of scheduling replenishment using both Delivery Boy, or Circuit Rider tactics. The results indicate that the algorithm runs quickly enough to be useful for scheduling underway replensihment in operational situations.

20 DISTRIBUTION/AVAILABILTIY OF ABSTRACT [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC	1a. REPORT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Robert F. Dell	22b. TELEPHONE (Include Area Code) (408)646-2853	22c. OFFICE SYMBOL OR/De

DD Form 1473, JUN 86 Previous editions are obsolete. S/N 0102-LF-014-6603 SECURITY CLASSIFICATION OF THIS PAGE Unclassified

Approved for public release; distribution is unlimited.

Scheduling Underway Replenishment
as a Generalized Orienteering Problem

by

Jeffrey S. Dunn
Lieutenant, United States Navy
B.A., University of Rochester, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS REASEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 1992

Author: _____
Jeffrey S. Dunn

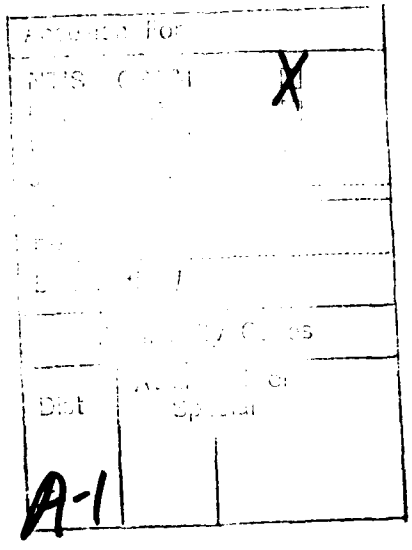
Approved by: _____
Robert F. Dell, Thesis Advisor

_____
Douglas M. Hartman, Second Reader

_____
Peter Purdue, Chairman
Department of Operations Research

ABSTRACT

The replenishment of a dispersed battle group requires logistic ships to travel long distances between ships in the battle group. When operational requirements limit the amount of time that can be spent conducting replenishment, decision makers must select which ships to replenish based on the amount of time needed to transit between ships, and the combat value added to the battle group by replenishment. With proper assumptions, this problem is analagous to the Generalized Orienteering Problem. A dynamic programming algorithm is developed using this approach and tested against a set of test problems. The algorithm is capable of scheduling replenishment using both Delivery Boy, or Circuit Rider tactics. The results indicate that the algorithm runs quickly enough to be useful for scheduling underway replenishment in operational situations.



DTIC QUALITY INSPECTED 1

TABLE OF CONTENTS

I. INTRODUCTION 1

A. BATTLE GROUP OPERATIONS 1

1. Gas Station 1

2. Delivery Boy 2

3. Circuit Rider 2

B. DISCUSSION 2

C. RELATED UNDERWAY REPLENISHMENT STUDIES 4

D. THE GENERALIZED ORIENTEERING PROBLEM 5

E. DYNAMIC PROGRAMMING 5

F. OUTLINE 6

II. PROBLEM FORMULATION 7

A. ASSUMPTIONS 7

B. NETWORK STRUCTURE 7

C. DISCUSSION OF THE GENERALIZED ORIENTEERING PROBLEM 11

D. RECURSIVE RELATIONSHIP BETWEEN STAGES 11

E. DYNAMIC PROGRAMMING FORMULATION 12

III. SOLUTION PROCEDURE 15

A. BREADTH-FIRST SEARCH 15

B. DEPTH-FIRST SEARCH 17

C. ALGORITHM DESCRIPTION 20
D. COLLECTING THE OPTIMAL PATH 22
E. STATE REPRESENTATION AND BEST MATRIX 23
F. THE GENERALIZED PROBLEM 24

IV. TEST AND EVALUATION 26
A. THE EFFECT OF NODE SCORES 27
B. MEASURING THE EFFECTIVENESS OF THE ALGORITHM . 29
1. CPU Requirements 31
2. Performance Based On Labels Generated . . . 31

V. CONCLUSIONS 34
A. ALGORITHM PERFORMANCE 34
B. RECOMMENDATION FOR FUTURE RESEARCH 34
C. SUMMARY 35

LIST OF REFERENCES 36

INITIAL DISTRIBUTION LIST 37

I. INTRODUCTION

A. BATTLE GROUP OPERATIONS

The ability of the United States Navy to project power away from her shores relies heavily on the Carrier Battle Group (CVBG). A CVBG consists of an aircraft carrier, her air-wing and six to nine combatant ships that are tasked with screening the carrier. The carrier and her escorts are required to stay at sea for long periods of time. With the exception of major equipment failure or battle damage, the only limitation to the CVBG's ability to stay at sea is the need for resupply. This need is met through the use of ships assigned to the Combat Logistics Force (CLF). These ships are specifically designed to deliver food, fuel, ordnance, and spare parts to the CVBG from shore support bases.

When the logistic ship arrives in the vicinity of the CVBG, it finds a formation of ships that are assigned to specific sectors for the mutual defense of the battle group. When the logistic ship is ready to resupply the screening ships, she has three primary tactics from which to choose.

1. Gas Station

This method of resupply has the logistic ship stay in a position, normally near the middle of the battle group, and requires the screening ships to leave their assigned sectors

and rendezvous at the position of the logistic ship. When the screening ship has been resupplied, it returns to its assigned sector.

2. Delivery Boy

This tactic requires the logistic ship travel to the screening ships and perform replenishment. There is no requirement for the screening ships to leave their assigned sectors during the replenishment process.

3. Circuit Rider

This tactic can be viewed as a combination of the Delivery Boy and Gas Station tactics. Each screening ship is given a sector where replenishment can take place. If each ship's sector only contains the position of the logistic ship, this tactic reduces to the Gas Station tactic. If each screening ship's replenishment sector only includes it's own on-station position, this tactic reduces to the Delivery Boy tactic. Under normal conditions, both the logistic ship and the screening ship move from their on-station positions and meet for replenishment at a mutually convenient rendezvous location.

B. DISCUSSION

Figure 1 illustrates the three tactic for a four ship formation. Depending on the operational situation, the Gas Station and Delivery Boy tactic have some potential drawbacks. In the Gas Station method, a ship that is assigned to a sector

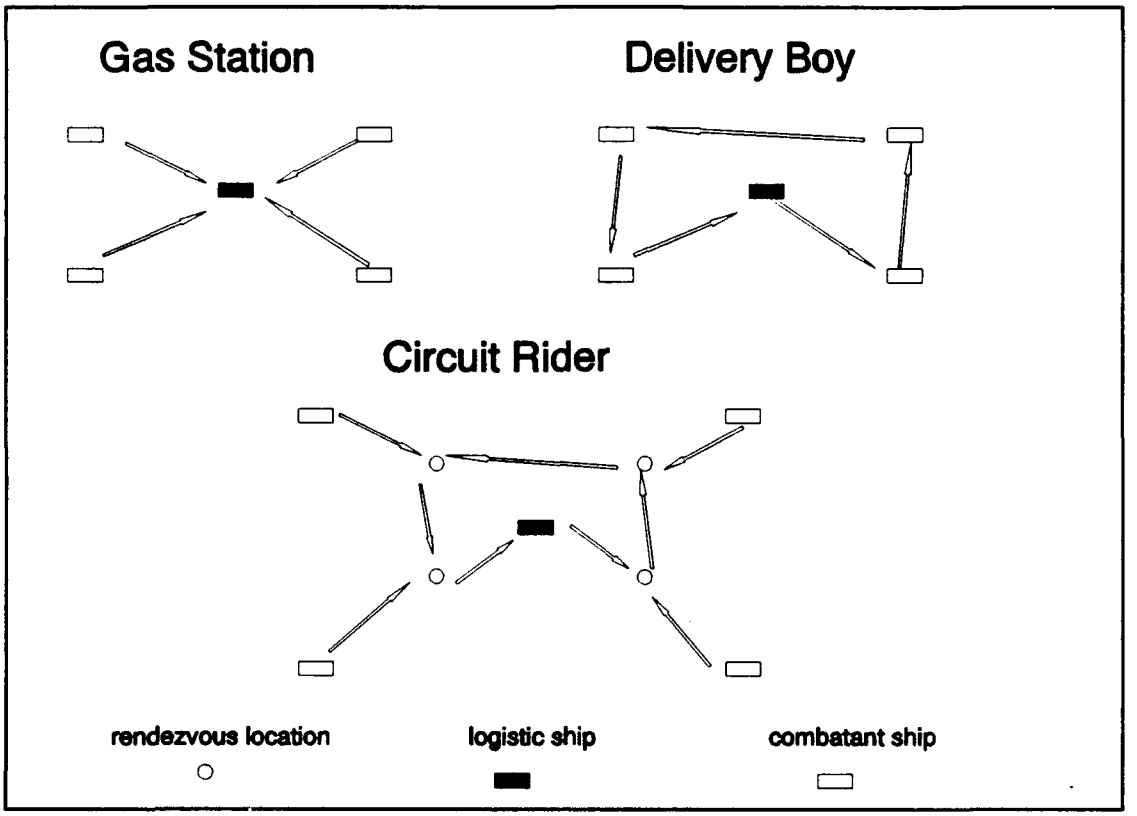


Figure 1 Graphic illustration of underway replenishment tactics

for defensive purposes is taken off station for replenishment. By vacating its sector, the screening ship is removing its defensive ability from the CVBG. In a hostile environment this would be undesirable, particularly if an alternative tactic is available. The Delivery Boy method is likewise deficient in that it may require the relatively defenseless logistic ship to travel alone outside the protected area of the battle group to rendezvous with a single ship, exposing the logistic ship to unnecessary and perhaps unacceptable risks. The Circuit Rider tactic combines these two tactics

and can therefore potentially eliminate some of the drawbacks mentioned above.

C. RELATED UNDERWAY REPLENISHMENT STUDIES

Hardgrave (1989) studied the feasibility of replenishing a CVBG using the tactics discussed above, simplifying the problem by using point locations for each of the screening ships. Zabarouskas (1992) extended the work of Hardgrave by creating a distinct group of locations that represent the sector that a screening ship is assigned, and developed an optimal branch and bound algorithm to determine the minimum time to replenish all ships. Both of these studies formulated the replenishment scheduling problem as a Traveling Salesman Problem (TSP), (see Wu (1992) for a comprehensive review of modelling underway replenishment using the TSP.)

In Wu (1992), models were developed to address scenarios where there is insufficient time to service all the ships needing replenishment. In these scenarios a combat value is assigned to each ship requiring replenishment. The optimal solution results in a schedule that restores the largest combat value to the battle group without exceeding the maximum time allowed for replenishment. The computational study of Wu (1992) demonstrates that solving these problems can require excessive CPU times using commercial integer programming software.

D. THE GENERALIZED ORIENTEERING PROBLEM

Orienteering (Golden, Wang, Liu, 1988) is a sport that requires participants to find their way through a course to a series of 'control points' where they collect a score when they arrive. Participants leave from a designated starting point and must finish at the end point, but are free to make their own paths between the control points. The objective is to collect the highest score while reaching the end point within the prescribed time.

In military situations, time constraints are almost always present and need to be considered. The Orienteering Problem, therefore, more closely resembles the scenario facing the decision maker scheduling underway replenishment.

The problem being addressed in this thesis is the Generalized Orienteering Problem (see Wu (1992)) because each 'control point' or ship sector is represented by a discrete group of points. All points in each sector are assumed to have the same reward, and that reward can only be obtained once.

E. DYNAMIC PROGRAMMING

Dynamic programming, first introduced by Bellman (1957), is an optimization technique in which decisions are made in stages. The technique implicitly enumerates all possible solutions. If the problem can be formulated such that several partial solutions can be grouped in a single state, then those

partial solutions that are inferior can be eliminated from future consideration. The process progresses from stage to stage eliminating dominated partial solutions until only the optimal solution to the problem remains.

F. OUTLINE

This thesis describes a dynamic programming procedure that under specific conditions meets the need reported by Wu (1992) for a specialized algorithm to solve the Generalized Orienteering Problem as applied to underway replenishment scheduling.

Chapter II describes the problem including the necessary assumptions, network structure, and dynamic programming formulation. Chapter III details the solution procedure with reference to data structures, traversal techniques and memory requirements. In Chapter IV, the procedure is used to solve a set of test problems. A description of these problems and the performance of the algorithm are reported. Chapter V concludes with a summary of results and topics for future research.

II. PROBLEM FORMULATION

A. ASSUMPTIONS

With the following assumptions the underway replenishment scheduling problem can be modelled as a Generalized Orienteering Problem.

- The time needed to replenish each ship is constant regardless of the order the ships are serviced. While this is not strictly true, the differential time spent replenishing is insignificant compared to the transit times.
- Ships move direccly from their location to the rendezvous point. In practice there is the need to avoid the other ships in formation.
- The logistics ship starts and ends its path at a point near the center of the formation.
- The battle group maintains its base course and speed throughout the time allotted for replenishment.
- Ships engaged in replenishment maintain their relative position with the battle group by matching its course and speed.
- The optimal path is that path which adds the most combat value to the battle group while allowing time for the logistic ship to return to its position near the center of the formation.

B. NETWORK STRUCTURE

The dynamic programming formulation of the Orienteering Problem as applied to scheduling underway replenishment can be viewed as a directed graph D = { N, A }. Figure 2 illustrates this graph for a scheduling problem where n, the number of

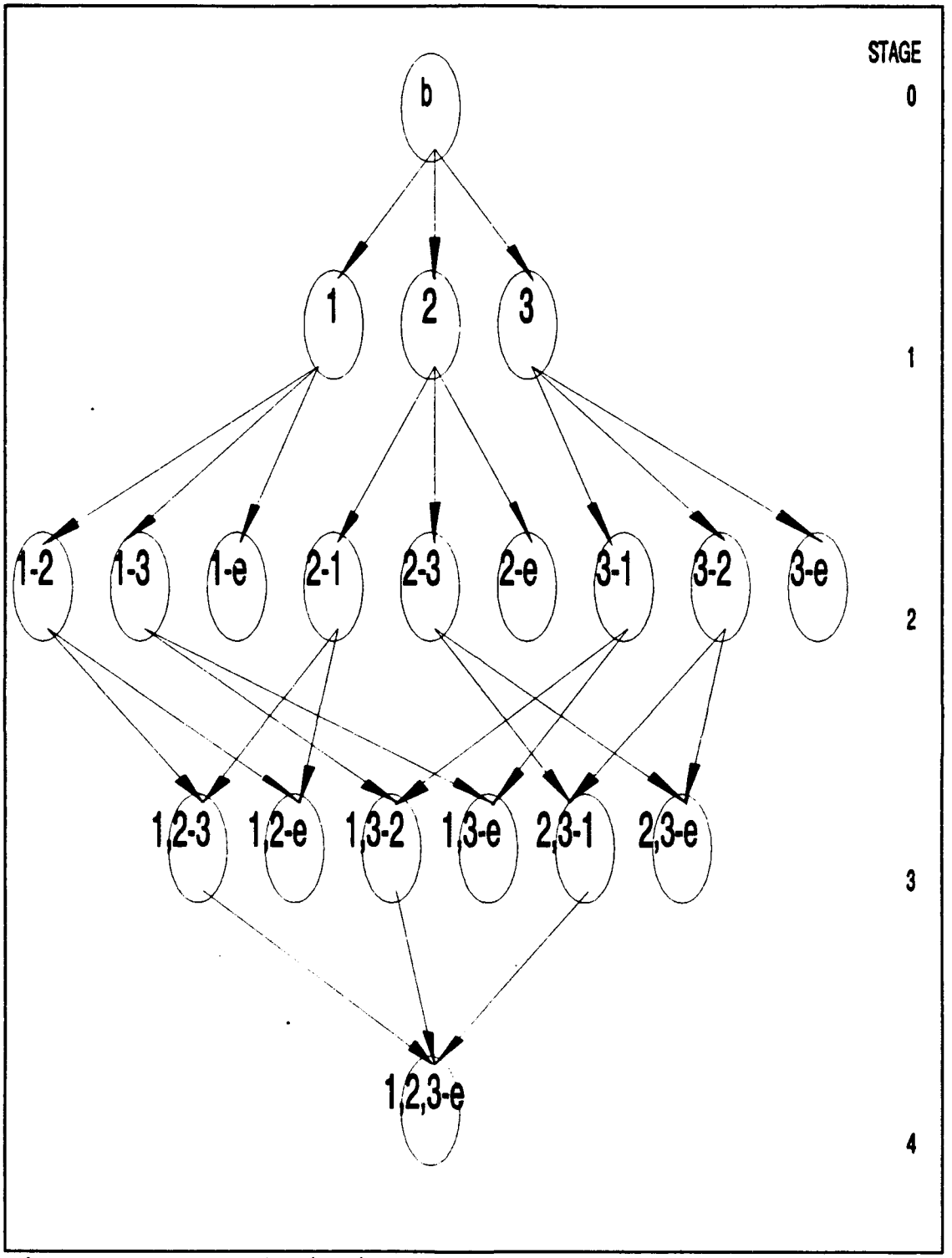


Figure 2 Graph depicting network structure of three node underway replenishment scheduling problem.

ships, is three and each ship is assigned a single rendezvous location. The set of nodes N, contains the possible states of the system. A state is a pair of elements where the first element contains the set of locations that were previously visited, and the second element contains the current location of the logistics ship. For example, the node (1,2-3) in Figure 2 indicates a state that has visited locations 1 and 2 and is currently at location 3.

The set of arcs, A, represent permissible moves from one state to the next, and have associated with them the applicable time of the move. Note that the graph contains no cycles, that is to say there are some locations that cannot be visited from certain states. By excluding arcs that would create cycles, it is guaranteed that a traversal of this graph from **b** to any leaf state yields a feasible solution to the Orienteering Problem, and that the shortest path traversal from **b** to any state produces the optimal path to that state. Also note that any state at stage k>2 can be arrived at through several states in stage k-1. It is this feature of the graph that eliminates the need to explicitly enumerate every permutation of locations to arrive at an optimal solution. At any state it is only necessary to further develop the best path that ended in that state since all others are inferior and cannot be part of an optimal solution.

The graph represented in Figure 2 has one stage for each ship in the formation, plus one additional stage. Stage one

represents the possible partial paths with one ship, stage two the partial paths with two ships and the complete one ship paths, and so on. For example, stage 3 contains the partial 3 ship paths (1,2-3), (1,3-2), (2,3-1), and the completed 2 ship paths (1,2-e), (1,3-e), (2,3-e).

It is possible to group several paths into a single state, note that paths 1 - 2 - 3 and 2 - 1 - 3 are grouped together in the state labeled 1,2 - 3. This is done because for these two paths only the shorter of the two can be part of an optimal tour.

The number of states, N_k, in any stage two through n can be calculated using equation 1.

$$N_k = \binom{n}{k-1} (n-k+2) \qquad (1)$$

The total number of states in the graph can be calculated using equation 2.

$$N_{total} = (n+2)\, 2^{n-1} \qquad (2)$$

The total number of states represents the minimum amount of information needed to guarantee an optimal solution to the problem.

C. DISCUSSION OF THE GENERALIZED ORIENTEERING PROBLEM

The network structure presented above can be easily modified for the Generalized Orienteering Problem where each ship is allowed to have several potential rendezvous locations. In the case of the single location Orienteering Problem, a cycle was defined to be a path that returned to the same location a second time. In the generalized case that principle is extended to prevent the path from returning to the same ship a second time, regardless of the ship's location. The number of states in stages, two through n of the generalized problem is calculated with equation 3.

$$N_k = L\{(\binom{n}{k-1})(n-k+1)\} + \binom{n}{k-1} \quad (3)$$

Where L represents the number of locations for each ship. This equation is further generalized for the case where not all ships have the same number of multiple locations in Chapter IV. Finally, the total number of states in the graph for the generalized problem is computed with equation 4.

$$N_{total} = \{(nL + 2)\}\, 2^{n-1} \quad (4)$$

D. RECURSIVE RELATIONSHIP BETWEEN STAGES

The fact that solutions to problems formulated as dynamic programs are made in stages leads to a search for a relationship between stages. These relationships are referred

to as recursive equations. The existence of a recursive equation highlights the fact that decisions are made in stages and the direction to move in getting from one stage to the next has nothing to do with how the system arrived at the initial stage. This fact can be stated succinctly as Bellman's Principle of Optimality:

> An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. (Bellman 1957)

E. DYNAMIC PROGRAMMING FORMULATION

A mixed integer programming formulation for the Generalized Orienteering Problem is presented in Wu (1992). In what follows, the Orienteering Problem is formulated as a dynamic program. The problem is viewed as having two objective functions. The primary objective ensures that the optimal tour collects the largest score possible. The second objective breaks ties based on the amount of time required to collect the score. For example, given sufficient time numerous tours will collect the total score available; the optimal tour will be that tour which collects the score in the minimum amount of time. This ensures that the solution will reduce to the TSP solution when a large amount of time is provided.

b, e Beginning and ending nodes.

i, j Node indices.

k	Stage index.
S	An unordered set of nodes, that always contains the beginning node b.
(S,i)	A set of nodes that corresponds to a path ending in node i, where i $\notin$ S.
TMAX	Total time allowed to complete a tour.
R(i)	Score received for visiting node i, R(b)=R(e)=0.
t(i,j)	Time required after leaving node i to complete replenishment a node j.
$f_k(S,i)$	Primary objective function; score received by visiting the nodes in set S, where, $\|S\| = k$.
$g_k(S,i)$	Secondary objective function value; minimum time required to travel from b to i via all nodes in set S. Note that $g_k(S,i)$ is only defined when $\|S\| = k$ and $i \notin S$.
f^*	The maximum score obtainable.
g^*	The minimum time to obtain the maximum possible score.

The recursive equations that link stages together are:

$$g_k(S,j) = \min_{i \in S} \{g_{k-1}(S \setminus \{i\}, i) + t(i,j)\}, \quad \forall\, |S| = k-1,\ j \notin S$$

$$f_k(S,j) = \begin{cases} \sum_{i \in S \cup \{j\}} R(i) & \text{if } g_k(S,j) \le TMAX \\ 0 & \text{otherwise} \end{cases}$$

This is clearly the case since the score received for visiting the same nodes (set S), regardless of order, is constant.

The initial conditions for the system are:

$$f_0(\emptyset, b) = 0$$

$$g_0(\emptyset, b) = 0$$

The optimal primary objective function value would be $f^* = \max \{ f_k (S, e)\}$, over all possible k and S where $| S | = k$
The optimal secondary objective function value would be $g^* = \min \{g_k(S, e)\}$ over all possible k and S where $| S | = k$ and $f_k(S,e) = f^*$

This formulation is easily modified for the Generalized Orienteering Problem. The only conceptual difference is that each node now represents both a ship and location of replenishing the ship. This dynamic programming formulation serves as the basis for the solution procedure contained in Chapter III.

III. SOLUTION PROCEDURE

An optimal solution to the Generalized Orienteering Problem provides the minimum time, within an established limit, to collect the maximum possible score. This is equivalent to finding the shortest path to all states in the decision tree shown in Figure 2, and selecting the best path based on the total score collected within the established time. Evaluating the states in the tree requires a systematic search which can be accomplished either in a breadth-first or depth-first manner.

A. BREADTH-FIRST SEARCH

Figure 3 illustrates the breadth-first search of the decision tree. State numbering indicates the order the states are evaluated. This is the technique typically used for solving dynamic programming problems. The procedure evaluates all the states in stage one before moving on to stage two and so on. The primary advantage of this procedure is that at every stage the state values are known to be optimal, and sub-optimal paths are never investigated beyond the point where their inferiority is discovered.

The price paid for this advantage is the space and work required to keep track of all the state information at stage k-1 while computing the node values in stage k. For instance,

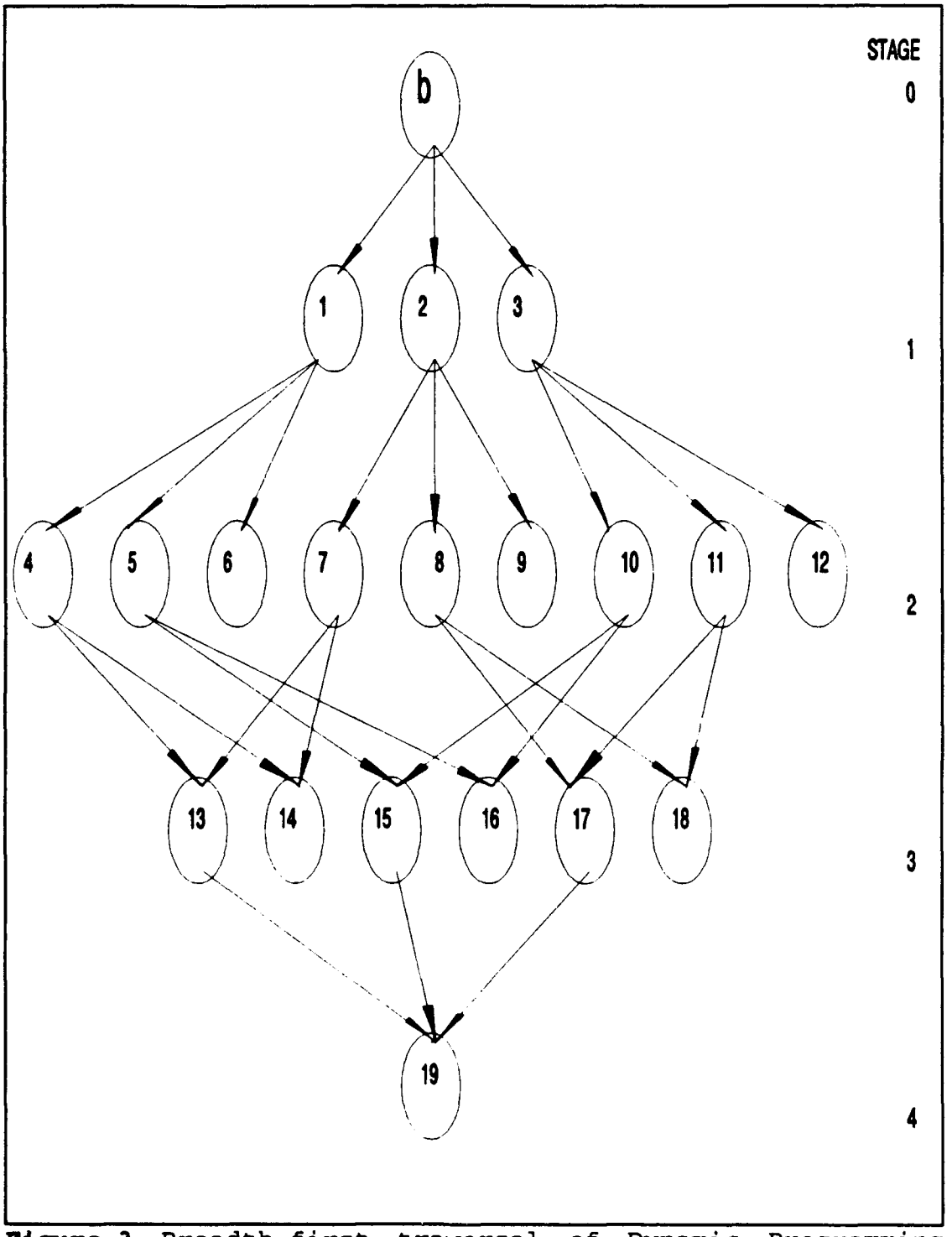


Figure 3 Breadth-first traversal of Dynamic Programming decision tree.

a problem with ten ships that each have 16 possible rendezvous locations has over 24,000 states in stage 5 that must be stored and managed in order to produce the over 17,000 states in stage 6. This served as the motivation for conducting a depth-first search.

B. DEPTH-FIRST SEARCH

The depth-first search moves quickly down the graph, always looking for the shortest feasible arc. If the system moves into a feasible state, a label is created recording the required time and collected score. When the system reaches the end node or an infeasible state, the system moves back up the decision tree and investigates the arcs from the labeled state furthest down the tree, breaking ties based on time required. As the procedure continues, the system can arrive back at a state that has already been labeled. If the new value at that state is inferior to the old value, the path can be discarded since the completion of that path cannot be optimal. This procedure continues until every feasible state in the decision tree is labeled.

Figure 4 is included to help demonstrate this process where the state numbers indicate the order of labelling. The algorithm determines order based on required time, but in this example the state to the left is arbitrarily chosen when a choice is made. Note that states in stages 3 and 4 are labelled more than once. While this can happen, it should not

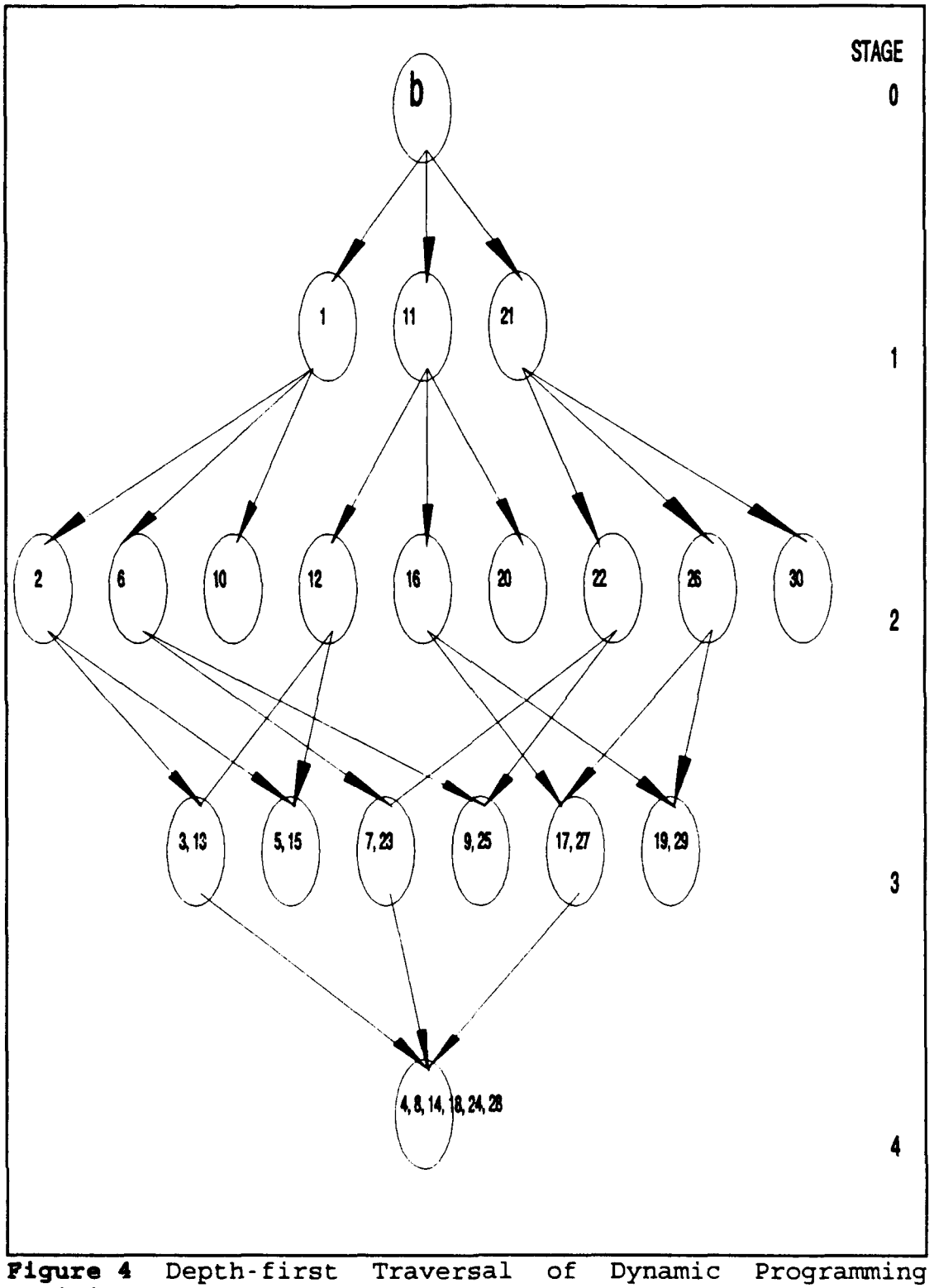


Figure 4 Depth-first Traversal of Dynamic Programming Decision Tree.

happen often since the first label created at any state will have the same time label as the nearest-neighbor heuristic solution to that state. To the extent that the nearest-neighbor heuristic provides 'good' paths, the first label created at any state can be expected to be superior to future labels that are generated.

Dynamic programming procedures are often ineffective for solving large combinatorial problems, like the Generalized Orienteering Problem, because of the enormous amount of state information that must be simultaneously managed to derive a solution. The depth-first traversal of the decision tree greatly reduces the amount of information being managed. For the ten ship example described above, no more than 880 labels must be managed at any time. The minimum time required to each state must still be stored, but this storage requires no processing. Again there is a trade-off, the convenience of reducing the amount of information being processed is paid for by some amount of repeated work. The fact that the nearest neighbor heuristic provides 'good' solutions to shortest path problems indicates that the additional work may justify the trade-off. The depth-first search also has the advantage that premature termination of the procedure will produce feasible solutions.

C. ALGORITHM DESCRIPTION

The following Depth-First Algorithm is described for solving problems where each ship is allowed a single rendezvous location, though it can be easily generalized to the multiple location scenario. Let t(i,j) be the time needed to transit from node i to node j for i = 1, 2, ..., n and j = 1, 2, ..., n where n is the total number of nodes. Let t(i,i) = ∞ for i = 1, 2, ..., n. A tour is a route that begins at the start node and arrives at the end node through some or all of the possible nodes. A path is a route that begins at the start node and ends at a node other than the designated end node. Let R(i) be the value added when the system arrives at state i. The Depth-First Algorithm requires the use of a list of temporary labels to store path data that have not yet been investigated, and a BEST matrix that records the required time of the best known path to each state.

The algorithm can be summarized with the following 7 steps.

Step 1

Set the incumbent objective function value to a large time and low combat value: OFV = $(10^{10},0)$

Step 2

Select the beginning node as the current node.

Step 3

Investigate all nodes adjacent to the current node and record on the temporary list: the total time to complete the path, total combat value, all nodes in the path where the last node and its predecessor are distinguished. This information is only recorded if extending the partial path does not:

- create a cycle.
- exceed the maximum time allowed for the path.
- create a path known to be inferior.

Step 4

For the states where a temporary label was made, update the BEST matrix with the time required to reach each state. It is unnecessary to record the value at the state since every path to a state contains the same nodes and must have the same value.

Step 5

Search the temporary list for the label that is lexicographically smallest on the list based on:

- Most arcs in the partial path.
- Least time to reach the current state.
- Highest combat value added.

Record the information from that label and remove the label from the temporary list. If the list is empty the problem is solved.

Step 6

If the label removed from the temporary list is from a state that represents a completed path, compare that tour to the incumbent solution. The new tour is superior if its value is more than the value of the incumbent. If the value is the same, the tour that requires less time to complete is superior.

Step 7

Make the node selected in Step 5 the current node and return to Step 3.

D. COLLECTING THE OPTIMAL PATH

The depth-first traversal of the decision tree greatly simplifies the collection of the optimal path. By using an assignment array, every time a new label is selected the assignment array is cleared from the new nodes predecessor forward to the end of the last path. When a tour is found that is superior to the best tour so far, the assignment array is stored as the best tour and the process continues. This can be done using the depth-first traversal since the path prior to the stage where the decision is made remains constant. This characteristic is generally not true with the breadth-first traversal, and would necessitate the use of a more elaborate path collection scheme.

E. STATE REPRESENTATION AND BEST MATRIX

The BEST matrix keeps track of the shortest path to all possible states. Because the state is defined as a pair, the first element being the set of all nodes included in the path so far, and the second element being the last node visited, the BEST matrix is two dimensional. The set of all nodes in the path can be represented by a binomial expansion of the set into a single integer. That is, the set {1, 2, 5, 9} can be represented as $2^0+2^1+2^4+2^8$ or 275. The time required to complete the best path that visits nodes 1, 2, 5, and, 9 and ends at node 2 can be found as element BEST (275,2).

This representation of the state has the additional advantage of providing a useful method of checking whether visiting a node will introduce a cycle in the path. A 'hashing function' $\oplus$ is defined to compare the set of nodes with the candidate node, and determine if the addition of the candidate node will introduce a cycle. For example, consider the addition of a node corresponding to ship 5 to a path containing ships {1, 2, 5, 9},

```
{1, 2, 5, 9} =>  275              100010011
       {5}   =>   16        and  000010000
        275 ⊕ 5                  000010000    =>   16
```

A one contained in both number 5 locations indicate a common element. This function can be accomplished with a single operation in many computer languages. In FORTRAN 77 the

'integer and' operation will compare bit strings of two integers and return an integer indicating which bits match.

F. THE GENERALIZED PROBLEM

The previously described algorithm only needs to be slightly altered to solve the Generalized Orienteering Problem. Because many nodes can represent a single ship additional information is required to determine which ships have been visited. A state changes to a pair where the first element is the set of ships that are part of the current path, and the second element is the node (ship and location) where the path ends. Figure 5 illustrates two paths, A and B, that visit the same ships. Ship numbers are represented by Roman numerals. Node numbers are represented with Arabic numerals.

Note that the paths have only their beginning and end nodes in common, but the longer path would be dominated by the shorter path because they visit the same ships, end at the same node and thus obtain the same score.

Note that the total number of states grows exponentially by number of sectors and geometrically by number of nodes. This characteristic is very attractive for Underway Replenishment Scheduling problems where the number of ships is likely to be small but better answers are achieved by allowing numerous locations in each sector.

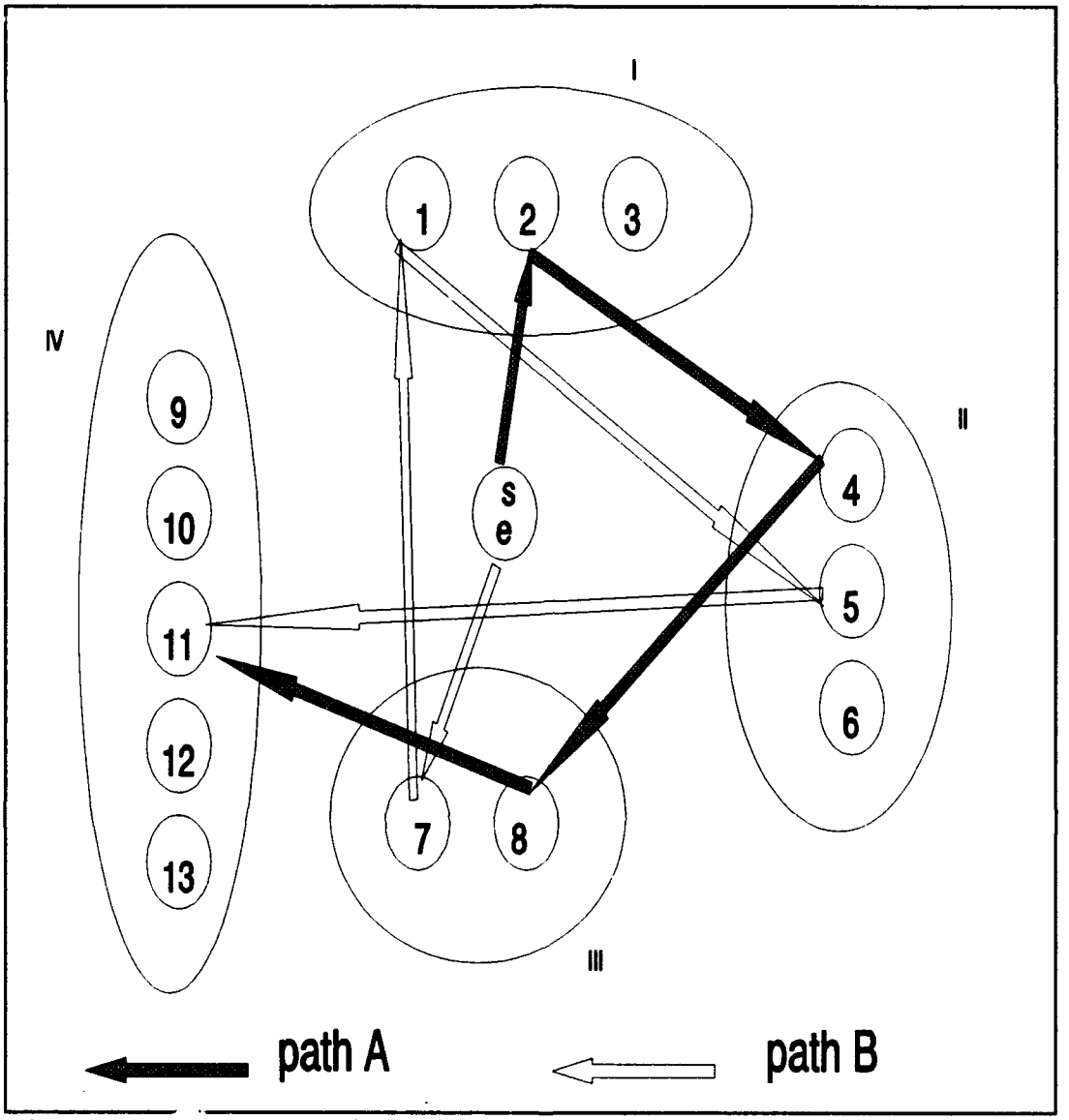


Figure 5 Illustration of dominated path in Generalized Orienteering Problem.

IV. TEST AND EVALUATION

The Depth-First Algorithm was programmed in FORTRAN 77 with computational results collected from a SUN/SPARC2 workstation. To evaluate the algorithm performance in scenarios likely to be faced by decision makers scheduling underway replenishment, it was tested against a set of 630 randomly generated formations. The 630 formations were grouped into 21 classes of 30 formations each. The classes were divided according to the number of ships, the number of ships with multiple locations, and the number of locations those ships had.

The cost matrix (time to transit between nodes) is asymmetric due to the movement of the formation. The data for each problem was processed to eliminate locations that could not be part of the solution. Table I shows in more detail the characteristics of the problem classes. For a complete description of how the test problems were generated see Zabarouskas (1992).

These test problems were derived for both the Delivery Boy and Circuit Rider tactics. For the Delivery Boy tactic, multiple ship locations represent an on-station sector, and the time between locations represents the replenishment ships transit time and the subsequent replenishment of the combatant ship. To be valid problems for the Circuit Rider tactic, the

time between locations represents the time needed for both combatant and replenishment ships to rendezvous and complete replenishment.

A. THE EFFECT OF NODE SCORES

It was decided to test the algorithm's performance for varied node scores. Each of the 630 problems were run with scores randomly generated between one and 100, and then again with scores randomly generated between 100 and 200. It was hypothesized that there might be some advantage to solving the problems with the one to 100 range because of the possibility of a node being worth many times that of another node (for instance, a node with a score of 80 would be twenty times more valuable than a node with a score of four.) This relationship between node scores would not be possible in the 100 - 200 range since, at best, one node could be worth only twice the score of another.

The results of the two test sets showed that the same number of labels were created and thus the same amount of CPU time was used to solve each set of problems. The paths were different when the maximum time allowed was insufficient to visit every node, but the same amount of work was required to generate the optimal solution. The reason for this is that the algorithm seeks the best path to each state based only on the time required to reach that state. This is true because every path to a state has exactly the same value.

Table I CHARACTERISTICS OF TEST PROBLEM SETS.

class	# of ships	# of ships with multiple locations	# of multiple locations	total # of locations	avg. # of locations removed
1	6	0	na	6	na
2	6	3	4	15	3.07
3	6	5	4	21	3.63
4	6	3	9	30	5.37
5	6	5	9	46	8.40
6	6	3	16	51	8.07
7	6	5	16	81	10.73
8	8	0	na	8	na
9	8	4	4	20	1.57
10	8	7	4	29	2.47
11	8	4	9	40	2.63
12	8	7	9	64	4.17
13	8	4	16	68	2.80
14	8	7	16	113	5.23
15	10	0	na	10	na
16	10	5	4	25	1.00
17	10	9	4	37	1.50
18	10	5	9	40	1.77
19	10	9	9	82	1.30
20	10	5	16	85	2.07
21	10	9	16	145	2.77

This result was considered positive in that the performance of the algorithm would not vary depending on the scoring scale used by the decision maker. On the other hand, it highlighted the fact that the algorithm does not take advantage of the information contained in the score when solving the problem.

B. MEASURING THE EFFECTIVENESS OF THE ALGORITHM

To thoroughly determine the effectiveness of the algorithm, it was necessary to test it not just with a random set of formations but also with a variety of time constraints. With this in mind each of the 630 problems were solved four times. The first time, the maximum time allowed for a tour was large enough to ensure that a tour was found that visited all the ships. This tour is equivalent to the solution of the TSP problem for that formation. The TSP time served as a baseline used to generate the other three problems for the formation. The maximum time for these problems were set at 100%, 75%, and 50% of the time required to complete the TSP tour. Problems with a maximum time greater than 100 % of the TSP time were not considered since the algorithm reduces TMAX whenever a solution that visits every sector is found. This adjustment results in times that vary insignificantly from the results reported for 100% of the TSP time.

Table II ALGORITHM RUN-TIME IN CPU SECONDS.

	100% of TSP time		75% of TSP time		50% of TSP time	
class	mean	st dev	mean	st dev	mean	st dev
1	0.029	0.005	0.018	0.005	0.011	0.004
2	0.079	0.021	0.045	0.015	0.022	0.008
3	0.143	0.042	0.083	0.026	0.034	0.015
4	0.219	0.085	0.132	0.054	0.046	0.020
5	0.531	0.224	0.286	0.120	0.094	0.038
6	0.463	0.192	0.272	0.129	0.106	0.058
7	1.673	0.632	0.964	0.394	0.306	0.130
8	0.230	0.024	0.130	0.018	0.041	0.013
9	0.993	0.247	0.540	0.164	0.169	0.084
10	2.099	0.503	1.081	0.299	0.273	0.127
11	2.731	0.810	1.520	0.483	0.478	0.198
12	8.884	2.851	4.755	1.537	1.203	0.492
13	7.101	1.875	3.714	1.152	1.105	0.533
14	26.262	6.915	13.422	3.847	3.354	1.394
15	1.907	0.214	1.047	0.243	0.283	0.101
16	8.900	1.594	4.555	1.126	1.084	0.472
17	21.983	4.599	11.217	3.324	2.422	1.186
18	30.032	6.240	14.303	3.518	3.142	1.394
19	101.975	21.103	49.882	11.180	10.241	4.021
20	75.013	18.324	36.885	10.060	7.215	3.139
21	280.663	66.892	133.087	35.941	24.935	12.352

1. CPU Requirements

The CPU time required to solve each of these problems was recorded. Table II contains the means and standard deviations computed for each class. Note that in each class the reduction in CPU time is greater than the percentage change in TMAX that caused it. The reduction increases as the number of ships and the number of locations increase. Additionally, the table shows a relatively stable record of performance across all classes of problems, as evidenced by standard deviations that are generally 20 to 30 percent of the mean CPU time.

2. Performance Based On Labels Generated

The second measure of the algorithm's effectiveness relates to the amount of work that is actually being done in order to solve the problem. The algorithm creates a label every time a path to a state is found for the first time, and any time a path is found that is superior to the label already created for that state. By comparing the number of times a label is made and the number of states in the system, the performance when the maximum amount of time to complete a tour varies can be made.

The removal of dominated locations, and the fact that not all ships have multiple locations, changes the way the total number of states in the system is calculated. Equation 4 from Chapter II becomes equation 5.

$$N_{total} = (((n+2)+m(L-1))-r)\ 2^{n-1} \quad (5)$$

Where m is the number of ships with multiple locations and r is the number of locations removed by dominance. This formula is for problems where there is sufficient time to visit every ship. In scenarios where the maximum time allowed constrains the solution, the number of feasible states is a function of the formation and cannot be computed deterministicly. In these cases the number of feasible states is kept track of in the program.

Table III shows the average number of states for each class and the average number of labels that were created before the shortest path to each state was found. The table shows a reduction in the number of states similar to the reduction that occurred in CPU requirements. The table also shows that an increase in the number of locations increases the number of times labels are written over. The overwrites are an estimate of the extra work needed to use the depth-first traversal of the decision tree. This extra work grows at a distressing pace for the larger problems, but it is these problems that would require the breadth-first traversal to manage a large number of states. Additionally, the CPU requirements for these problems remains modest despite the additional work being done.

Table III WORK REQUIRED TO OBTAIN SOLUTIONS

	100% of TSP time		75% of TSP time		50% of TSP time	
class	avg total # of states	avg # of labels /state	avg total # of states	avg # of labels /state	avg total # of states	avg total# of labels
1	255.0	1.79	174.3	1.45	90.9	1.25
2	444.9	2.55	313.7	2.00	153.7	1.53
3	617.7	3.34	429.6	2.47	204.1	1.88
4	851.3	3.44	607.9	2.61	284.6	1.81
5	1266.2	5.27	885.9	3.55	427.9	2.26
6	1436.9	3.89	972.4	2.99	483.0	2.13
7	2311.5	7.45	1692.4	5.26	820.9	3.24
8	1279.0	2.28	840.2	1.74	307.2	1.36
9	2614.5	4.27	1769.6	3.06	725.9	2.22
10	3651.3	6.12	2479.3	4.15	919.5	2.67
11	5037.9	5.41	3377.1	4.03	1433.3	2.88
12	7913.7	10.15	5591.7	6.86	2295.6	4.06
13	8600.6	7.28	5945.2	4.93	2570.2	3.27
14	14049.1	13.87	9800.8	8.94	3914.6	5.31
15	6143.0	3.01	4063.5	2.28	1350.0	1.72
16	13311.0	5.73	8729.7	4.10	2889.4	2.82
17	19181.9	9.38	13099.6	6.35	4314.1	4.00
18	25718.5	9.04	16759.9	6.01	5627.8	3.78
19	42341.4	16.78	28395.5	10.98	9774.0	6.28
20	43484.9	11.75	25495.8	7.74	9454.7	4.35
21	73846.4	22.37	49202.4	13.91	15163.2	7.95

V. CONCLUSIONS

A. ALGORITHM PERFORMANCE

The computational results of the algorithm showed excellent performance over a wide range of scenarios. These results, however, are probably more a function of the improved network structure than the traversal technique chosen. The depth-first traversal created many more labels in the multiple location scenarios than would have been created by a breadth-first traversal. But despite this, the algorithm provided in this thesis is an excellent solver for the Delivery Boy, and Circuit Rider tactics, with or without a time constraint.

B. RECOMMENDATION FOR FUTURE RESEARCH

Clearly, there is reason to think an algorithm that conducts a breadth-first search of the described decision tree may improve on these results. Furthermore, any additional means of removing nodes or arcs prior to the implementation of the algorithm is likely to improve performance.

Future research should also include the relaxation of the assumption requiring the formation to maintain course and speed during the entire replenishment cycle, and the assumption that the replenishing ships will be able to match the course and speed of the battle group.

C. SUMMARY

This thesis was undertaken to develop a specialized solution procedure to schedule underway replenishment without the use of commercial integer programming software. The reported algorithm selects the optimal set of ships to replenish, establishes the correct order, and finds the optimal rendezvous locations to conduct the replenishment. The algorithm was tested against a set of randomly generated formations, and was determined to provide optimal solutions quickly enough to meet the needs of underway replenishment schedulers.

LIST OF REFERENCES

Bellman, R, *Dynamic Programming,* Princeton University Press, 1957.

Golden, B., Wang, Q., Liu, L., "A Multifaceted Heuristic for the Orienteering Problem", *Naval Research Logistics,* v.35, 1988.

Hardgrave, S.W., *Determining the Feasibility of Replenishing a Dispersed Carrier Battle Group,* Master's Thesis, Naval Postgraduate School, Monterey, California, September 1989.

Wu, T., *Optimization Models for Underway Replenishment of Dispersed Carrier Battle Groups,* Master's Thesis, Naval Postgraduate School, Monterey, California, March 1992.

Zabarouskas, M.W., *Scheduling Underway Replenishment Using the Delivery Boy and Circuit Rider Tactics, an Asymmetric Generalized TSP,* Master's Thesis, Naval Postgraduate School, Monterey, California, March 1992.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Professor Robert F. Dell Department of Operations Research Naval Postgraduate School, Code OR/De Monterey, California 93943-5000	1
4.	Professor Siriphong Lawphongpanich Department of Operations Research Naval Postgraduate School, Code OR/Lp Monterey, California 93943-5000	1
5.	Professor David Schrady Department of Operations Research Naval Postgraduate School, Code OR/So Monterey, California 93943-5000	1
6.	CDR Douglas Hartman Department of Operations Research Naval Postgraduate School, Code OR/Ht Monterey, California 93943-5000	1
7.	Deputy Chief of Naval Operations (Logistics) OP-402D Washington, District of Columbia 20350	1
8.	Office of the Chief of Naval Operations OP-814 Washington, District of Columbia 20350	1
9.	Defense Logistics Studies Information Exchange US Army Logistics Management College Fort Lee, Virginia 23801-6043	1
10.	LT Jeffrey S. Dunn Rt. 1, Box 131-L Parsons, Tennessee 38363	2