Image Cover Sheet

CLASSIFICATION	SYSTEM NUMBER
UNCLASSIFIED	509725

TITLE

A DYNAMIC PROGRAMMING CONCEPT FOR STRATEGIC PLANNING

System Number:

Patron Number:

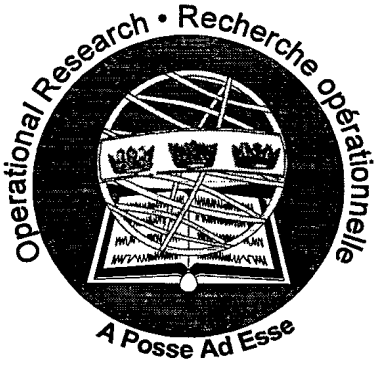
Requester:

Notes:

DSIS Use only:

Deliver to:

DEPARTMENT OF NATIONAL DEFENCE
CANADA



OPERATIONAL RESEARCH DIVISION

DIRECTORATE OF OPERATIONAL RESEARCH (JOINT & LAND)

DOR(J&L) RESEARCH NOTE RN 9819

A DYNAMIC PROGRAMMING CONCEPT FOR STRATEGIC PLANNING

BY

Mr. S. Isbrandt

NOVEMBER 1998

OTTAWA, CANADA

National Defence Défense nationale

OPERATIONAL RESEARCH DIVISION

CATEGORIES OF PUBLICATION

ORD Reports are the most authoritative and most carefully considered publications of the DGOR scientific community. They normally embody the results of major research activities or are significant works of lasting value or provide a comprehensive view on major defence research initiatives. ORD Reports are approved personally by DGOR, and are subject to peer review.

ORD Project Reports record the analysis and results of studies conducted for specific sponsors. This Category is the main vehicle to report completed research to the sponsors and may also describe a significant milestone in ongoing work. They are approved by DGOR and are subject to peer review. They are released initially to sponsors and may, with sponsor approval, be released to other agencies having an interest in the material.

Directorate Research Notes are issued by directorates. They are intended to outline, develop or document proposals, ideas, analysis or models which do not warrant more formal publication. They may record development work done in support of sponsored projects which could be applied elsewhere in the future. As such they help serve as the corporate scientific memory of the directorates.

ORD Journal Reprints provide readily available copies of articles published with DGOR approval, by OR researchers in learned journals, open technical publications, proceedings, etc.

ORD Contractor Reports document research done under contract of DGOR agencies by industrial concerns, universities, consultants, other government departments or agencies, etc. The scientific content is the responsibility of the originator but has been reviewed by the scientific authority for the contract and approved for release by DGOR.

DEPARTMENT OF NATIONAL DEFENCE

CANADA

OPERATIONAL RESEARCH DIVISION

DIRECTORATE OF OPERATIONAL RESEARCH (JOINT & LAND)

DOR(J&L) RESEARCH NOTE RN 9819

A DYNAMIC PROGRAMMING CONCEPT FOR STRATEGIC PLANNING

by

Mr. S. Isbrandt

Recommended by: _____

G. L. Christopher
SPORT

Approved by: _____

G. Lafond
DOR(J&L)

OTTAWA, ONTARIO NOVEMBER 1998

ABSTRACT

This concept paper explores how a Dynamic Programming methodology can contribute to a defence-oriented strategic planning analysis. A sample problem is described and solved using Dynamic Programming. Advantages and disadvantages of the methodology are presented. Implementation details and software code are included in an Annex.

RÉSUMÉ

Ce rapport conceptuel indique comment la méthodologie de programmation dynamique peut contribuer à une analyse de planification stratégique pour la défense. On décrit un problème comme exemple et il est résolut avec la programmation dynamique. Les bénéfices et désavantages de la méthode sont présentés. Les détails de la mise en oeuvre du logiciel sont inclus dans une annexe.

EXECUTIVE SUMMARY

In this note, long-term strategic planning is viewed as being intrinsically related to Canadian Forces capabilities. The driving force is to plan for a capability mix which best suits Canadian Forces requirements, and to let the desired capability mix in turn drive acquisition and maintenance programs. A methodology that supports such long-term planning would not replace work already done in the individual environments, but would provide an integrating overview of the total forces.

One of the methodologies that might be suitable for long-term strategic planning is Dynamic Programming. A small sample problem using three capabilities is described, and an implementation of the method is illustrated for the problem.

A short assessment of the Dynamic Programming method is presented. In theory, it is a method applicable to strategic planning. In practice, the usefulness of the methodology depends on whether it is possible and practical to bring together the types of input data which best suit the capabilities of the method.

In an annex, considerable detail is presented for both a software implementation of Dynamic Programming, and the solution of the sample problem.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

ACKNOWLEDGEMENTS

The author acknowledges the importance of contributions from fellow Strategic Planning Operational Research Team members in developing a concrete prototype from a vague idea. In particular, observations, comments, and questions from P. Comeau, R. Dickson, and I. Taylor were most helpful in developing ideas presented in this note.

A DYNAMIC PROGRAMMING CONCEPT FOR STRATEGIC PLANNING

Introduction

1. Within DDA 2 work is currently underway researching methodologies that hold promise for helping to conduct Canadian Forces long-term strategic planning. The purpose of this note is to present one such candidate. This is one of several possibilities that may be considered for this purpose, or that may be used to complement other methodologies in a suite of tools used to address analysis supporting strategic planning. Although the current work involving Scenarios is targeted to eventually address a time-frame of 5 to 20 years in the future, the methodology presented here can address a longer time-frame, if needed.

The Strategic Planning Problem

2. This presentation does not consider very specific characteristics of the long-term strategic planning process. Rather, some generic characteristics are used for a "concept" look at long-term strategic planning. This allows more exact considerations to be defined and discussed over time on the way to developing a more mature and realistic methodology.

3. Long-term strategic planning for the Department of National Defence is viewed as being intrinsically related to Canadian Forces capabilities. The driving force is to plan for a capability mix which best suits Canadian Forces requirements, and to let the desired capability mix in turn drive the programs which acquire and maintain equipment, facilities, personnel, and so on. The time period of analysis is not fixed, but is seen to be from 5 years in the future up to 20 or more years into the future.

4. The capabilities taken into account by the planning process are in turn related to a set of eleven scenarios, which are representative of possible Canadian Forces missions and operations. These scenarios have been developed within DGSP, with guidance from the department, for use in planning within DND (Ref. 1). By relating possible capability mixes to projected performance in the various scenarios, it is possible to "rate" the capabilities, in order to help determine "better" as opposed to "worse" capability mix for the forces.

5. The analysis foreseen by this methodology is largely an integration of analysis that would be undertaken within the three forces' environments (land, maritime, and air). The environments already carry out studies on their own particular perspectives. It is not expected that this methodology would try to replace work already done by the environments, rather, it is to provide an integrating overview of the total forces. This will help in investigating trade-offs where there may be inconsistencies or duplication between capabilities that might be provided by different environments. It will also help provide understanding as to how the capabilities provided by the three environments might best work together in a projection of the future world.

The Dynamic Programming Methodology

6. One of the potential methodologies being considered for the long-term strategic planning problem is Dynamic Programming (Ref. 2). This technique could be used to investigate long-term planning options, where each individual option consists of a series of capability levels, dynamically changing over a sequence of time periods (nominally years, or groups of years). Some of its desirable characteristics are:

a. During the analysis, it can be an effective method for reducing a large number of potential future options to a manageable number of only the best options (problem pruning).

b. Given the right characteristics for problem parameters, Dynamic Programming can determine an optimal answer, rather than just a best guess at a "good" but not necessarily optimal answer (optimizer as opposed to heuristic).

c. Implemented appropriately, the method can illustrate "what-if" differences between different options of interest to decision-makers.

d. The methodology incorporates options described over time. The phasing in and out of capabilities, or changes in capability levels, can be analyzed over a number of subsequent time periods.

e. When analyzing options over time, the method does not "lock in" to choices that seem best in the early periods, but rather takes a balanced viewpoint where the options are analyzed over the complete time period.

7. The method is amenable to a wide variety of outlooks. For instance, it could be set up to either optimize for a desired end state (e.g. during a warning time period where capabilities are ramped up for possible war) or to optimize throughout a time period (e.g. to maintain suitable capability levels over the duration of a time period).

8. The methodology requires substantial input defined in quantitative terms, rather than qualitative. These quantitative items would comprise descriptions of:

a. Capabilities and "costs" of moving from one capability level to another,

b. Objectives for the analysis, used to determine how good or bad one possible option is compared to others, and

c. Constraints which determine bounds within which feasible options exist.

9. Although the Dynamic Programming method has a number of desirable characteristics, the long-term strategic planning problem would require a large number of analysis inputs. This in turn means that considerable time would be required to obtain and verify suitable ratings, constraints, and so on in order to implement the method. As with all methods being considered, it is essential to have good, "believable" inputs in order to be able to provide a quality analysis with meaningful results.

10. The Dynamic Programming methodology as proposed here begins with an initial state, which would correspond to, say, the capabilities of the Canadian Forces today. All possible transitions from the initial stage to allowed capability options in the first stage

are enumerated and recorded. The effectiveness or desirability of each of the options reached in the first stage is rated against some pre-defined objective. Any constraints (cost or otherwise) which restrict the capability options allowed are taken into account.

11. Then, all possible transitions from the first stage to the second stage are enumerated, recorded, and rated. Each of the feasible options that can be reached in the second stage retains a "memory" of the option which preceded it in the first stage.

12. The methodology repeats this process throughout the time period of analysis. All transitions from options at one stage to the next stage are examined, each of the resulting options is rated, and a "memory" of how best to reach each possibility is retained. Again, constraints are allowed for as necessary.

13. Eventually the last stage is reached, an option that is rated highest according to the pre-defined objective is identified, and the intermediate options can be identified to show how a progression can be made from the initial option to be desired last option.

The Sample Problem

14. For the purpose of this note, a very simplified smaller problem, but with characteristics typical of the larger strategic planning problem, will be used to illustrate the dynamic programming methodology.

15. The sample problem used to illustrate the Dynamic Programming methodology is nominally based on a simple "collective defence" scenario, with only three defined capabilities. An initial capability level is specified for a base time 0, and possible capability options are analyzed over 5 subsequent time periods.

16. The three capabilities are:

a. Medical Support,

b. Lift and Position, and

c. Command & Control (Joint Force Headquarters)

17. Transition costs, for maintaining capabilities at certain levels, or for changing capability levels from one time period to another are defined as inputs to the methodology. Constraints such as costs or threshold performance measures are defined and incorporated into the methodology as well.

18. An "objective" measure is also required in order to determine the desirability of any mix of capabilities. This allows the identification of "better" or "worse" capability mix options over time, from the initial base time to the end of the last time period.

Possible Solution Methodology Oriented to Dynamic Programming

Dynamic Programming Characteristics for the Sample Problem

19. The sample problem represents a planning process starting with an initial stage 0 and progressing through subsequent stages until stage 5.

20. The transition costs for a change in any capability level are shown in Table 1.

Table I - Capability Transition Costs

CAPABILITY	START LEVEL	TRANSITION LEVEL	COST
Medical Support	1	1	0
	1	2	4
	1	3	9
	2	1	0
	2	2	1
	2	3	5
	3	1	0
	3	2	2
	3	3	2
Lift & Position	1	1	0
	1	2	10
	1	3	18
	2	1	0
	2	2	5
	2	3	20
	3	1	0
	3	2	0
	3	3	10
C&C (JFHQ)	1	1	0
	1	2	5
	1	3	20
	2	1	0
	2	2	2
	2	3	8
	3	1	0
	3	2	0
	3	3	5

21. For simplicity in the sample problem, it is assumed that the transition costs for a change in any capability level are constant over the time period of the analysis. The methodology can be made more realistic by incorporating a time dimension and defining transition costs for each individual time period.

22. Again, for simplicity in the sample problem, the "total" transition cost from one capability mix at one stage, to another capability mix at the next stage, is simply the sum of the individual capability transition costs. A more realistic approach, which can be

incorporated within the methodology, is to cost the transition of moving from one vector of capabilities, to another vector of capabilities (which may be different from the sum of the individual capability transition costs), taking the time dimension into account. The overall logic of the methodology remains the same, but it necessitates that more complicated data structures must be accommodated in any program code.

23. Constraints can be included in the dynamic programming process. As option transitions from one stage to another stage are investigated, any applicable constraints are taken into account. This may potentially flag an option as infeasible. In the sample program, one constraint applied is that the total cost of capability transitions in moving through the time periods cannot exceed ten times the number of periods. So, for example, at the end of the fifth period, the total cumulative cost cannot exceed 10 * 5 = 50. This particular constraint has the effect of allowing "banking" of resources. For instance, one feasible solution may involve a cost of 8 going to stage 1, and a cost of 12 going to stage 2. Although one of the period costs is in excess of the average allowed per year, the cumulative cost is still within the overall constraint. All transitions will have to be consistent with this cumulative cost constraint.

24. Another constraint applied is that any capability mix allowed as a feasible option must meet at least a certain threshold of performance capability. The intent of this is to ensure that during the analysis no particular capability mix at one time period will be "sacrificed" to low performance levels, in order to "up" the other capability mixes in the other time periods for a higher average capability.

25. Enforcing constraints such as the ones described above simply reduce the "solution space" to be examined by the methodology, by screening out options which would not be allowed as a final solution in any case.

26. A Dynamic Programming analysis requires an objective in order to judge which capability mix options are preferable to others. Possible objectives could be based on paradigms such as:

a. One encompassing measure of effectiveness of the sets of capability mixes over the time periods.

b. Some overall confidence or reliability measure that any significant defence requirement will be adressed throughout the time period of analysis.

c. The objective could be based on the average measure of capability mixes over all periods, or by maximizing the objective at only the end period, and so on.

27. The objective measure that is best to use for the purpose of analysis will depend on what are the important driving factors encompassing the strategic planning process.

28. For the sample problem, a simple "reliability measure" was used as the objective measure for ***illustrative purposes*** only. ***At any given stage***, a reliability term is calculated as:

$$(0.9999- ((3-CAP1) /12)) * (0.9999-((3-CAP2)/6)) * (0.9999-((3-CAP3)/12))$$

where CAP1 corresponds to the Medical Support capability level, CAP2 corresponds to Lift & Position, and CAP3 corresponds to Command & Control, (JFHQ).

29. Note that when capability levels are high at 3, nothing will be subtracted from the 0.9999 term paired with the capability, and the individual capability level will not reduce the overall measure of reliability. On the other hand, as capabilities are rated at lower levels, they will impact the overall product measure. The overall reliability will be more sensitive to the CAP2 term, (since it is divided by 6 rather than by 12) – so, lower values of CAP2 will reduce overall reliability more than equivalent lower values of either CAP1 or CAP3. This reflects an assumption in the problem that CAP2 is more critical to overall reliability, and that CAP1 and CAP3 have lesser but equal impacts.

30. ***The overall objective measure is applied over all the stages*** – in this case we use as a measure the average reliability up to and including the stage being examined, resulting in an overall average when the final stage is completed. So, the intent of the example is to find the set of options over all stages that maximizes the average reliability of the capability mix over the given time period.

Dynamic Programming Algorithm Concept

31. As mentioned before, the Dynamic Programming methodology begins at an initial state. All possible transitions from the initial stage to allowed options in the first stage are enumerated and recorded. The effectiveness of each option is rated against an objective. Any constraints that restrict the capability options are considered.

32. Then, all possible transitions from the first stage to the second stage are enumerated, recorded, and rated. What this means, in more detail, is that every option in the second stage is considered, one by one. All possible transitions from feasible options in the first stage to a particular second stage option are examined, and any transition is discarded from the process if it violates problem constraints. Of those transitions that remain (if any) the one associated with the best objective value is chosen as the preferred route to reach that particular second stage option. (If no feasible transition exists to a second stage option, it is discarded from any possible solution.) After looking at all the second stage options, each of them that can be reached by a feasible transition retains a "memory" of its best predecessor (i.e. the stage one option which provides its best transition path).

33. The methodology repeats this process. All transitions from options at one stage to the next stage are examined, rated, and a "memory" of how best to reach each is retained. Again, constraints are allowed for.

34. Eventually the last stage is reached, the option with the highest rating according to the objective is identified, and the transition path can be identified to show how a progression can be made from the initial option to be desired last option.

35. This process is illustrated below as a logic diagram, in the context of time periods expressed as years:

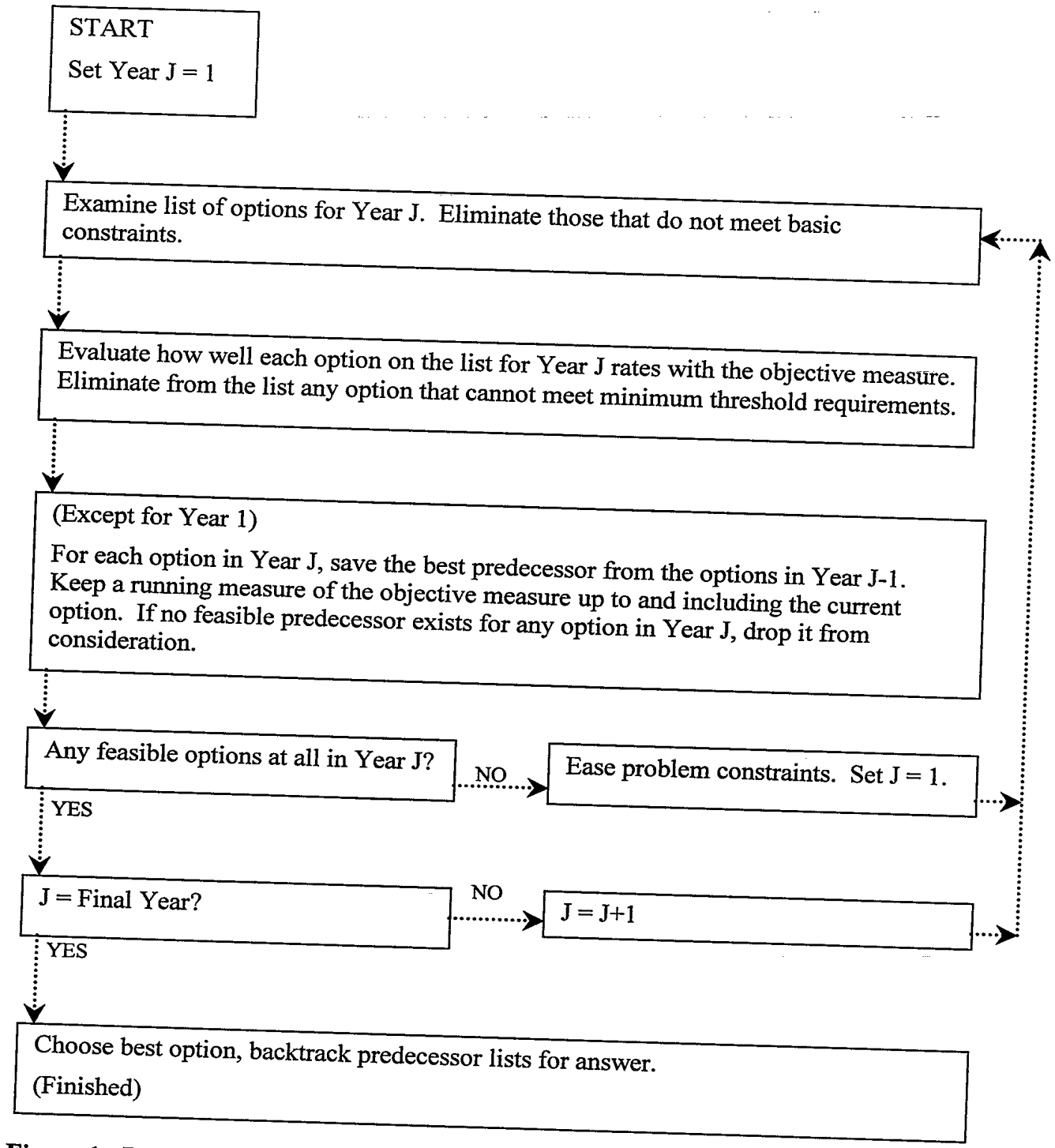


Figure 1 - Dynamic Programming Logic

Sample Problem Solution

36. Details of the solution procedure for the sample problem are described in the Annex. The generated solution is summarized in the following table:

Table II - Generated Solution

Stage	Option	Medical Support	Lift & Position	C&C (JFHQ)	Predecessor	Cumulative Cost	Average Reliability
5	23	3	2	2	23	47	0.68
4	23	3	2	2	22	38	0.67
3	22	3	2	1	22	26	0.63
2	22	3	2	1	10	19	0.60
1	10	2	1	1	1	4	0.51
0	1	1	1	1	N/A	0	N/A

37. The starting position for the sample problem is one unit of each capability at stage 0. This position is referred to as option 1, and is represented by a vector [1, 1, 1], which corresponds to the levels for Medical Suport, Lift & Position, and C&C (JFHQ) respectively. Similarly, options are defined from option 2 (as [1, 1, 2]), option 3 (as [1, 1, 3), and so on through option 27 (as [3, 3, 3]). For the sample analysis, all 27 options are considered at each stage.

38. There are only 4 feasible transitions from the initial stage option [1, 1, 1] to stage 1. The four resulting feasible options are [1, 1, 2], [2, 1, 1], [2, 1, 2], and [3, 1, 1] (i.e. options 2, 10, 11, and 19). If one were to stay at the same capability level of [1, 1, 1] a threshold reliability constraint applied in the example, where the reliability at any stage must not fall below 0.5, would be violated. The other non-feasible transitions are excluded because they exceed the allowed cost constraint of 10 per stage. For example, a transition from [1, 1, 1] to [2, 3, 2] when examining the cost Table I results in a cost of 4+18+5 = 27 total cost, which is considerably in excess of the allowed cost.

39. Continuing from stage 1, all of the transitions between stages 1 and 2 which have one of the four feasible options as a starting point are examined. Reliability threshold constraints and cost constraints are applied to "weed out" non-allowed transitions, and the rest are retained. If an option at stage 2 can be reached from more than one predecessor, the predecessor which contributes to the larger objective value is retained.

40. Stage by stage, the process continues until the options at the final stage have been reached and examined. (A detailed representation is shown in the Annex.)

41. The "best" option mix having the maximum average reliability over all five periods, given the constraints already mentioned, is reflected by option 23, with a Medical Support level of 3, a Lift & Position level of 2, and a Command and Control (Joint Force Headquarters) level of 2. The optimal predecessor at period 4 was also option 23, and the cumulative cost throughout the five time periods was 47. By backtracking through the predecessor chain, a complete picture of the optimal option mixes over all five time periods can be determined.

Assessment / Conclusion

42. The Dynamic Programming methodology is a powerful technique for generating a sequence of option decisions over time, providing the problem to be analyzed can be stated in terms required by the methodology. Based on the concept trial described here, the methodology looks promising enough to warrant a further trial on a larger problem, one which is larger and more complex in the nature of a realistic strategic planning problem.

43. The methodology requires a sizable amount of input related to operational capabilities, such as:

a. transition costs and constraints from one capability mix option to another capability mix option,

b. specification of all operational capability mixes which are to be considered over an extended time period,

c. identification of cost and performance constraints which limit the feasible options which can be considered at different time periods, and

d. determination of performance, reliability, or some other measure to use in judging which options perform better than others.

44. Whether this sort of input can be practically obtained and implemented may be an important factor as to whether the methodology can eventually be used.

45. It may turn out that other tools, such as H-Frame, could be used to help determine appropriate input for a tool based on the Dynamic Programming methodology. H-Frame is a software application developed for the Department of National Defence, based on work done on Hierarchial Analysis Frameworks (Ref. 3).

46. Also, the results of the methodology might in turn be used by another analysis tool. For instance, candidate solutions from Dynamic Programming analyses using different input assumptions might be considered within the CEPS (Capital Equipment Plan Simulator, Ref. 4) to help understand the differences in impact between them in terms of an actual acquisition plan.

References

1. Concept Paper – Departmental Force Planning Scenarios (FPS), DGSP/DDA, October 16, 1997.
2. Hillier F.S., Lieberman G.J., Introduction to Operations Research, Fourth Edition, Holden-Day, Inc., Oakland CA, 1986.
3. Mason D.W., The Applicability of Hierarchial Analysis Frameworks (HAFs) To High-Level Defence Problems, DLOR Staff Note 90/18, November 1990.
4. Lefebvre, Capt L., Methods For Scheduling Capital Equipment Projects, ORD Project Report PR 9612.

ANNEX A –Sample Program Description

User Guide to the Sample Program

The program described here was developed using Delphi, as a prototype concept illustration of how the Dynamic Programming methodology could be implemented. (Further software details are included later in the Annex.) When the application is started, the user is met with the following screen:

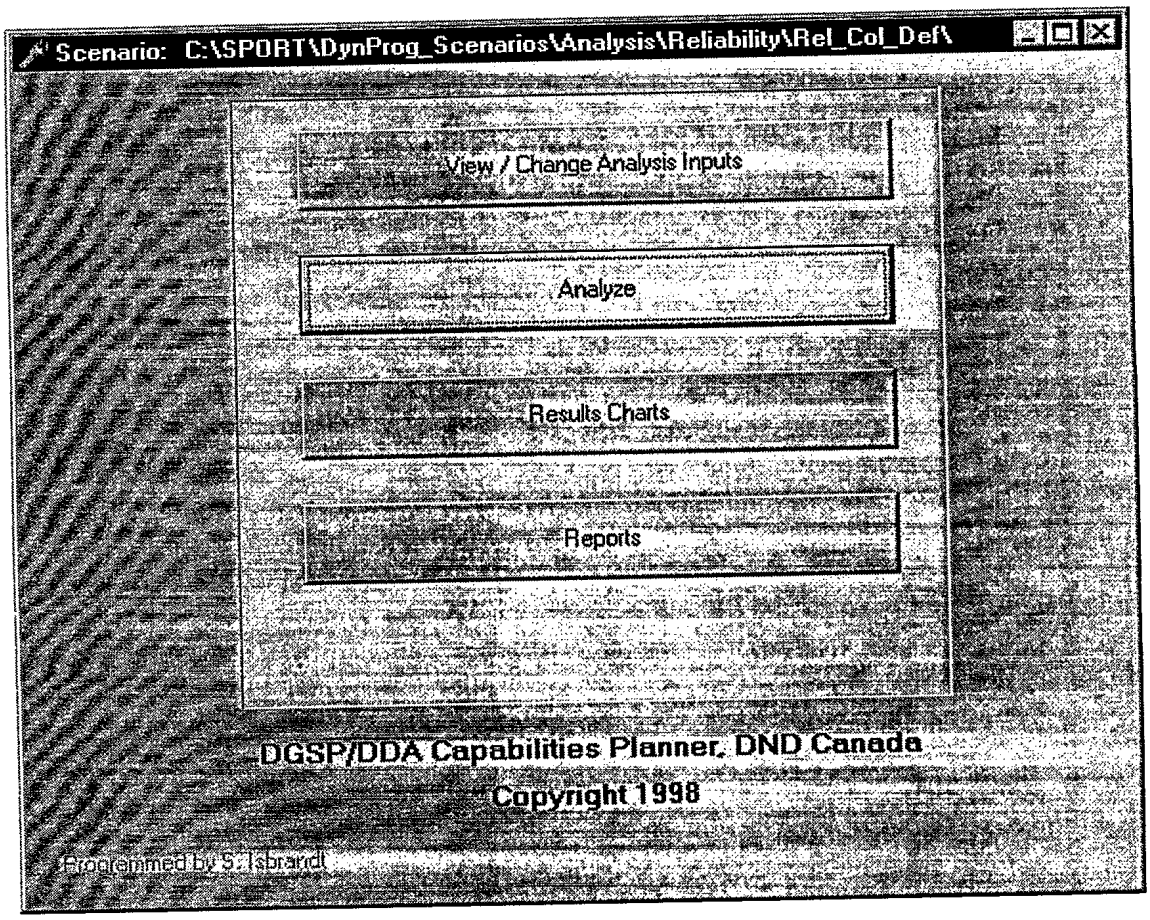


Figure A - 1 Main User Screen

Upon choosing the "View / Change Analysis Inputs" button, the user is first informed that they will have to specify a folder location where database files for the scenario to analyze can be found,

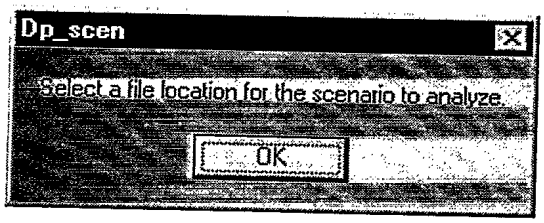


Figure A - 2 Select Input Location Prompt

and then they are prompted to actually specify the folder with an Open Dialog.

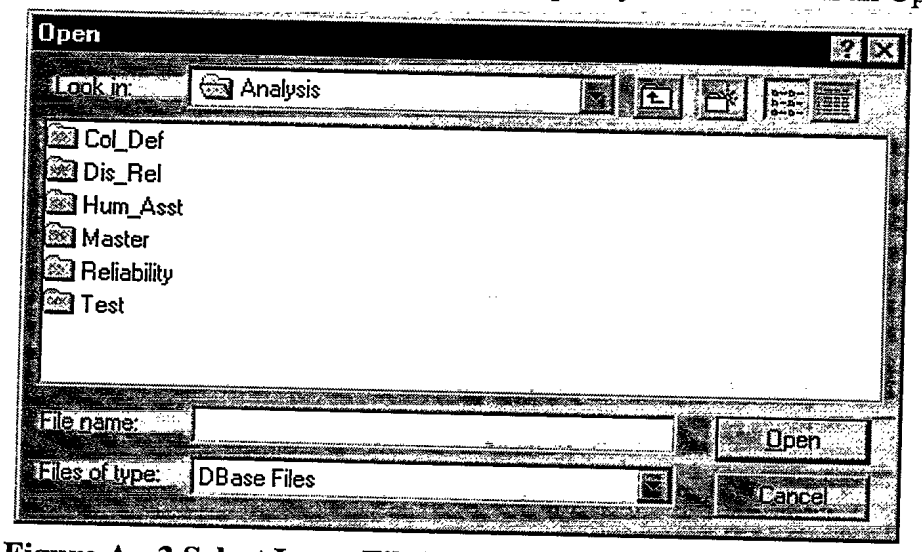


Figure A - 3 Select Input File Dialog

The program proceeds to display basic input information contained in the scenario database inputs, as shown in Figure A - 4:

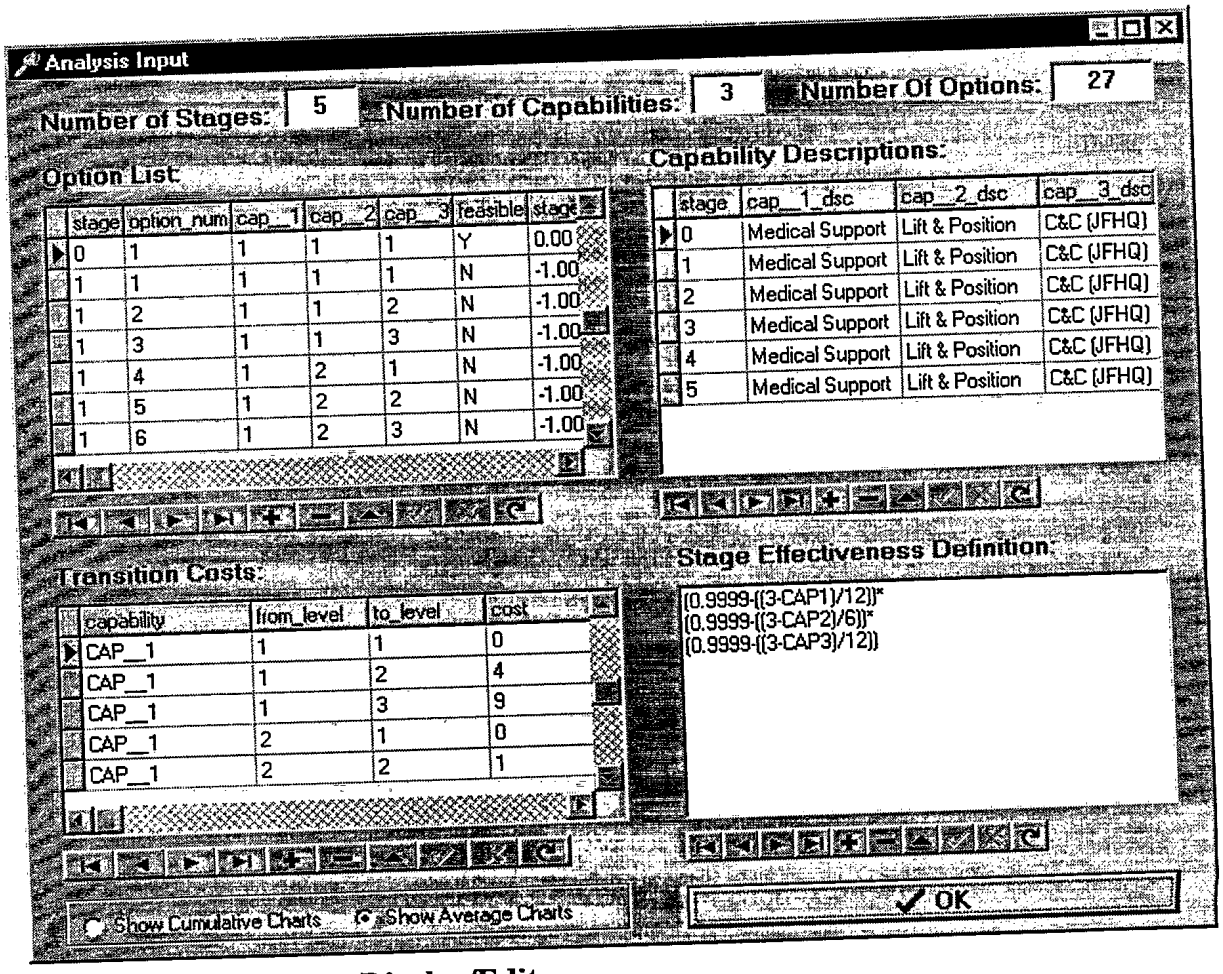


Figure A - 4 User Input Display/Edit

In Figure A - 4, the option list simply enumerates the option mixes which are to be considered at each time period or stage, and the transition costs specify how much a change in any capability level costs when moving from one capability value to another. The capability descriptions table simply ties together the model format for tracking capabilities (CAP__1, CAP__2, etc.) with a descriptive label. The display shows only a portion of all the options – for a large number of capability types and possible levels the list of options could be very long. This raises the issue of input data specification and management, which is not dealt with here, but should be at least recognized as a potential difficulty.

The number of stages or time periods, the number of capabilities used in the analysis, and the number of options at each stage are indicated at the top of the form.

The Stage Effectiveness Definition is a mathematical expression involving the capabilities, used to determine an "objective value" for any capability mix option during the analysis. The program during execution parses this expression. So, different objectives can be tried out simply by specifying different expressions in a database, rather than by hard-wiring the objectives into program code.

Presently, the cost constraints are hard-wired into the concept program code, but they also could be set into a format where they would be set at run-time instead.

The user can change information by entering changes directly on the form, and "posting" them (hitting the curved arrow on the navigation bar below the appropriate table) to the database. The form can be closed, to return to the main screen.

By hitting the "Analyze" button, the program is directed to carry out the dynamic programming algorithm based on the information that was previously displayed.

By depressing the "Results Charts" button, a new screen comes up which displays two choices, one for overall results ("Show Stages") and one for the best allocation of options mixes over all of the stages ("Show Max Return").

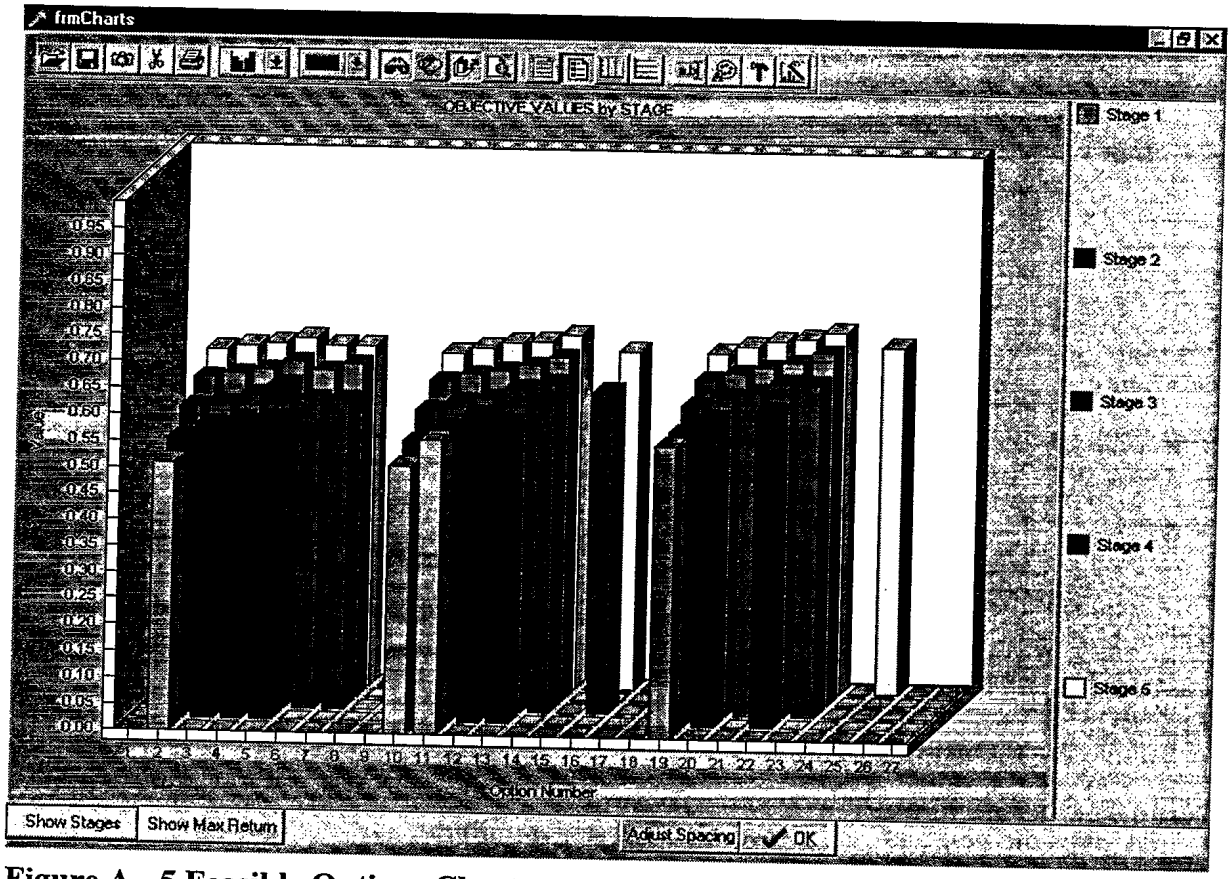


Figure A - 5 Feasible Options Chart

("Show Stages") shows all feasible (or reachable, given model constraints) options throughout the five stages of the sample problem.

The first row of the chart has only four values, representing the only four option mixes which are reachable at stage one from the initial state of the system. The second row represents the option mixes which are feasible at the second stage, and which are also reachable by feasible options mixes at the previous stage. The third, fourth, and fifth rows continue out similarly. The height of the bars represents the "average effectiveness" over all stages, using the "best" set of option mixes from the initial state through to the stage represented by the bar.

The toolbar above the chart can be use for numerous functions as desired: change to a line or other chart style, change colours of chart elements, show or hide the chart series legend, zoom in on a chart detail, and so on.

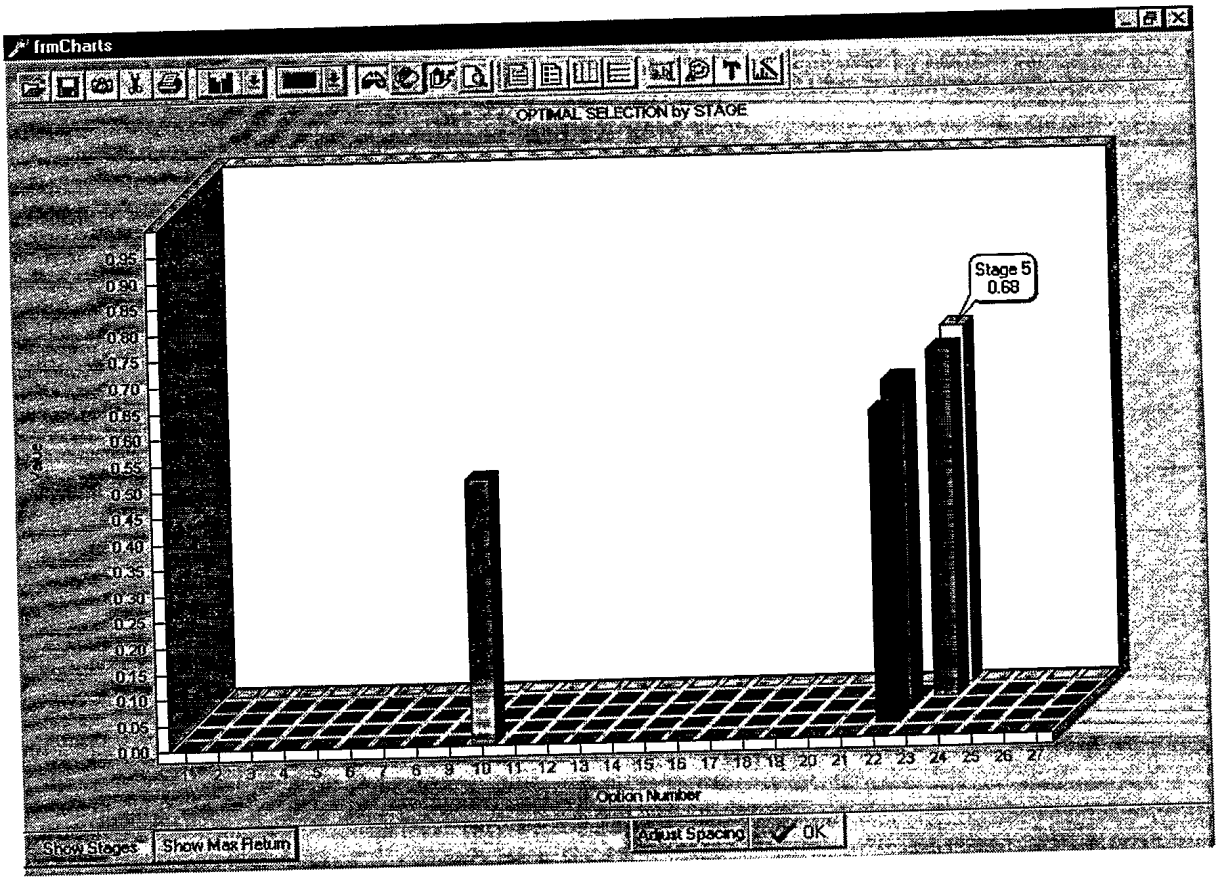


Figure A - 6 Optimal Option Chart

Figure A - 6 Optimal Option Chart ("Show Max Return") shows the particular sequence that gives the best answer over all the five stages. Double clicking on the bar (as illustrated above) can raise a balloon prompt showing the value of any particular bar. This answer corresponds to the solution shown in the main paper, and diagrammatically illustrated in detail elsewhere in this Annex.

From the main application screen, the user can also activate a "Reports" button. This brings forward a form as shown in Figure A – 7 where one of three reports can be chosen from a list. The report can be displayed by activating the "Show Report" button.

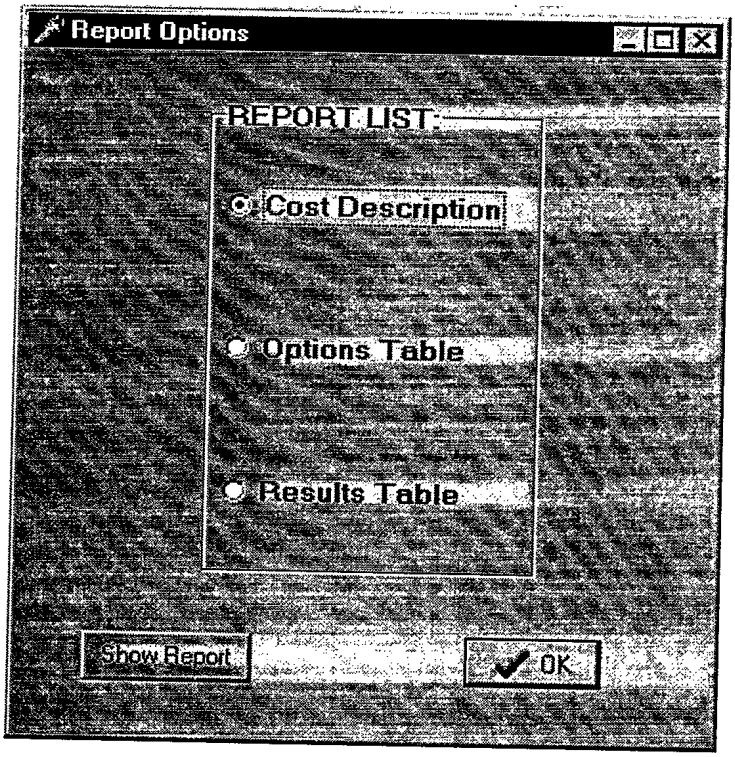


Figure A - 7 Report Selection List

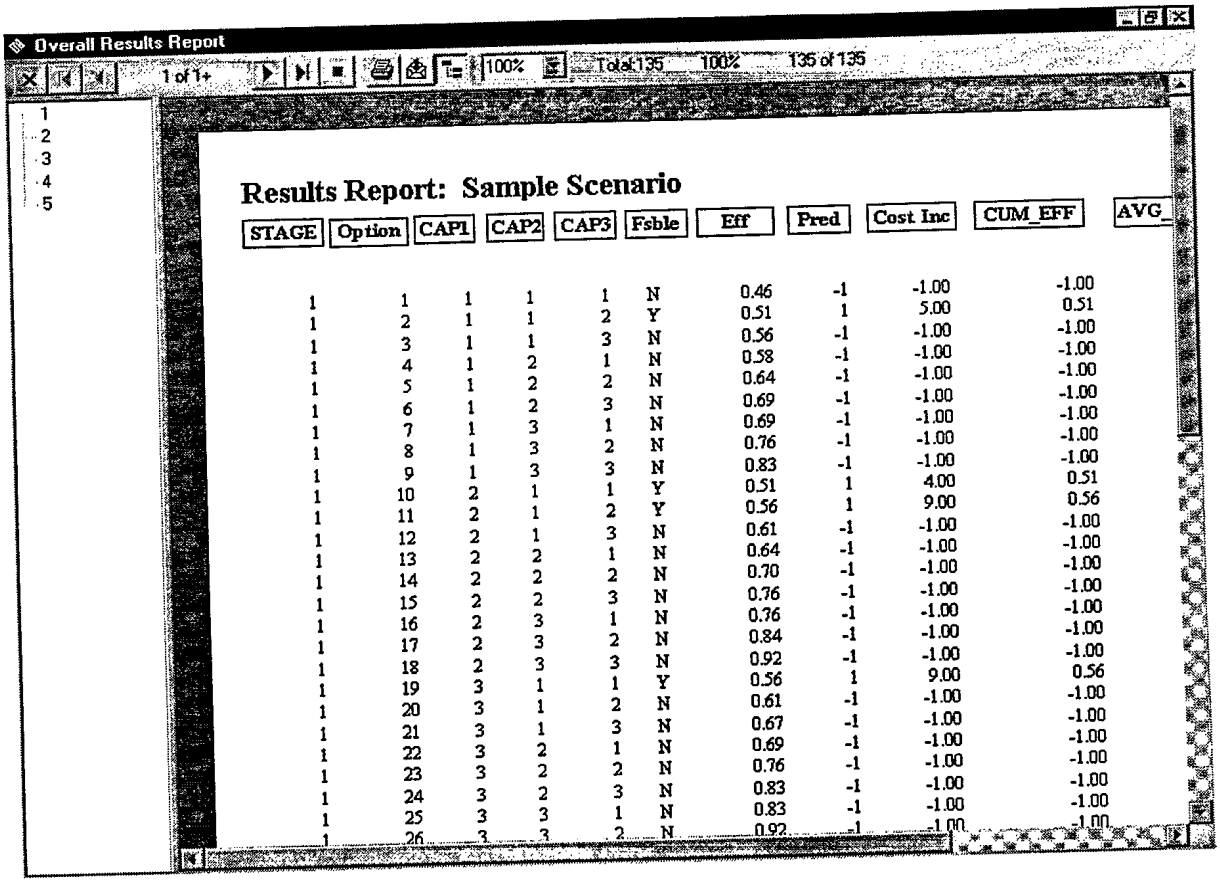


Figure A - 8 Report Screen

This will introduce a document screen similar to above, where the relevant report can be examined, printed out, or exported to a disk file in a variety of formats.

Detailed Results Diagrams of the Sample Analysis

In the following pages, the solution process carried out by the Dynamic Programming application on the sample problem is shown diagrammatically.

Figure A - 9 shows the "solution space" available initially, and within the five time periods or stages. Stage 0 shows the initial state with a capability vector [1, 1, 1] which represents a Level 1 for the Medical Support capability, a Level 1 for the Lift & Position capability, and a Level 1 for the C&C (JFHQ) capability. Stage 1 shows 27 capability option vectors ranging from [1, 1, 1] (all capabilities at lowest level considered) to [3, 3, 3] (all capabilities at highest level considered). The other stages similarly have 27 option vectors shown.

In a more realistic problem, the capability option vectors considered would not necessarily be the same from one stage to another.

Figure A - 10 represents the same solution space of capability options, but represents them as a series of points. So Stages 1 through 5 have a point corresponding to [1, 1, 1] at the bottom of a column of 27 points representing options to [3, 3, 3] at the top.

Figure A - 11 illustrates how the initial "fan-out" from the initial capability option to those options at Stage 1 which are feasible – they are within cost limits, performance constraints, and basically satisfy all restrictions which have been applied as part of the problem setting.

Figure A - 12 continues showing the continuing chain of feasible options at Stage 2 which are "reachable" from feasible options at Stage 1. Where an option at Stage 2 might be reached by more than one route, the path which allows the option to be reached with the greatest objective value is the one that is recorded. (This is a very important part of the Dynamic Programming methodology.) Note that not all feasible options at Stage 1 have a way of continuing on to Stage 2 within the problem restrictions. This reflects the case where an option at Stage 1, although feasible and reachable, does not make sense because there are no possibilities it can use to lead to a feasible Stage 2 option.

Figure A - 13, **Figure A - 14**, and **Figure A - 15** detail the forward chaining process, from feasible options at one stage to feasible options at the next stage, up to Stage 5.

Figure A - 16 highlights with a star the one feasible option at Stage 5 which has the greatest value for the problem objective (option vector [3, 2, 2].

Figure A - 17 illustrates the backtracking process from the best option at Stage 5 to the best feasible predecessor option at Stage 4 (option vector [3, 2, 2] again).

Figure A - 18, **Figure A - 19**, **Figure A - 20**, and **Figure A - 21** show the complete backtracking process back to the initial option. It can easily been seen from the diagrams that none of the "top" options such as [3, 3, 3] appear as feasible options, since they exceed cost limits. Similarly, none of the "lower" options appear on the best path, since they are superceded by other superior options which are affordable, and provide better objective values.

The path shown in the figures corresponds to the **Table II** solution shown in the main paper, and the **Figure A - 6** Optimal Option Chart shown in the application User Guide of this Annex.

		[3,3,3]	-	-	-	[3,3,3]
		-	-	-	-	-
		-	-	-	-	-
		[1,2,1]	-	-	-	-
		[1,1,3]	-	-	-	-
		[1,1,2]	[1,1,2]	-	-	[1,1,2]
	[1,1,1]	[1,1,1]	[1,1,1]	-	-	[1,1,1]
Stage	0	1	2	3	4	5

Figure A - 9: Diagram of Capability Vectors

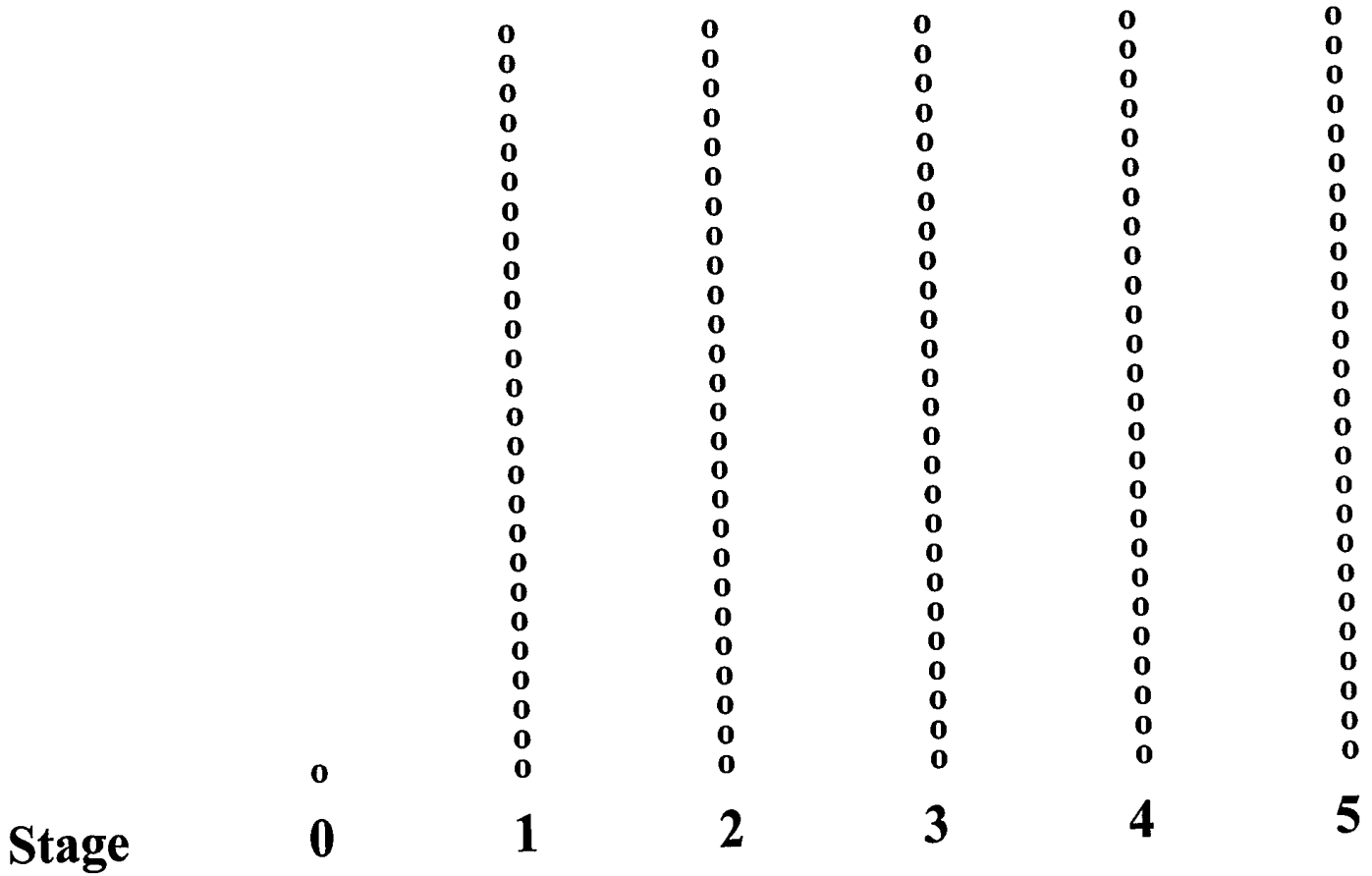


Figure A - 10: Capability Vectors Represented as Points

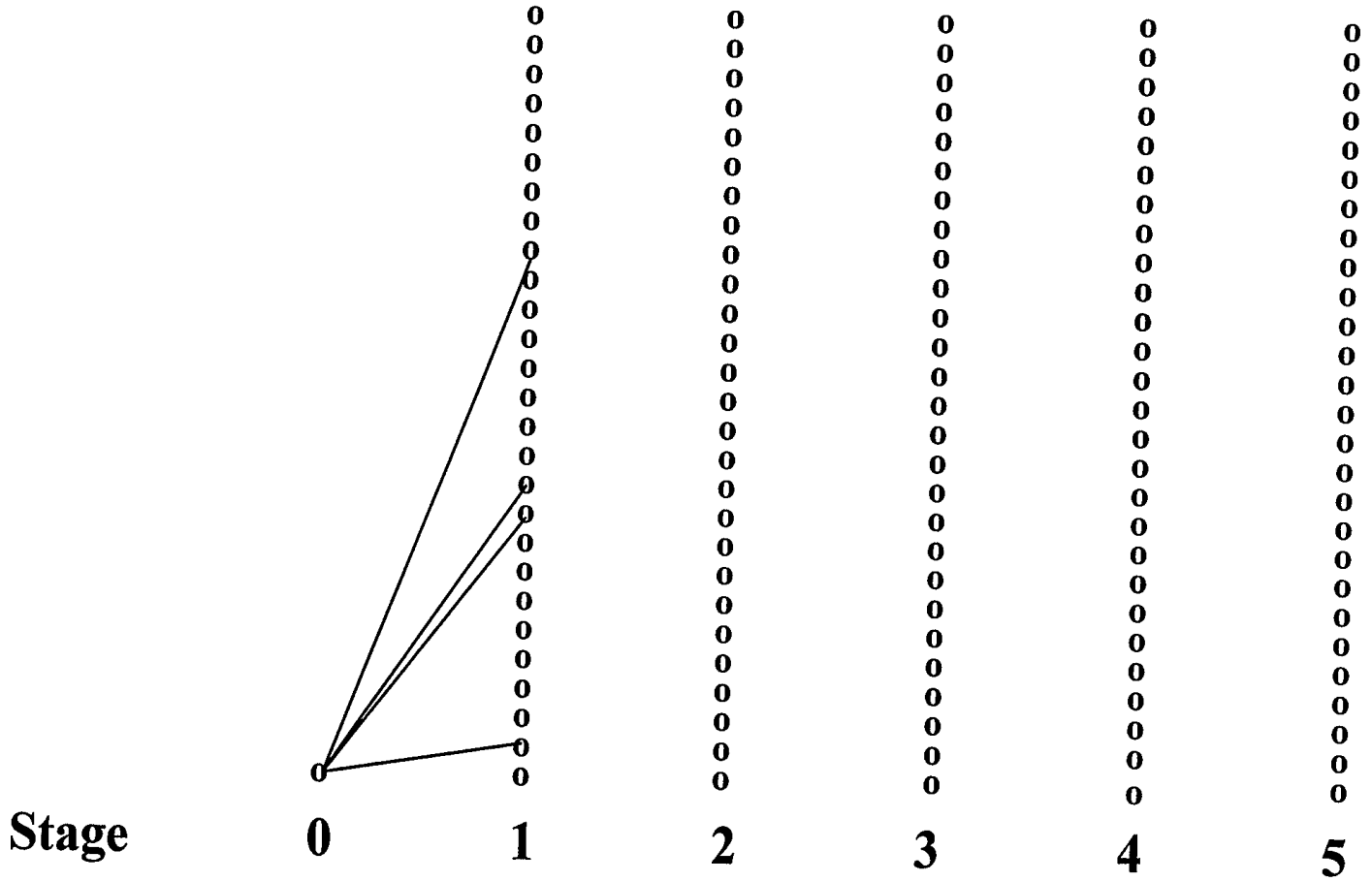


Figure A - 11: Feasible Stage One Options

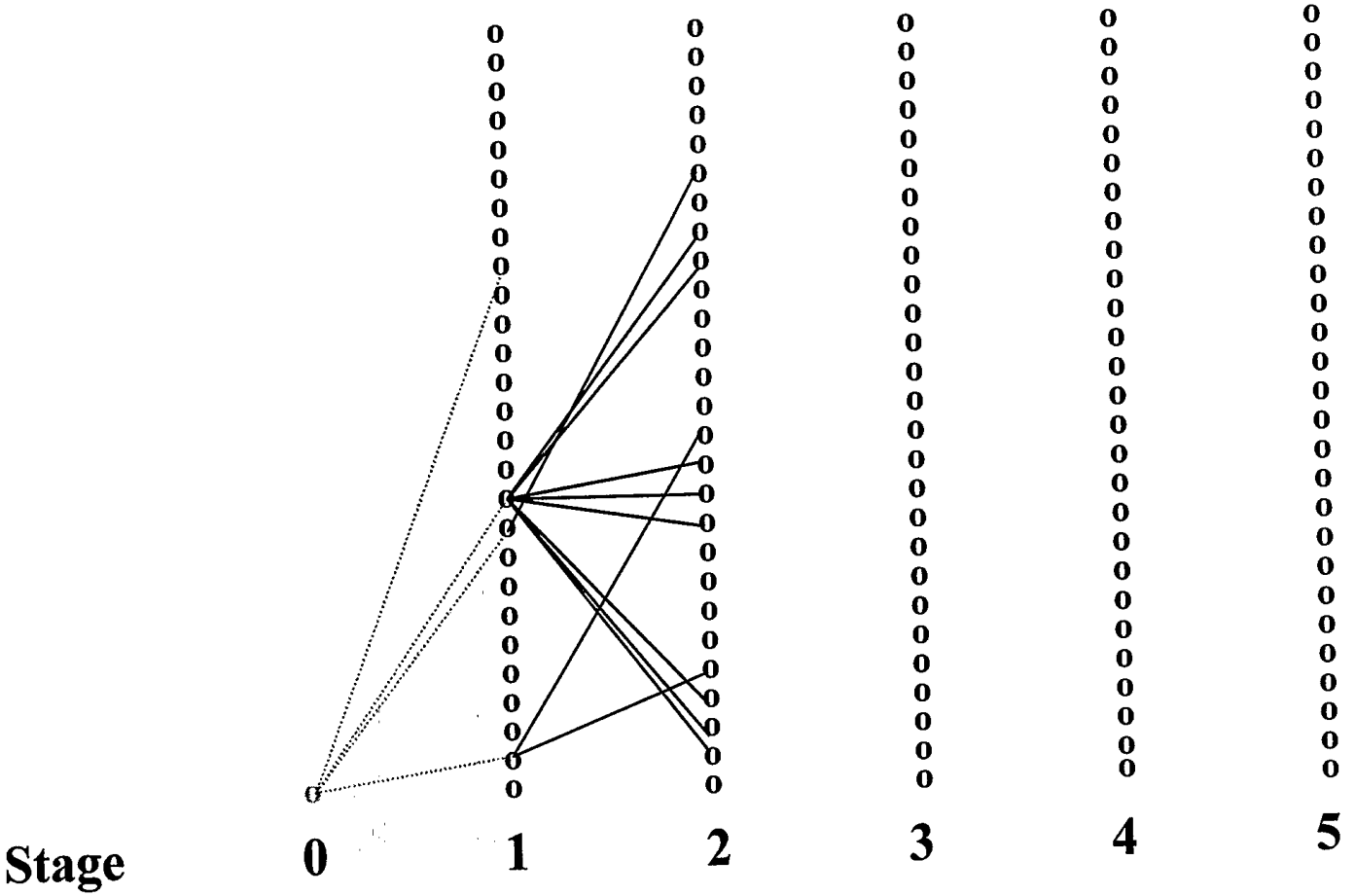


Figure A - 12: Stage Two Feasible Options

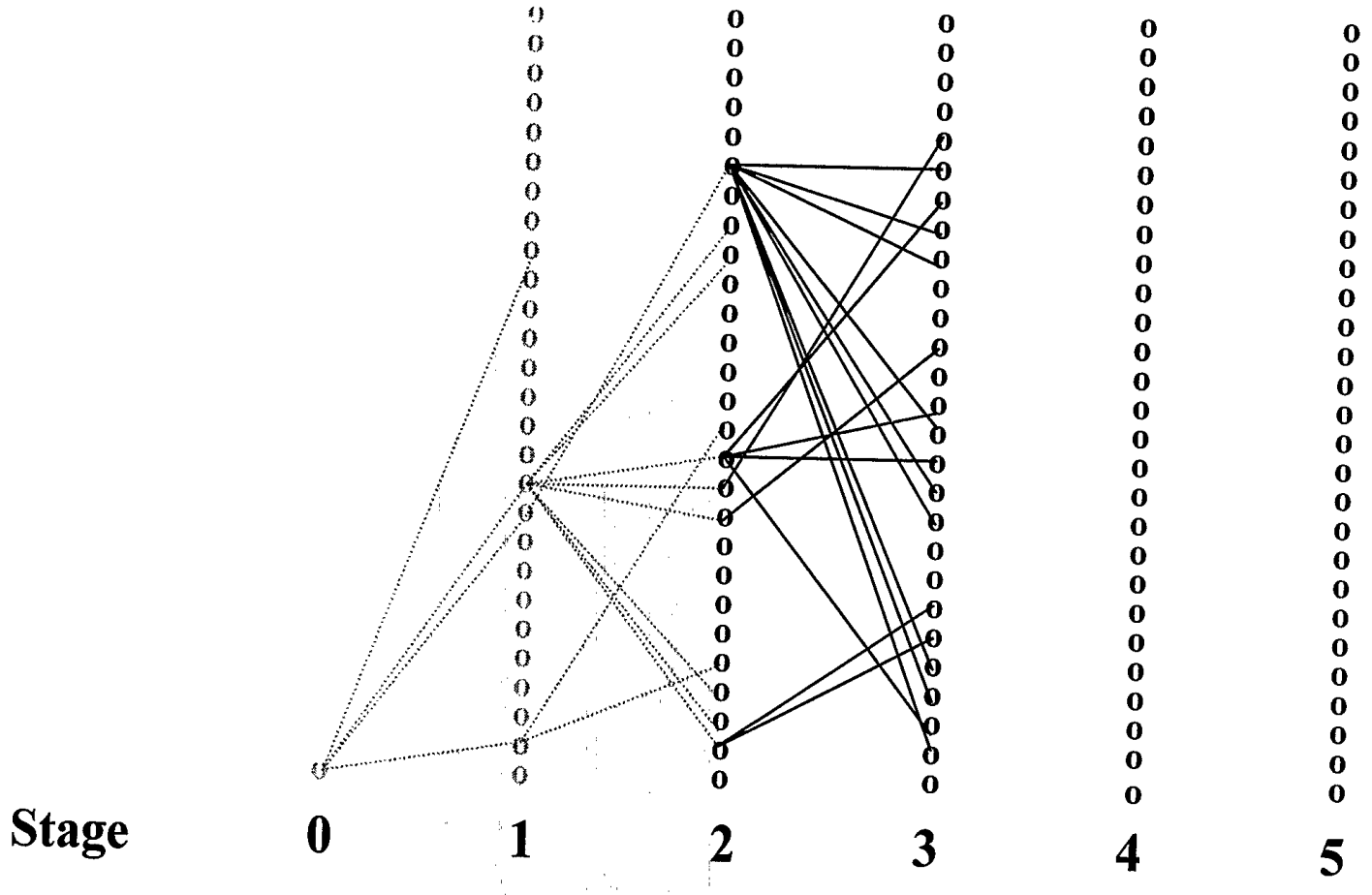


Figure A - 13: Stage Three Feasible Options

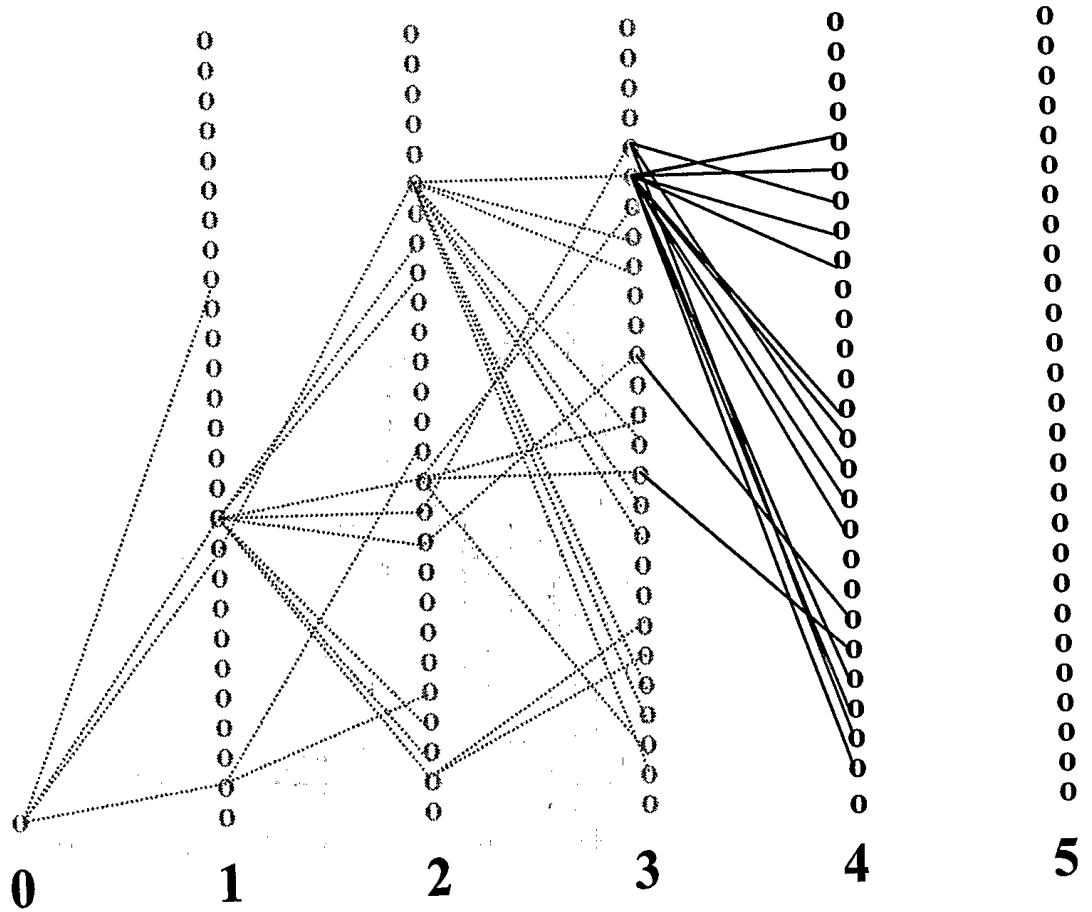


Figure A - 14: Stage Four Feasible Options

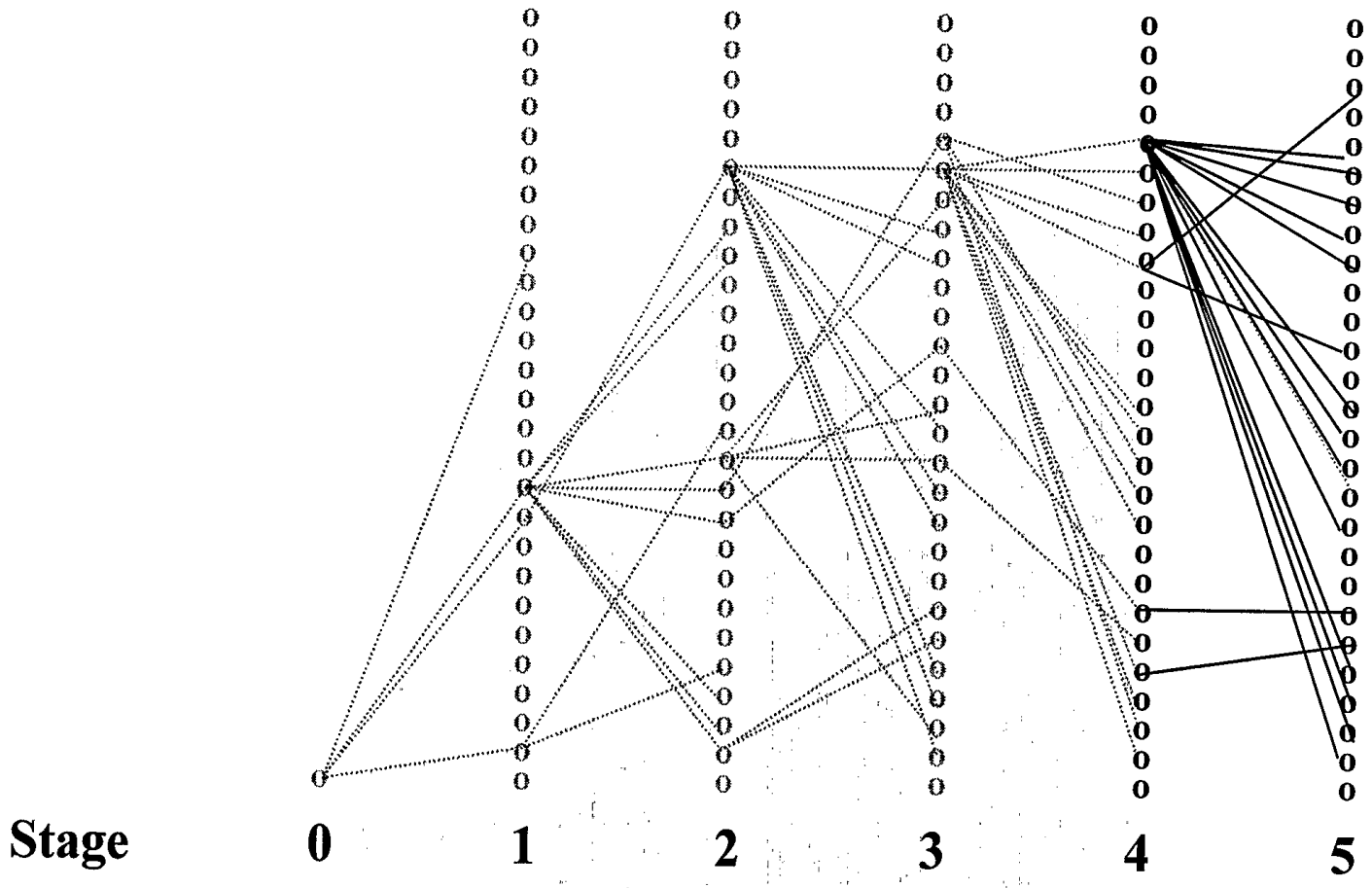


Figure A - 15: Stage Five Feasible Options

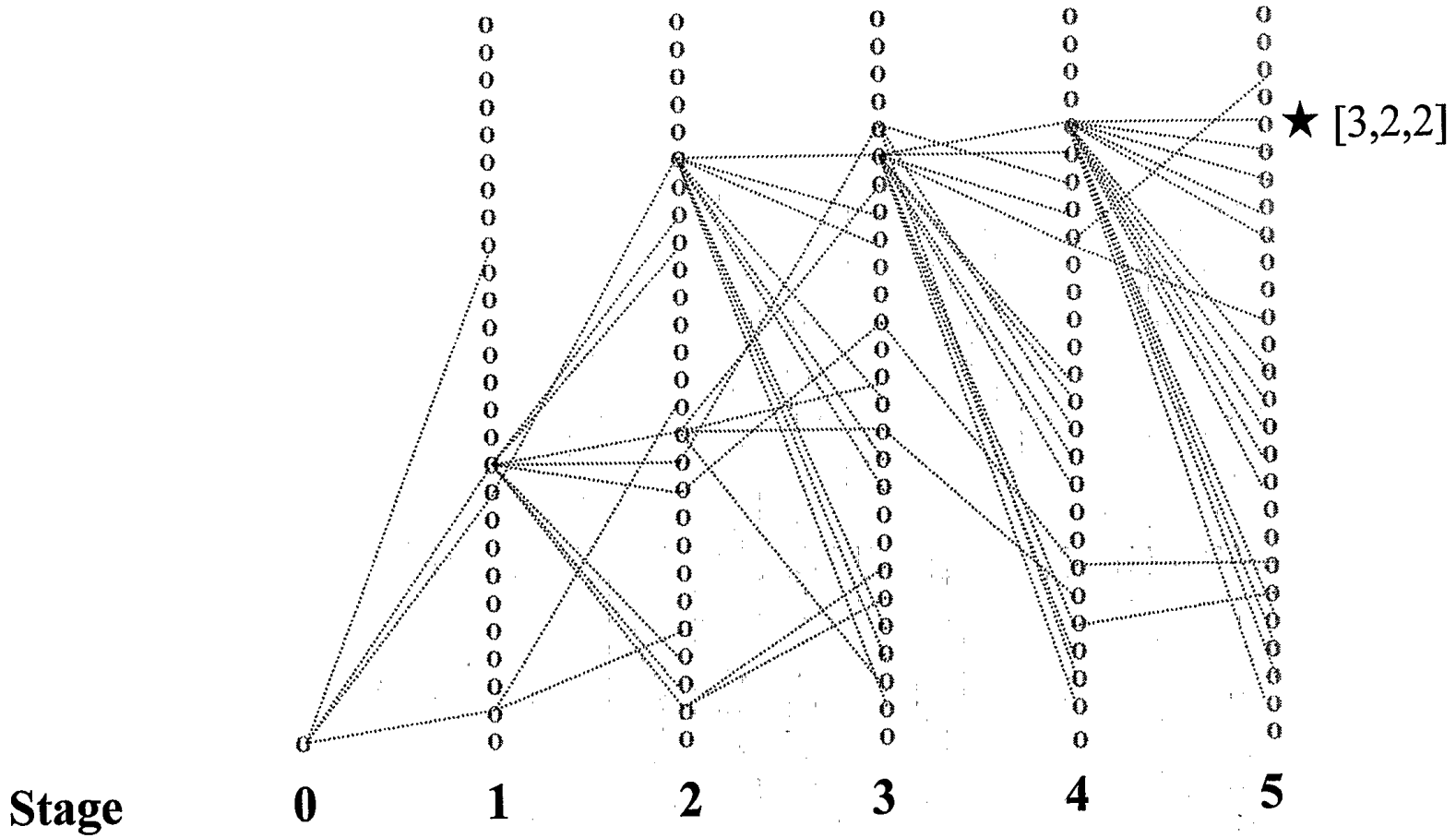


Figure A - 16: Stage Five Overall Optimum

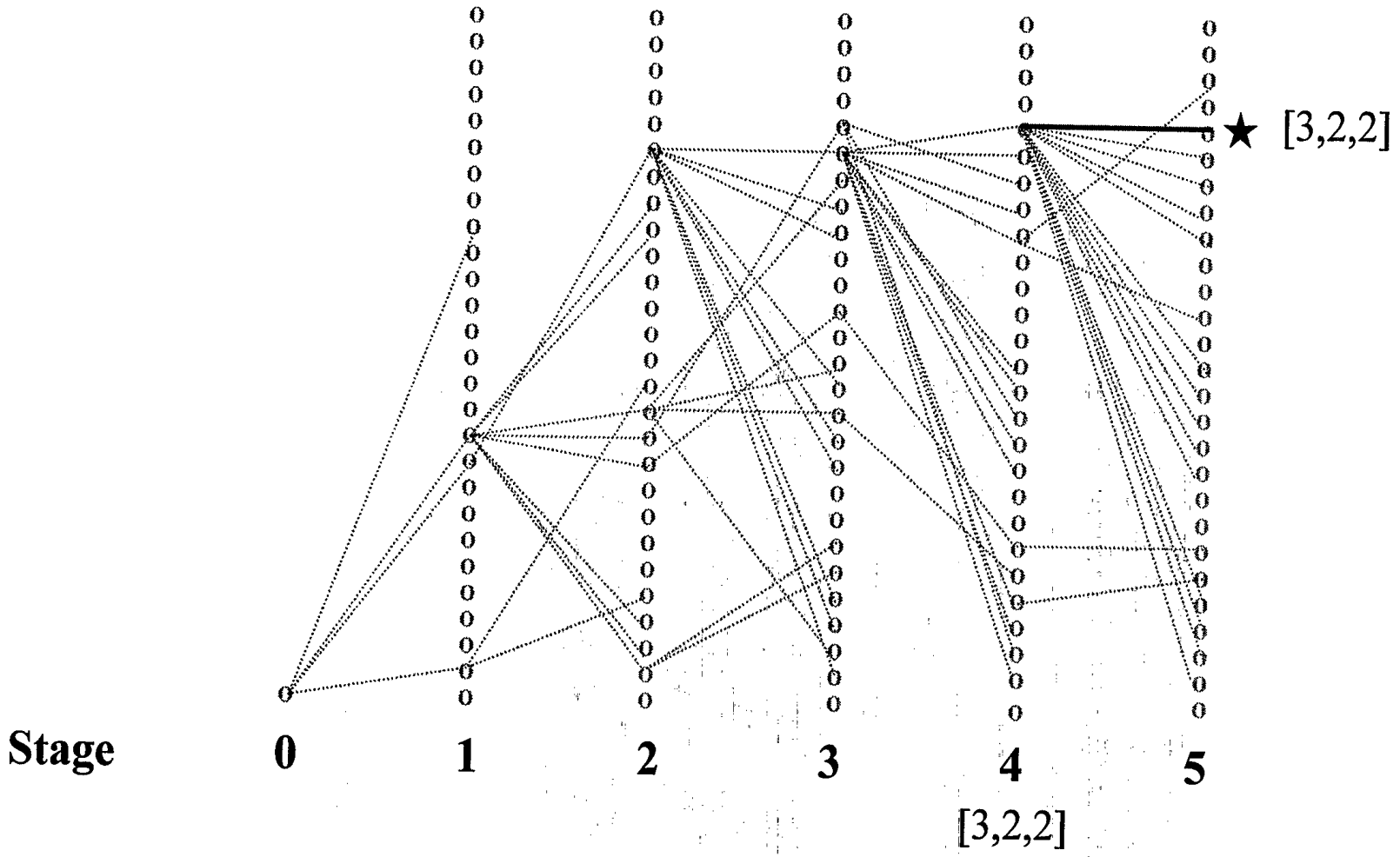


Figure A - 17: Backtracking to Best Option at Stage Four

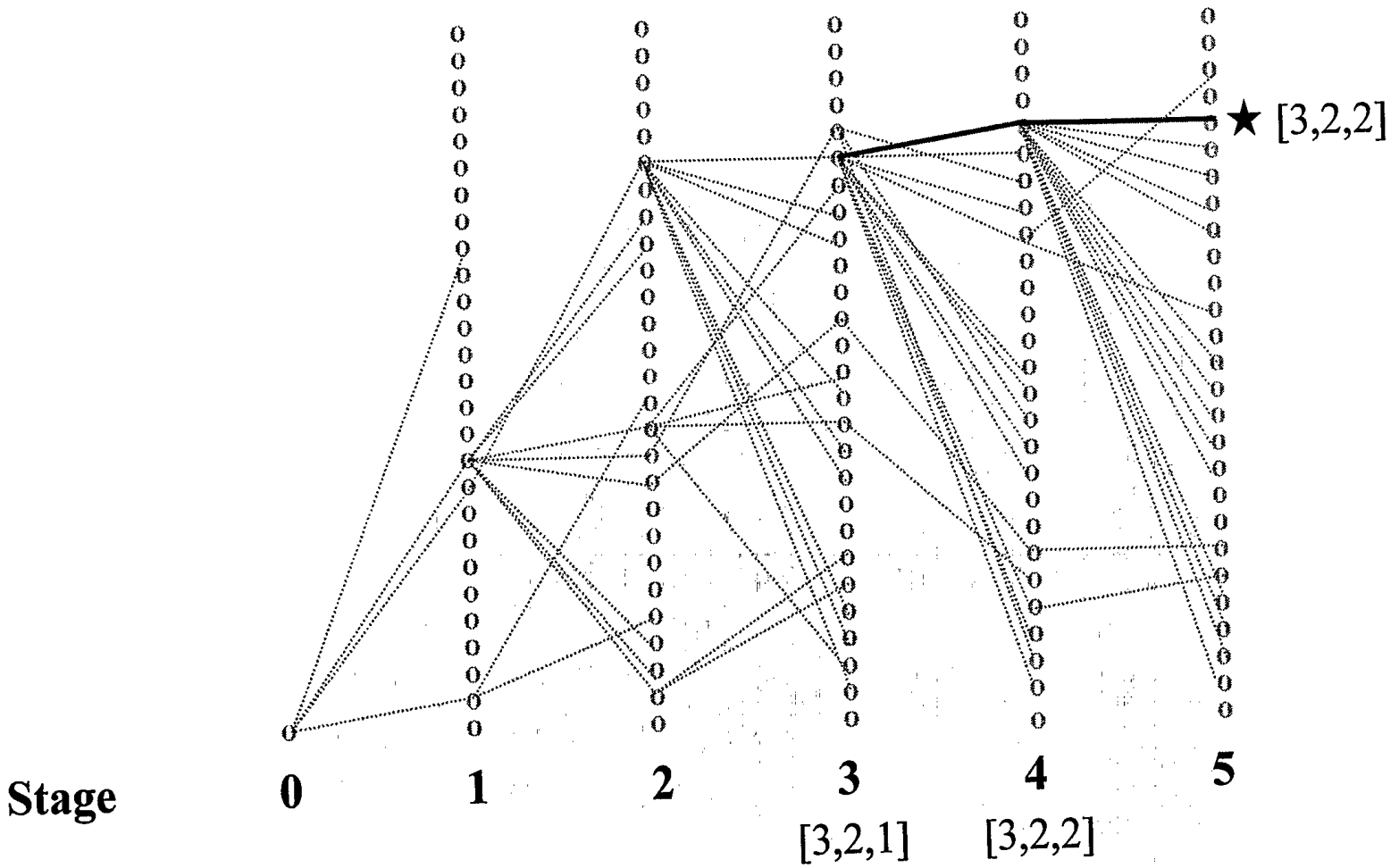


Figure A - 18 : Backtracking to Best Option for Stage Three

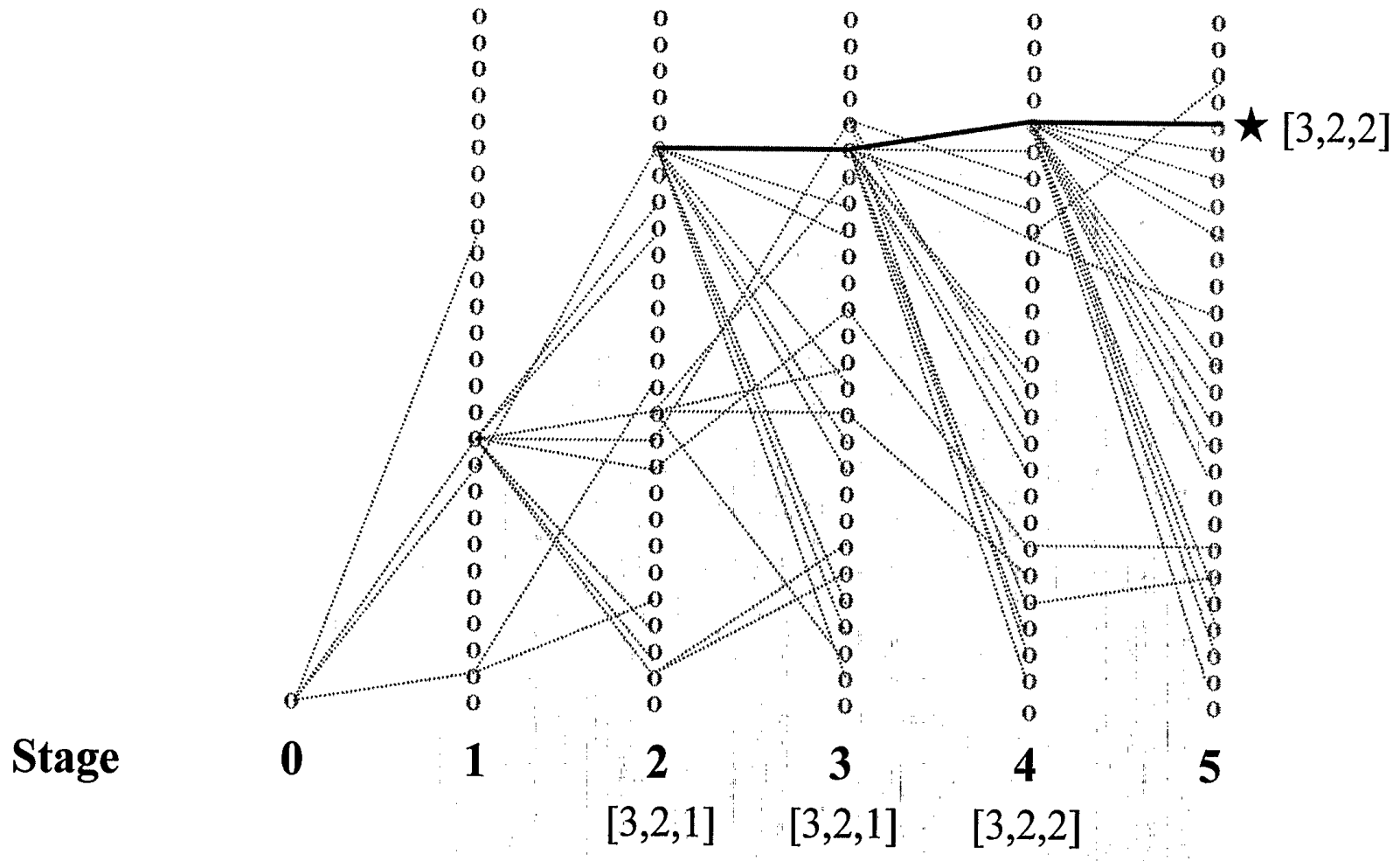


Figure A - 19: Backtracking to Stage Two Best Option

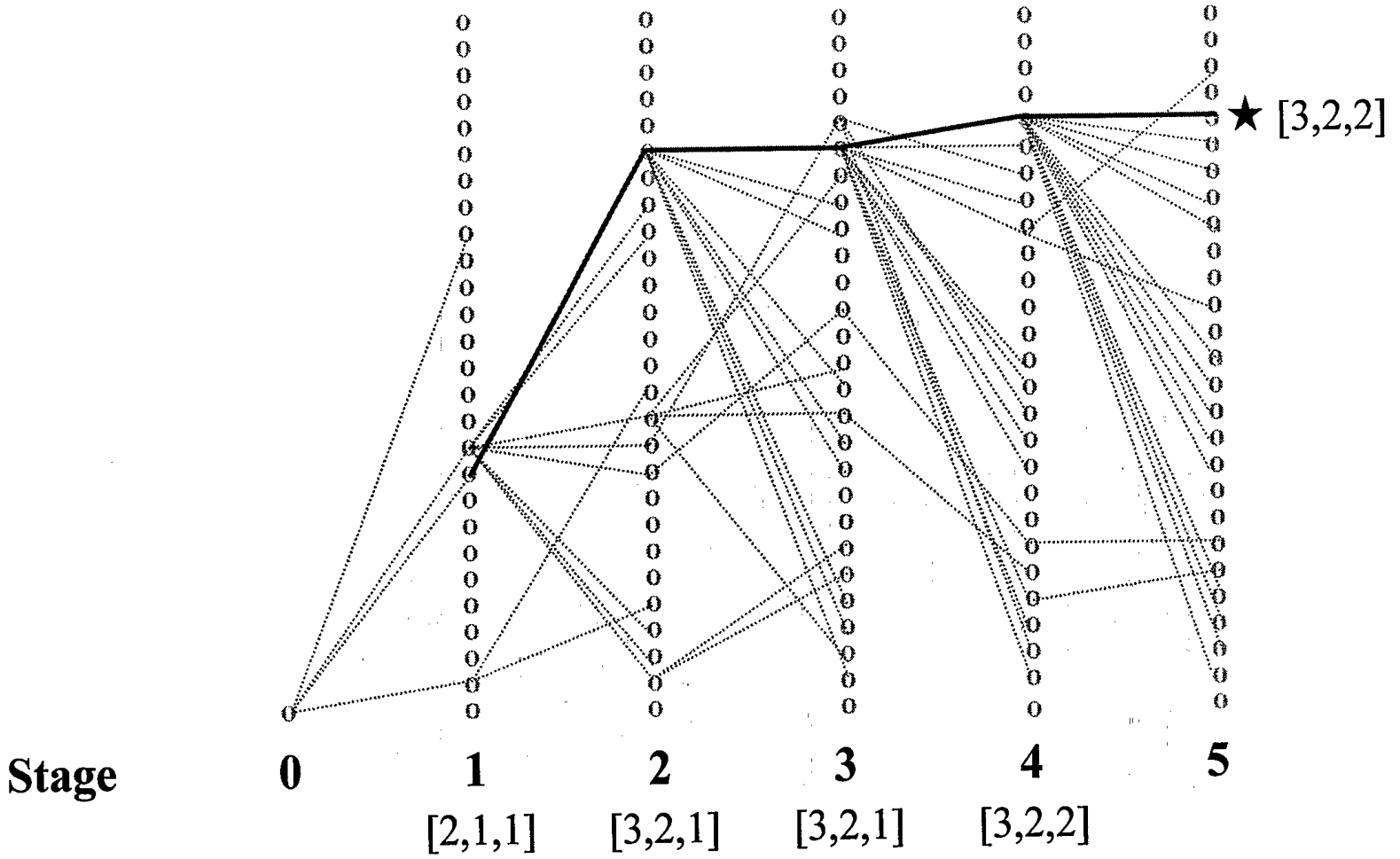


Figure A - 20: Backtracking to Stage One Best Option

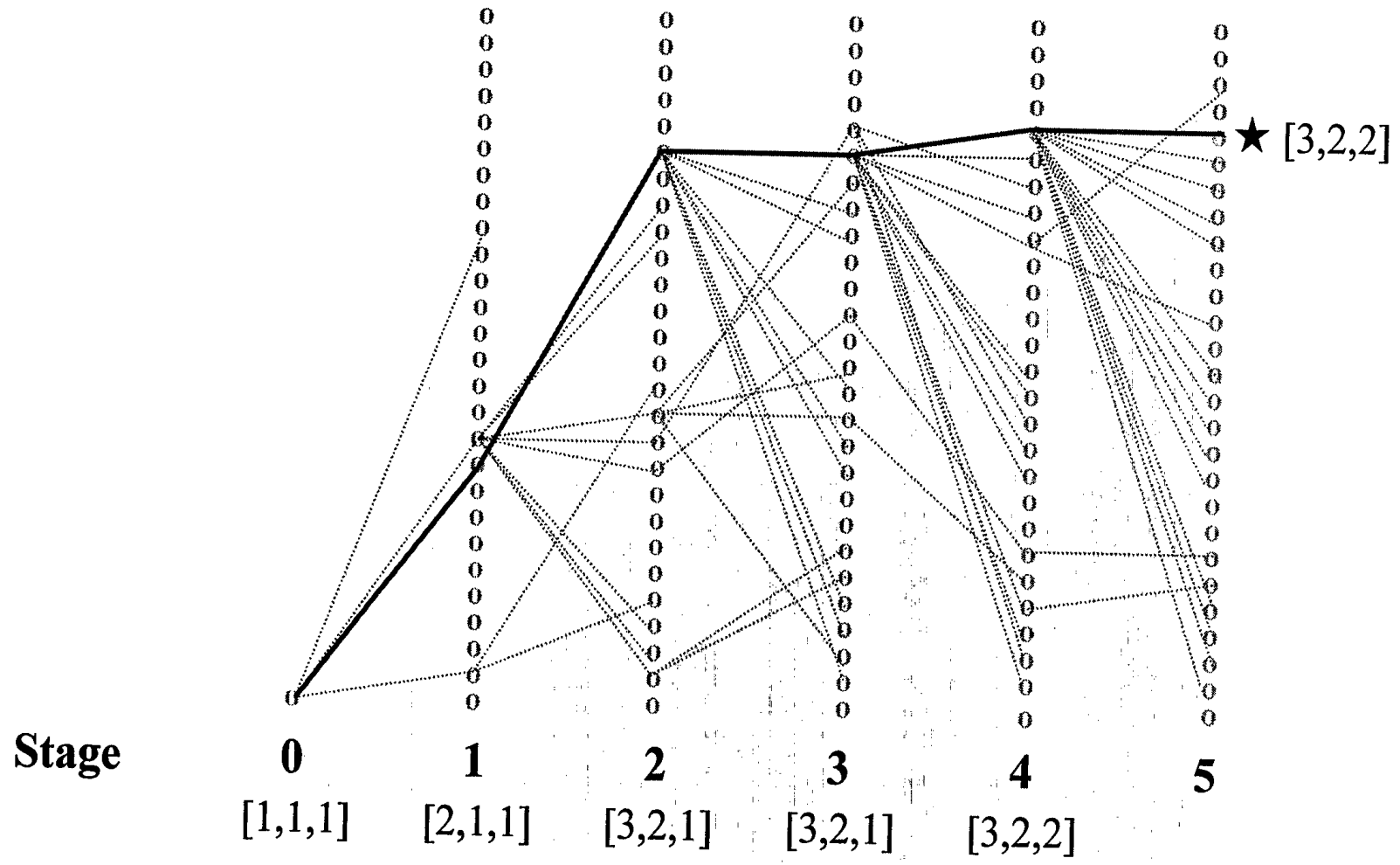


Figure A - 21: Complete Option Sequence Determined

Software Used in Developing the Sample Program

The software application designed to illustrate the Dynamic Programming concept was coded using Borland Delphi 3 Professional. Delphi is available from Inprise (formerly Borland), found on the World Wide Web at http://www.inprise.com/delphi/. The application runs in 32-bit Microsoft Windows applications. The application also incorporates the use of some add-on developer tools:

- Topaz 6.5 for Delphi – A dBase compatible add-on. It allows use of dBase databases without the significant extra overhead of the Borland Database Engine that comes with Delphi. Topaz is available from Software Science Inc. Its Web Site is: http://softsci.com/
- Crystal Reports 6.0 Professional – a reporting utility add-on. Crystal Reports allows flexible reporting of model input and output results which are stored in database format. It is available from Seagate Software. Information for Crystal Reports can be found on the Web at http://www.seagatesoftware.com/
- Chart FX 3.0 32 Bit – The full Chart FX3.0 is an upgrade to the version provided with Delphi, and is available from Software FX. This tool allows flexible creation of charts to help portray results of the program analysis. Information on the Web is available at: http://www.softwarefx.com/
- TParser Version 10.1 – a Delphi VCL control. TParser (originally written by Renate Schaaf and enhanced by Alin Flaider and Stefan Hoffmeister) is available on the Internet as a freeware Delphi control, with source code. This control is used in the sample program to allow entry of mathematical expressions at run-time, parse them, and calculate the result. This allows greater flexibility for the program, since certain problem characteristics can be specified "after-the-fact" as opposed to being "hard-wired" into the compiled program code. It is available from the Delphi Super Page on the Web at http://sunsite.icm.edu.pl/delphi/ (use the DSP search engine to search for TParser in order to find it).

Delphi source code for the *.pas files of the application, as well as pictures of the form files, are presented in another section within the Annex. The Delphi form files themselves (*.dfm), Crystal Reports report files (*.rpt) and the dBase input and output tables (*.dbf) are not reproduced, since they are not ASCII based formats. Interested parties can request copies of these files, if they have requirements related to Defence Purposes.

Description of the Program Code

The Concept Application program code is contained in a number of Delphi modules, some of which are also associated with user-interface forms.

The various modules or **units** are:

- **charts**, with form **frmCharts**,
- **DatMod1**, with form **DataModule1**,
- **input**, with form **inputForm**,
- **main**, with form **mainForm**,
- **option**,
- **report**, with form **frmReport**,
- **spacing**, with form **frmSpace**.

Unit charts

This unit provides a form and code for representing analysis results in chart form. This form is the one that contains the ChartFX charting control, named in the program as ChartFX1.

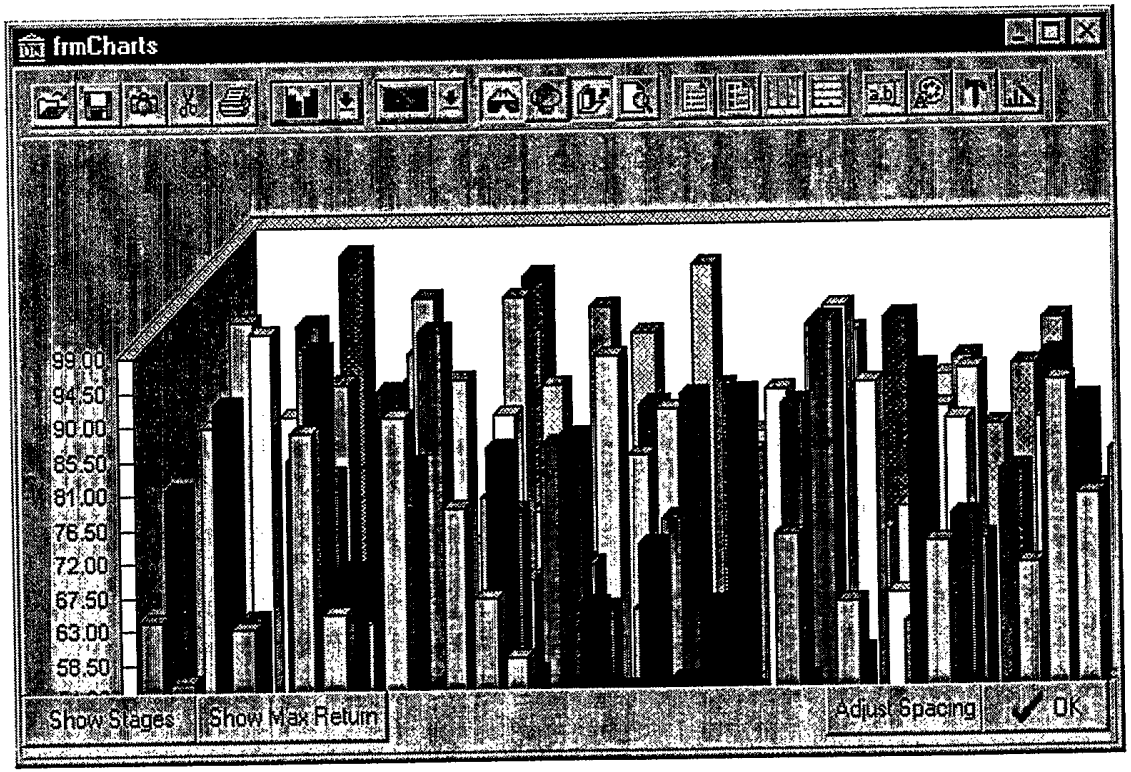


Figure A - 22 Charts Form

UNIT CHARTS CODE

```
unit charts;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, CfxVcl32, ExtCtrls, Buttons, spacing;
type
  TfrmCharts = class(TForm)
    Panel1: TPanel;
    Button1: TButton;
    Panel2: TPanel;
    ChartFX1: TChartFX;
    Button2: TButton;
    BitBtn1: TBitBtn;
    btnMaxReturn: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormResize(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure btnMaxReturnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmCharts: TfrmCharts;
  showAverage: boolean;

implementation
{$R *.DFM}
uses
   main, input;

procedure TfrmCharts.Button1Click(Sender: TObject);
var
   series, point, maxSoFar: integer;
begin
   if inputForm.rgDisplay.ItemIndex = 1
      then showAverage := true
      else showAverage := false;
   maxSoFar := 0;
   {numberOfOptions := 27;   //Intended to generalize later! }
```

```
ChartFX1.Title[CHART_TOPTIT] := 'OBJECTIVE VALUES by STAGE';
ChartFX1.Title[CHART_BOTTOMTIT] := 'Option Number';
ChartFX1.Title[CHART_LEFTTIT     ] := 'Value';
ChartFX1.NSeries := numberOfStages;
ChartFX1.NValues := numberOfOptions;
ChartFX1.OpenDataEx( COD_VALUES, numberOfStages, numberOfOptions );
for series := 0 to (numberOfStages - 1) do
begin
   ChartFX1.ThisSerie := series;
   for point := 0 to (numberOfOptions - 1) do
   begin
      if (stageOptions[ (series+1), (point+1) ] <> nil)
      then
      begin
         if showAverage
            then ChartFX1.Value[ point ] :=
                  (stageOptions[(series+1),(point+1)].CUM_EFF )/(series+1)
            else ChartFX1.Value[ point ] :=
                  stageOptions[ (series+1), (point+1) ].CUM_EFF;
         if ChartFX1.Value[point] > maxSoFar
            then maxSoFar := Round(ChartFX1.Value[point]+1);
      end
      else  ChartFX1.Value[ point ] := CHART_HIDDEN;
  end;
```

```
    end;
    ChartFX1.CloseData( COD_VALUES);
    if showAverage
       then ChartFX1.Adm[CSA_MAX] := 1.0
       else ChartFX1.Adm[CSA_MAX] := maxSoFar;

    for series := 0 to (numberOfStages - 1) do
    begin
       ChartFX1.SerLeg[ series ] := 'Stage '+IntToStr( series + 1);
    end;

    ChartFX1.Invalidate;
    ChartFX1.Visible := True;

end;

procedure TfrmCharts.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    ChartFX1.Visible := False;
end;

procedure TfrmCharts.FormResize(Sender: TObject);
begin
    ChartFX1.Top := Panel2.Top + 4;
```

```
   ChartFX1.Left := Panel2.Left + 4;
   ChartFX1.Height := Panel2.Height - 8;
   ChartFX1.Width := Panel2.Width - 8;
end;

procedure TfrmCharts.Button2Click(Sender: TObject);
begin
   frmSpace.ShowModal;
end;

procedure TfrmCharts.btnMaxReturnClick(Sender: TObject);
var
   index, index2, series, point, maxSoFar: integer;
   bestOptionID, bestPredecessorID, currentOption: integer;
   delimiter: string;
   maxReturn: single;
begin
   if inputForm.rgDisplay.ItemIndex = 1
      then showAverage := true
      else showAverage := false;

   ChartFX1.Title[CHART_TOPTIT] := 'OPTIMAL SELECTION by STAGE';
   ChartFX1.Title[CHART_BOTTOMTIT] := 'Option Number';
   ChartFX1.Title[CHART_LEFTTIT     ] := 'Value';
```

```
ChartFX1.NSeries := numberOfStages;
ChartFX1.NValues := numberOfOptions;
{Zero Out Array}
ChartFX1.OpenDataEx( COD_VALUES, numberOfStages, numberOfOptions );
for series := 0 to (numberOfStages - 1) do
begin
   ChartFX1.ThisSerie := series;
   for point := 0 to (numberOfOptions - 1) do
   begin
      if (stageOptions[ (series+1), (point+1) ] <> nil)
      then
      begin
         ChartFX1.Value[ point ] := 0;
      end
      else  ChartFX1.Value[ point ] := CHART_HIDDEN;
   end;
end;
ChartFX1.CloseData( COD_VALUES);
ChartFX1.Adm[CSA_MAX] := maxSoFar;
for series := 0 to (numberOfStages - 1) do
begin
   ChartFX1.SerLeg[ series ] := 'Stage '+IntToStr( series + 1 );
end;
```

```
maxReturn := 0;
{Find the Max Return Value in the last stage, and corresponding option}
for index := 1 to numberOfOptions do
begin
   if stageOptions[ numberOfStages, index ].CUM_EFF > maxReturn then
   begin
      maxReturn := stageOptions[ numberOfStages, index ].CUM_EFF;
      bestOptionID := index;
   end;
end;
ChartFX1.OpenDataEx( COD_VALUES, COD_UNCHANGE, COD_UNCHANGE);
ChartFX1.ThisSerie := (numberOfStages - 1);
if showAverage
   then ChartFX1.Value[ bestOptionID - 1] :=
       (stageOptions[ numberOfStages, bestOptionID ].CUM_EFF)/numberOfStages
   else ChartFX1.Value[ bestOptionID - 1] :=
        stageOptions[ numberOfStages, bestOptionID ].CUM_EFF;
ChartFX1.CloseData( COD_VALUES);

// Determine Best Predecessor of Final Stage Best Option:
currentOption := stageOptions[ numberOfStages, bestOptionID ].BST_PREDEC;

ChartFX1.OpenDataEx( COD_VALUES, COD_UNCHANGE, COD_UNCHANGE);
for index2 := (numberOfStages - 1) downto 1 do
```

```
  begin
     // Backtrack one level:
     currentOption := stageOptions[ index2 + 1, currentOption ].BST_PREDEC;
     ChartFX1.ThisSerie := (index2 - 1);
     if showAverage
     then ChartFX1.Value[ currentOption - 1] :=
          (stageOptions[ index2, currentOption ].CUM_EFF)/index2
     else ChartFX1.Value[ currentOption - 1] :=
          stageOptions[ index2, currentOption ].CUM_EFF;
  end;
  ChartFX1.CloseData( COD_VALUES);
  if showAverage
     then ChartFX1.Adm[CSA_MAX] := 1.0
     else ChartFX1.Adm[CSA_MAX] := maxReturn + 1;
  ChartFX1.Invalidate;
  ChartFX1.Visible := True;

end;

procedure TfrmCharts.FormCreate(Sender: TObject);
begin
   ChartFX1.StyleEx := CSE_LANDSCAPE;
   ChartFX1.TypeEx := ChartFX1.TypeEx or CTE_NOTITLESHADOW;
end;
```

```
procedure TfrmCharts.BitBtn1Click(Sender: TObject);
begin
   self.close;
end;

end.
```

Unit DatMod1

This unit contains holders for the Topaz database table components, which tie into the input and results database tables, and allow representation of the tables within Delphi code and on other application forms.

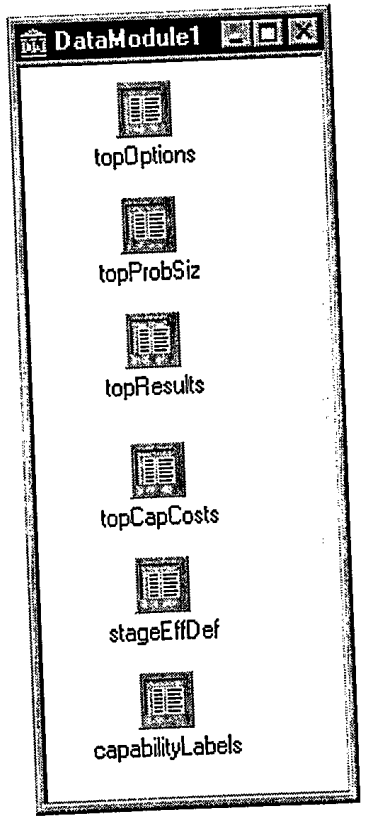


Figure A - 23 Data Module Form

UNIT DATMOD1 CODE

```
unit datMod1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  UCommon, UTzODbf, UTzDbf;

type
  TDataModule1 = class(TDataModule)
    topOptions: TTzDbf;
    topResults: TTzDbf;
    topProbSiz: TTzDbf;
    topCapCosts: TTzDbf;
    stageEffDef: TTzDbf;
    capabilityLabels: TTzDbf;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  DataModule1: TDataModule1;

implementation

{$R *.DFM}

end.
```

Unit input

The **input** unit provides code and a user-interface form **inputForm** for reviewing and editing the database tables which correspond to user input.

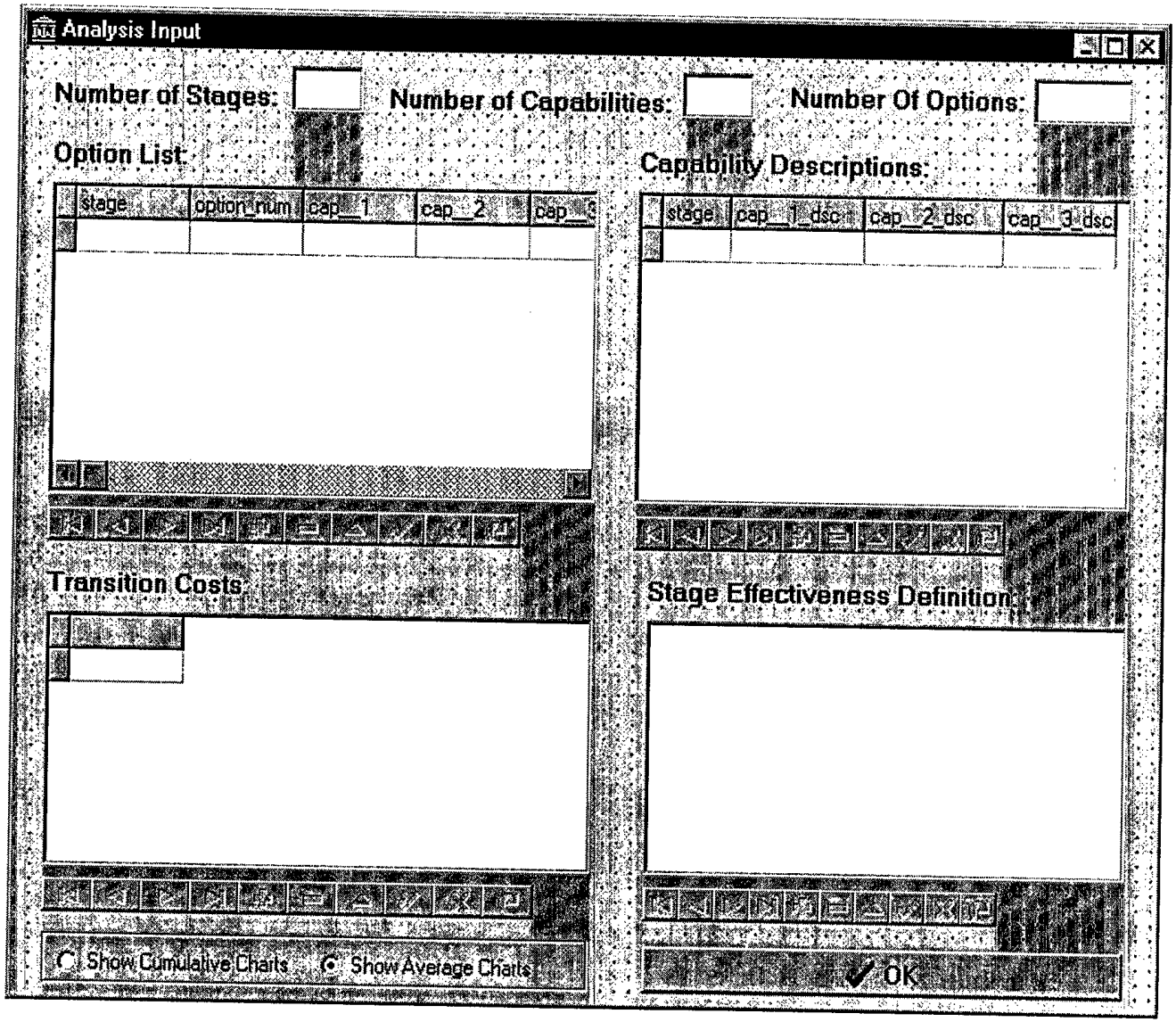


Figure A - 24 Input Form

UNIT INPUT CODE

```
unit input;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, Grids, ExtCtrls, Mask, aTzGrids,
  aTzNav, aTzMask, aTzEdit, aTzMemo;

type
  TinputForm = class(TForm)
    BitBtn1: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    TzNavigator1: TTzNavigator;
    TzGrid1: TTzGrid;
    TzEdit1: TTzEdit;
    TzEdit2: TTzEdit;
    TzNavigator2: TTzNavigator;
    TzGrid2: TTzGrid;
```

```
    TzEdit3: TTzEdit;
    TzEdit3Caption: TLabel;
    TzMemo1: TTzMemo;
    Label5: TLabel;
    TzNavigator3: TTzNavigator;
    TzGrid3: TTzGrid;
    TzNavigator4: TTzNavigator;
    rgDisplay: TRadioGroup;
    Label6: TLabel;
    procedure FormActivate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  inputForm: TinputForm;

implementation

uses datMod1;
```

```
{$R *.DFM}

procedure TinputForm.FormActivate(Sender: TObject);
begin
   DataModule1.topProbSiz.Active := true;
   DataModule1.topProbSiz.GoTop;
   DataModule1.topCapCosts.Active := true;
   DataModule1.topCapCosts.GoTop;
   DataModule1.stageEffDef.Active := true;
   DataModule1.stageEffDef.GoTop;
   DataModule1.capabilityLabels.Active := true;
   DataModule1.capabilityLabels.GoTop;
end;

procedure TinputForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
   DataModule1.topProbSiz.Active := false;
   DataModule1.topCapCosts.Active := false;
   DataModule1.stageEffDef.Active := false;
   DataModule1.capabilityLabels.Active := false;
end;

end.
```

Unit main

The **main** unit and its user-interface form **mainForm** provide the screen from which all other actions in the program derive from, and return to. This is the form on which the TParser (called TParser1) and Crystal Report (called CRPE1) components are placed.

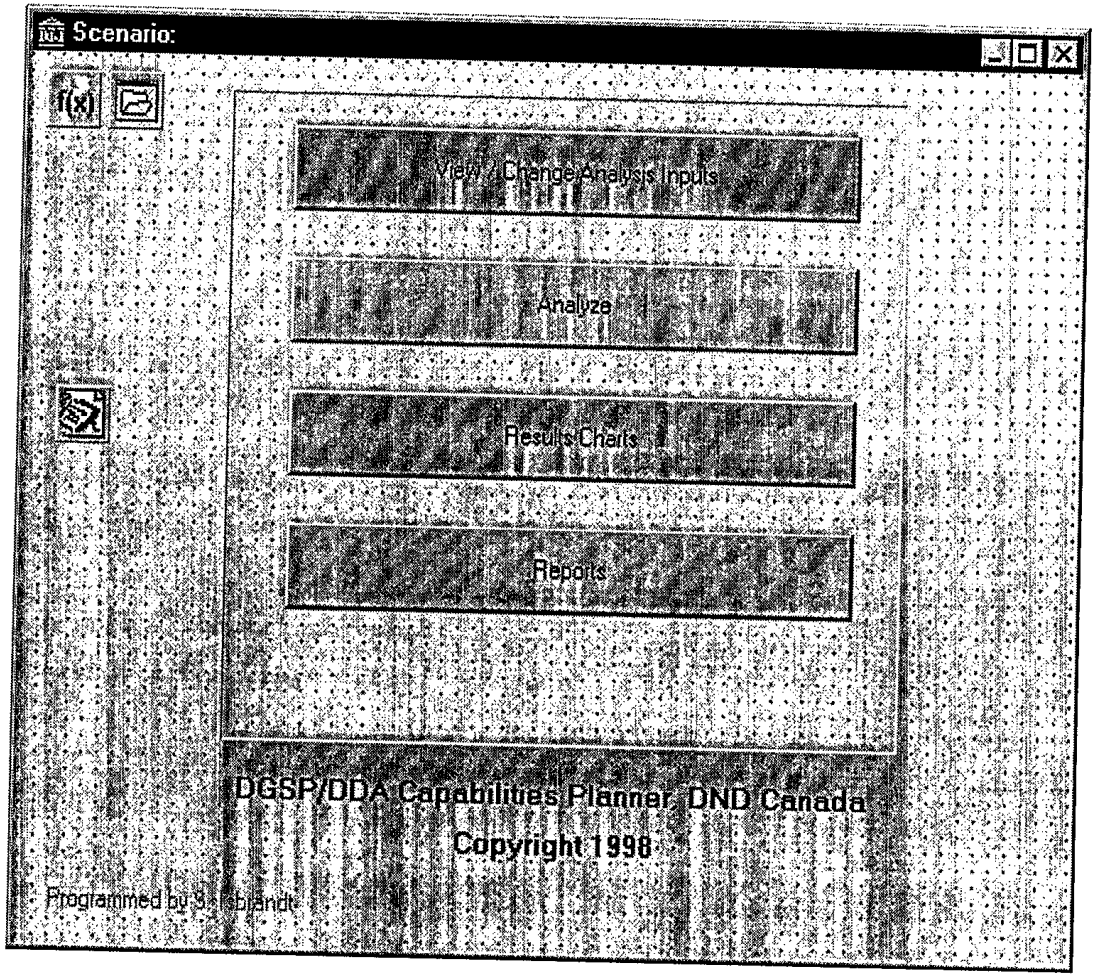


Figure A - 25 Main Form

UNIT MAIN CODE

```
unit main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, option, Parser10, UCRPE32, report, SplshDlg;

type
  TmainForm = class(TForm)
    Bevel1: TBevel;
    btnView: TButton;
    btnAnalyze: TButton;
    btnResults: TButton;
    btnReports: TButton;
    Parser1: TParser;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    OpenDialog1: TOpenDialog;
    Crpe1: TCrpe;
```

```
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    procedure btnViewClick(Sender: TObject);
    procedure btnAnalyzeClick(Sender: TObject);
    procedure btnResultsClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure btnReportsClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  mainForm: TmainForm;
  numberOfStages, numberOfOptions: integer;
  // Note that the stageOptions size MUST be matched in main.close event
  // cleanup of TOption instances.
  stageOptions: array[0..30, 1..1000] of TOption;
  connectDatabasesToggle: boolean;
  CapArray: array[1..1000] of string;
  FilesLocation, AppLocation: string;
```

```
implementation

uses input, datMod1, math, charts, IO_Form;

{$R *.DFM}

var

   minStageScore: single;
   stageEffectiveness, sampleEffectiveness, currentBestEffectiveness: single;
   numberOfCapabilities: integer;
   transitionCost: array[1..500, 1..10, 1..10] of integer;
   largeCostValue, cumulativeCost, costIncrement: Longint;
   feasiblePredecessorExists: boolean;

   prevCapabilityLevel, currentCapabilityLevel: array[1..500] of integer;

procedure analyze;
var
   index, stageIndex, optionIndex, previousStageOptionIndex: integer;
   expressionToEvaluate: TStringList;
   stringExpression: string;
begin
```

```
DataModule1.topProbSiz.Active := true;
numberOfStages := DataModule1.topProbSiz.GetLongField('num_stages');
DataModule1.topProbSiz.Active := false;

//Calculate dynamic program

DataModule1.stageEffDef.Active := true;

expressionToEvaluate := TStringList.create;
dataModule1.stageEffDef.DisableControls;
dataModule1.stageEffDef.GoTop;
dataModule1.stageEffDef.GetMField('DEFINE_EFF', expressionToEvaluate);
dataModule1.stageEffDef.EnableControls;

stringExpression := expressionToEvaluate[0];
if expressionToEvaluate.count > 1 then
   for index := 1 to (expressionToEvaluate.count - 1) do
   begin
      stringExpression := stringExpression + expressionToEvaluate[index];
   end;

mainForm.Label3.Caption := IntToStr( GetTickCount() );
mainForm.Parser1.Expression := stringExpression;
```

```
mainForm.Label4.Caption := IntToStr( GetTickCount() );
expressionToEvaluate.Free;


For stageIndex := 1 to numberOfStages do
begin

  for optionIndex := 1 to 1000 do
   if stageOptions[ stageIndex, optionIndex ] <> nil then
   begin

        feasiblePredecessorExists := false;

      (*  HARD-WIRED PARAMETERS FOR COST ANALYSIS CASE:
        //*********************************************
        // largeCostValue is set at 10 units per stage:
        largeCostValue := 10 * stageIndex;
        minStageScore := 8;
        //*********************************************    *)

      (*  HARD-WIRED PARAMETERS FOR RELIABILITY CASE:     *)
        //*********************************************
        // largeCostValue is set at 10 units per stage:
        largeCostValue := 10 * stageIndex;
```

```
minStageScore := 0.5;
//*********************************************

currentBestEffectiveness := 0;

mainForm.Parser1.Variable[ 'CAP1' ]
   := stageOptions[ stageIndex, optionIndex ].CAP_1;
mainForm.Parser1.Variable[ 'CAP2' ]
   := stageOptions[ stageIndex, optionIndex ].CAP_2;
mainForm.Parser1.Variable[ 'CAP3' ]
   := stageOptions[ stageIndex, optionIndex ].CAP_3;
stageEffectiveness := mainForm.Parser1.Value;
stageOptions[ stageIndex, optionIndex ].STAGE_EFF := stageEffectiveness;

for previousStageOptionIndex := 1 to 1000 do
begin
 if stageOptions[ stageIndex-1, previousStageOptionIndex ] <> nil then
  if (     (stageEffectiveness > minStageScore)
       and (stageOptions[ stageIndex-1, previousStageOptionIndex ]
             .Feasible = 'Y')
    )
  then
  begin
    //Look at possible transition from feasible predecessor
```

```
//USE largeCostValue to weed out infeasible transitions
sampleEffectiveness :=
   stageOptions[ stageIndex-1, previousStageOptionIndex ].CUM_EFF
   +  stageOptions[ stageIndex, optionIndex ].STAGE_EFF;
if (sampleEffectiveness > currentBestEffectiveness) then
begin
    prevCapabilityLevel[1]
      := stageOptions[ stageIndex-1, previousStageOptionIndex ].CAP__1;
    currentCapabilityLevel[1]
      := stageOptions[ stageIndex, optionIndex ].CAP__1;
    prevCapabilityLevel[2]
      := stageOptions[ stageIndex-1, previousStageOptionIndex ].CAP__2;
    currentCapabilityLevel[2]
      := stageOptions[ stageIndex, optionIndex ].CAP__2;
    prevCapabilityLevel[3]
      := stageOptions[ stageIndex-1, previousStageOptionIndex ].CAP__3;
    currentCapabilityLevel[3]
      := stageOptions[ stageIndex, optionIndex ].CAP__3;
    costIncrement :=
      transitionCost[ 1,
            prevCapabilityLevel[1], currentCapabilityLevel[1] ]
      + transitionCost[ 2,
            prevCapabilityLevel[2], currentCapabilityLevel[2] ]
      + transitionCost[ 3,
```

```
                    prevCapabilityLevel[3], currentCapabilityLevel[3] ];
                cumulativeCost :=
                  stageOptions[ stageIndex-1, previousStageOptionIndex ].CUM_CST
                  + costIncrement;
                if (cumulativeCost < largeCostValue) then
                begin
                  currentBestEffectiveness := sampleEffectiveness;
                  feasiblePredecessorExists := true;
                  stageOptions[ stageIndex, optionIndex ].CUM_EFF
                     := currentBestEffectiveness;
                  stageOptions[ stageIndex, optionIndex ].BST_CST_IN
                     := costIncrement;
                  stageOptions[ stageIndex, optionIndex ].CUM_CST
                     := cumulativeCost;
                  stageOptions[ stageIndex, optionIndex ].BST_PREDEC
                     := stageOptions[ stageIndex-1, previousStageOptionIndex ].OPTION_NUM;
                  stageOptions[ stageIndex, optionIndex ].FEASIBLE := 'Y';
                end;
              end;
            end;

        end;
    end;
end;
```

```
   DataModule1.stageEffDef.Active := false;
end;

procedure TmainForm.btnViewClick(Sender: TObject);
var
   capabilityIndex, index, index1, index2: integer;
   locationSpecified: boolean;
begin

   if ConnectDatabasesToggle then
   begin
      locationSpecified := false;
      try
         ShowMessage('Select a file location for the scenario to analyze.');
         while not locationSpecified do
         begin
            if OpenDialog1.Execute then
            begin
               FilesLocation := ExtractFilePath(OpenDialog1.FileName);
               locationSpecified := true;
               mainForm.Caption := mainForm.Caption + FilesLocation;
            end;
         end;
         DataModule1.topOptions.DbfFileName := FilesLocation + '\options.DBF';
```

```
        DataModule1.topProbSiz.DbfFileName := FilesLocation + '\prob_siz.DBF';
        DataModule1.topResults.DbfFileName := FilesLocation+ '\results.DBF';
        DataModule1.topCapCosts.DbfFileName := FilesLocation + '\cap_cost.DBF';
        DataModule1.stageEffDef.DbfFileName := FilesLocation + '\def_eff.DBF';
        DataModule1.capabilityLabels.DbfFileName := FilesLocation + '\cap_lab.DBF';
        ConnectDatabasesToggle := false;
    except
       ShowMessage('Unsuccessful at using input file location.');
    end;
end;

// Reset Feasibility Values on Input Table:
DataModule1.topOptions.disableControls; //Necessary to enable proper looping
DataModule1.topOptions.Active := true;
DataModule1.topOptions.GoTop;
while not DataModule1.topOptions.dEOF do
begin

   DataModule1.topOptions.SetCField('FEASIBLE','-');
   DataModule1.topOptions.SetRealField('STAGE_EFF', (-1) );
   DataModule1.topOptions.SetLongField('BST_PREDEC', ( -1) );
   DataModule1.topOptions.SetLongField('BST_CST_IN',(-1) );
   DataModule1.topOptions.SetRealField('CUM_EFF',(-1) );
   DataModule1.topOptions.SetLongField('CUM_CST',(-1) );
```

```
if (DataModule1.topOptions.GetLongField('STAGE') = 0 )
then
begin
    DataModule1.topOptions.SetCField('FEASIBLE','Y');
    DataModule1.topOptions.SetRealField('STAGE_EFF', 0 );
    DataModule1.topOptions.SetLongField('BST_PREDEC', 0 );
    DataModule1.topOptions.SetLongField('BST_CST_IN', 0);
    DataModule1.topOptions.SetRealField('CUM_EFF', 0);
    DataModule1.topOptions.SetLongField('CUM_CST', 0);
 end
 else
     DataModule1.topOptions.SetCField('FEASIBLE','N');
 DataModule1.topOptions.RLock;
 DataModule1.topOptions.Replace;
 DataModule1.topOptions.unLock;
 DataModule1.topOptions.skip(1);
 mainForm.Invalidate;
end;
DataModule1.topOptions.enableControls;    //Necessary to enable proper looping
DataModule1.topOptions.GoTop;

DataModule1.topCapCosts.Active := true;
DataModule1.topCapCosts.GoTop;
inputForm.showModal;
```

```
DataModule1.topProbSiz.Active := true;
numberOfCapabilities := DataModule1.topProbSiz.GetLongField('num_cap');
DataModule1.topProbSiz.Active := false;
// Read cost values into transitionCost matrix:
DataModule1.topCapCosts.Active := true;
DataModule1.topCapCosts.GoTop;

// Initialize cost matrix to zeroes:
for capabilityIndex := 1 to numberOfCapabilities do
   for index1 := 1 to 10 do                    // array dimensions should
      for index2 := 1 to 10 do                 // correspond to array declaration!!!
         transitionCost[ capabilityIndex, index1, index2 ] := 0;
// Read in database cost values:
while not DataModule1.topCapCosts.dEOF do
begin
   capabilityIndex := 0;
   if trim(DataModule1.topCapCosts.GetCField('CAPABILITY')) = 'CAP__1'
      then capabilityIndex := 1
      else
         if trim(DataModule1.topCapCosts.GetCField('CAPABILITY')) = 'CAP__2'
            then capabilityIndex := 2
            else
               if trim(DataModule1.topCapCosts.GetCField('CAPABILITY')) = 'CAP__3'
```

```
                then capabilityIndex := 3
                else ShowMessage('Capability Idenfication error: '
                   + DataModule1.topCapCosts.GetCField('CAPABILITY')+'!' );
    index1 := DataModule1.topCapCosts.GetLongField('FROM_LEVEL');
    index2 := DataModule1.topCapCosts.GetLongField('TO_LEVEL');
    transitionCost[ capabilityIndex, index1, index2 ] := DataModule1.
                              topCapCosts.GetLongField('COST');
    DataModule1.topCapCosts.Skip(1);
  end;

  //Setup variables for use internally in TParser
  if numberOfCapabilities > sizeOf( CapArray ) then
     ShowMessage('Too many capabilities specified.');
  for index := 1 to numberOfCapabilities do
  begin
     CapArray[ index ] := 'Cap' + IntToStr( index );
     //Initialize variables in Parser, set value to index
     Parser1.Variable[ capArray[index] ] :=  index;
   end;

   // Allow Analysis to Proceed
   btnAnalyze.enabled := true;
end;
```

```
procedure TmainForm.btnAnalyzeClick(Sender: TObject);
var
   stageIndex, optionIndex: integer;
   tempOption: TOption;
begin

   //Move options information to array of option objects (For Execution Speed
   //   when performing analysis.
   DataModule1.topOptions.Active := true;
   DataModule1.topOptions.GoTop;
   DataModule1.topOptions.disableControls;
   while (not DataModule1.topOptions.dEOF) do
   begin
     stageIndex := DataModule1.topOptions.GetLongField('STAGE');
     optionIndex := DataModule1.topOptions.GetLongField('OPTION_NUM');
     // If stageOptions[ , ] is not nil, FREE before using.
     if (stageOptions[  stageIndex, optionIndex  ] <> nil)
        then stageOptions[  stageIndex, optionIndex  ].Free;
     stageOptions[ stageIndex, optionIndex ] := TOption.Create;
     stageOptions[ stageIndex, optionIndex ].readFromOptionArray;
     DataModule1.topOptions.skip(1);
   end;
   DataModule1.topOptions.enableControls;
```

```
analyze;

try
    DataModule1.topResults.Active := true;
    DataModule1.topResults.disableControls;
    DataModule1.topResults.GoTop;
    while not DataModule1.topResults.dEOF   do
    begin
       //Lock record to delete
       DataModule1.topResults.RLock;
       DataModule1.topResults.DeleteRec;
       DataModule1.topResults.UnLock;
       DataModule1.topResults.Skip(1);
    end;
    DataModule1.topResults.pack;
    DataModule1.topResults.enableControls;
except
   //Do nothing
end;

DataModule1.topResults.disableControls;
if (DataModule1.topResults.RecCount  > 0)
    then ShowMessage('Results Table not cleared correctly');
For stageIndex := 1 to numberOfStages do
```

```
for optionIndex := 1 to 1000 do
begin
  // If stageOptions[ stageIndex, optionIndex ] <> nil then
  // copy the option to the results table!
  if stageOptions[ stageIndex, optionIndex ] <> nil then
  begin
     tempOption :=  stageOptions[ stageIndex, optionIndex ];
     // APPEND chronology is important to sequence right!!
     // CLEAR the record buffer, input new record, THEN append.
     DataModule1.topResults.clearRecord;
     DataModule1.topResults.SetLongField('STAGE',tempOption.STAGE);
     DataModule1.topResults.SetLongField('OPTION_NUM',tempOption.OPTION_NUM);
     DataModule1.topResults.SetLongField('CAP__1',tempOption.CAP__1);
     DataModule1.topResults.SetLongField('CAP__2',tempOption.CAP__2);
     DataModule1.topResults.SetLongField('CAP__3',tempOption.CAP__3);
     DataModule1.topResults.SetCField('FEASIBLE',tempOption.FEASIBLE);
     DataModule1.topResults.SetRealField('STAGE_EFF',tempOption.STAGE_EFF);
     DataModule1.topResults.SetLongField('BST_PREDEC',tempOption.BST_PREDEC);
     DataModule1.topResults.SetLongField('BST_CST_IN',tempOption.BST_CST_IN);
     DataModule1.topResults.SetRealField('CUM_EFF',tempOption.CUM_EFF);
     DataModule1.topResults.SetLongField('CUM_CST',tempOption.CUM_CST);
     // FILE MUST BE LOCKED TO APPEND
     DataModule1.topResults.ALock;
     DataModule1.topResults.Append;
```

```
            DataModule1.topResults.unLock;
          end;

        end;
        DataModule1.topResults.enableControls;
        btnResults.Enabled := true;
        btnReports.Enabled := true;
        DataModule1.topResults.Active := False;
end;

procedure TmainForm.btnResultsClick(Sender: TObject);
begin
   // Intended to generalize later!
   numberOfOptions := 27;
   frmCharts.Show;
end;

procedure TmainForm.FormClose(Sender: TObject; var Action: TCloseAction);
var
   index1, index2: integer;
begin
   DataModule1.topOptions.close;
   DataModule1.topResults.close;
   DataModule1.topProbSiz.close;
```

```
    DataModule1.topCapCosts.close;
    DataModule1.capabilityLabels.close;
    for index1 := 0 to 30 do
       for index2 := 1 to 1000 do
          if (stageOptions[  index1, index2  ] <> nil)
               then stageOptions[  index1, index2  ].Free;
end;

procedure TmainForm.FormCreate(Sender: TObject);
begin
   ConnectDatabasesToggle := true;
   AppLocation := ExtractFilePath(Application.ExeName);
end;

procedure TmainForm.btnReportsClick(Sender: TObject);
begin
   frmReport.ShowModal;
end;

end.
```

Unit option

This unit defined an "option" object which corresponds to one particular set of capabilities, which is a possible option at any given stage. In this concept version, the option is "hard-wired" at three capabilities, but the intention was to generalize to either a much larger or arbitrary number of capabilities in a later version of the application. This unit has no user-interface form associated with it.

UNIT OPTION CODE

```
unit option;

{$M+}  {Allow published vars - generate RTTI information}

interface

uses
   SysUtils;

type
  TOption = class(TObject)
  public
     STAGE: integer;
     OPTION_NUM: integer;
     CAP__1: integer;
     CAP__2: integer;
     CAP__3: integer;
```

```
      FEASIBLE: string;
      STAGE_EFF: single;
      BST_PREDEC: integer;
      BST_CST_IN: longInt;
      CUM_EFF: single;
      CUM_CST: longInt;
      constructor Create;
      procedure readFromOptionArray;
      procedure saveToResultArray;
      published

      end;



implementation

uses
   main, datMod1;

constructor TOption.Create;
begin
     inherited Create;
     STAGE := -1;
```

```
    OPTION_NUM := -1;
    CAP__1 := -1;
    CAP__2 := -1;
    CAP__3 := -1;
    FEASIBLE := '-';
    STAGE_EFF := -1;
    BST_PREDEC := -1;
    BST_CST_IN := -1;
    CUM_EFF := -1;
    CUM_CST := -1;
end;

procedure TOption.readFromOptionArray;
begin
   STAGE := DataModule1.topOptions.GetLongField('STAGE');
   OPTION_NUM := DataModule1.topOptions.GetLongField('OPTION_NUM');
   CAP__1 := DataModule1.topOptions.GetLongField('CAP__1');
   CAP__2 := DataModule1.topOptions.GetLongField('CAP__2');
   CAP__3 := DataModule1.topOptions.GetLongField('CAP__3');
   FEASIBLE := DataModule1.topOptions.GetCField('FEASIBLE');
   STAGE_EFF := DataModule1.topOptions.GetRealField('STAGE_EFF');
   BST_PREDEC := DataModule1.topOptions.GetLongField('BST_PREDEC');
   BST_CST_IN := DataModule1.topOptions.GetLongField('BST_CST_IN');
   CUM_EFF := DataModule1.topOptions.GetRealField('CUM_EFF');
```

```
    CUM_CST := DataModule1.topOptions.GetLongField('CUM_CST');
end;

procedure TOption.saveToResultArray;
begin
end;

end.
```

Unit report

Unit **report** and its form **frmReport** provide code and a user-interface for the presentation, printing, and file export of pre-designed input and results information. The reports use *.rpt files which are designed using Crystal Reports professional, and coordinated with the application to provide interactive reporting facilities.

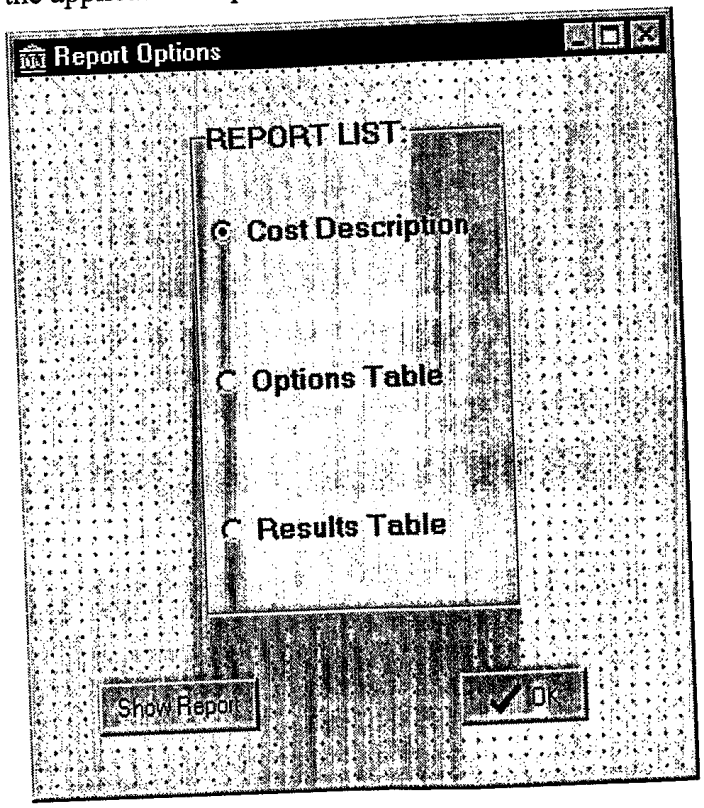


Figure A - 26 Report Form

UNIT REPORT CODE

```
unit report;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, StdCtrls, ExtCtrls;

type
  TfrmReport = class(TForm)
    RadioGroup1: TRadioGroup;
    Button1: TButton;
    BitBtn1: TBitBtn;
    procedure Button1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
  frmReport: TfrmReport;
  reportSubTitle: string;

implementation

uses main;

{$R *.DFM}

procedure TfrmReport.Button1Click(Sender: TObject);
var
   reportNumber: integer;
   reportFile, reportTitle, windowTitle: string;
   index: integer;
begin
   //'AppLocation' already set as: ExtractFilePath(Application.ExeName);
   reportNumber := RadioGroup1.itemIndex + 1;
   if reportNumber = 1 then
      begin
         reportTitle := 'Cost Report:  '+ reportSubTitle;
         reportFile := AppLocation + 'Cost.rpt';
         windowTitle := 'COST Report';
      end;
    if reportNumber = 2 then    begin
```

```
            reportTitle := 'Option Report:  '+ reportSubTitle;
            reportFile := AppLocation + 'Options.rpt';
            windowTitle := 'OPTIONS Description Report';
         end;
      if reportNumber = 3 then
         begin
            reportTitle := 'Results Report:  '+ reportSubTitle;
            reportFile := AppLocation + 'Results.rpt';
            windowTitle := 'Overall Results Report';
         end;

      // Set the report DataFilesLocation to the database FilesLocation.
      mainForm.CRPE1.DataFilesLocation := FilesLocation;

      mainForm.CRPE1.ReportName := reportFile;
      mainForm.CRPE1.ReportTitle := reportTitle;
      mainForm.CRPE1.WindowTitle := windowTitle;
      mainForm.CRPE1.Execute;
end;

procedure TfrmReport.FormActivate(Sender: TObject);
var
   description: string;
begin
```

```
   if reportSubTitle = ''
      then description := 'A Scenario'
      else description := reportSubTitle;
   reportSubTitle:= InputBox('Sub-Title',
         'Please provide a short report description', description);
end;

end.
```

Unit spacing

The **spacing** unit and its associated form **frmSpace** is a simple utility which coordinates with the charts interface form to allow the user to vary the width of bars in a bar charts.

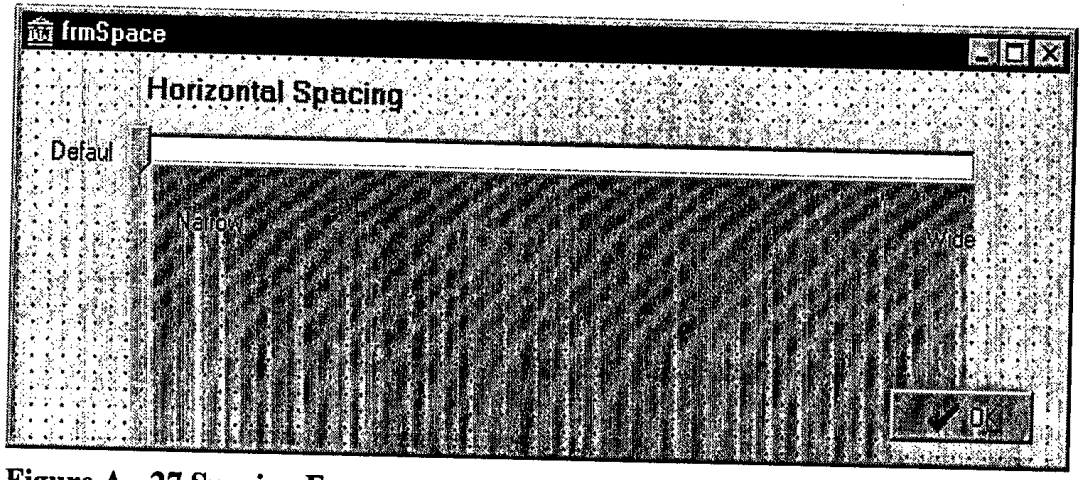


Figure A - 27 Spacing Form

UNIT SPACING CODE

```
unit spacing;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
```

```
  ComCtrls, StdCtrls, Buttons;

type
  TfrmSpace = class(TForm)
    TrackBar1: TTrackBar;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    BitBtn1: TBitBtn;
    procedure TrackBar1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmSpace: TfrmSpace;

implementation

uses charts;
```

```
{$R *.DFM}

procedure TfrmSpace.TrackBar1Change(Sender: TObject);
begin
   frmCharts.ChartFX1.fixedGap :=  5 * TrackBar1.Position;
   Label1.Caption := 'Gap is: '
      + intToStr(frmCharts.ChartFX1.fixedGap);
   frmCharts.invalidate;
end;

end.
```

Formulation of the Sample Problem

The sample problem can be formulated under the following headings:

- State space,
- Optimal value function,
- Policy reliability,
- Recurrence relationship,
- Boundary conditions,
- Statement of answer, and
- Optimal policy function.

State Space

At each stage, there are twenty-seven capability vector options ranging from [1,1,1] to [3,3,3] as described in the main paper.

For stage i and capability vector j where $1 \leq j \leq 27$, an option can be represented as :

$O_{ij} = [\, x_{ij1}, x_{ij2}, x_{ij3} \,]$

where x_{ij1}, x_{ij2}, and x_{ij3} are used to map the capability vectors. x_{ij1} represents the Medical Support capability level, x_{ij2} represents the Lift & Position capability level, and x_{ij3} represents the C&C (JFHQ) capability level. In the sample problem, for instance, $O_{11} = [\, 1, 1, 1]$.

This state space is restricted in this problem by cost constraints, which are applied in concert with the Dynamic Programming recurrence relationship, to ensure that any possible solution path satisfies the following:

$$(\text{Total Cost through Stage i}) \leq (i * 10)$$

The state space is further restricted with an objective value threshold constraint (see reliability below.)

Policy Reliability

The choice of a policy (or an option) O_{ij} gives a reliability g_{ij} where:

$$g_{ij} = (0.999 - ((3 - x_{ij1}) / 12)) * (0.999 - ((3 - x_{ij2}) / 6)) * (0.999 - ((3 - x_{ij3}) / 12))$$

A reliability threshold is applied in the sample problem. Whenever $g_{ij} < 0.5$ the corresponding option is considered to be unsuitable, and is not considered as part a potential solution.

Optimal value function

The optimal value function can be described as:

f_{ij} = maximum reliability obtainable by combining some assignment up to stage i -1 with the option O_{ij} in stage i, subject to cost constraints.

For results presentation, the maximum average reliability (f_i / i) based on the optimal value function is shown.

Recurrence Relationship

The state space is restricted when applying the recurrence relationship to recording only those options that meet both the cost constraint and the reliability threshold.

The cost of going from Option $O_{i-1,k}$ in stage i-1 to Option O_{ij} in stage i is defined as:

$$c(O_{i-1,k}, O_{ij})$$

and is supplied as an input value.

The recurrence relationship used to define the optimal overall policy assignment is:

$$f_{ij} = \max \{ g_{ij} + f_{i-1,k} \} \qquad \text{for } 1 \le j \le 27 \text{ and } 1 \le k \le 27$$

subject to

$$c(f_{ij}) \le (i * 10)$$

and

$$g_{ij} < 0.5$$

where the cost of f_{ij} is defined as:

$$c(f_{ij}) = c(O_{i-1,k}, O_{ij}) + c(f_{i-1,k})$$

[HOW IT'S DONE IN THE CODE: When examining options at a stage i, the optimal intermediate paths for all feasible options through stage i-1 are already determined. For each option j in stage i, each feasible option in stage i-1 is examined, and the one which is both feasible (i.e. it can be reached, the cost constraint is not violated, and the reliability threshold is met) and provides the maximum reliability is chosen for f_{ij}. A toggle switch is used for each option to record whether it is feasible or not as the algorithm proceeds. Also, the best predecessor is recorded for future reference as f_{ij} is determined. In the sample code, only one predecessor is recorded. A possible generalization to the code is, in the case of ties, to record a list of optimal predecessor options.]

Boundary conditions

Boundary conditions for applying the recurrence relationship are:

$f_{01} = 0$

$c(f_{01}) = 0$

The starting option at stage 0 is $O_{01} = [1, 1, 1]$

Statement of Answer

In the sample problem with five stages modelled, the answer is:

$f_{5*} = \max f_{5j}$ for $1 \leq j \leq 27$

giving f_{5*} as the optimal value of the objective function.

Optimal Policy Function

Having identified f_{5*}, the intermediate policies or options which provide the answer can be identified with the relationships:

Optimal policy or option at stage 5 is:

O_{5j} where j is the option associated with the optimal answer f_{5*}.

Given an identified optimal policy O_{ij} at stage i, $O_{i-1,k}$ is an optimal policy if:

$c(O_{i-1,k}, O_{ij}) + c(f_{i-1,k}) = c(f_{ij})$

and $O_{i-1,k}$ is a feasible option.

Backtracking from the stage 5 option with the relationship can identify intermediate policies or options

[HOW IT'S DONE IN THE CODE: Once the answer has been determined, it is straightforward to identify the optimal option at the final stage, and to backtrack using recorded predecessor information to the initial option. The list of options so identified is an optimal overall policy, given the assumptions, input data, and constraints of the sample problem. A possible generalization, if all optimal predecessors were recorded in the case of ties, would be to produce either all, or some representative subset, policies which are optimal. Another possible generalization would be to record even more detailed information along the way so that a number of the "close to optimal" policies could also be generated for analysis and discussion purposes.]

DOCUMENT CONTROL DATA
(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared e.g. Establishment Sponsoring a contractor's report, or tasking agency, are entered in Section 8).

Operational Research Division
Department of National Defence
Ottawa, Ontario K1A 0K2

2. SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title)

A DYNAMIC PROGRAMMING CONCEPT FOR STRATEGIC PLANNING (U)

4. AUTHORS (last name, first name, middle initial)

Mr. S. Isbrandt

5. DATE OF PUBLICATION (month Year of Publication of document)

NOVEMBER 1998

6a. NO OF PAGES (total containing information. Include Annexes, Appendices, etc.)

98

6b. NO OF REFS (total cited in document)

4

7. DESCRIPTIVE NOTES (the category of document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Directorate of Operational Research (Joint & Land) Research Note

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address).

DGSP/DDA National Defence Headquarters

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR's document number (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DOR(J&L) RN 9819

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors: further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the test is bilingual).

This concept paper explores how a Dynamic Programming methodology can contribute to a defence-oriented strategic planning analysis. A sample problem is described and solved using Dynamic Programming. Advantages and disadvantages of the methodology are presented. Implementation details and software code are included in an Annex

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified . If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

ANALYSIS
DYNAMIC PROGRAMMING
OPTIMIZATION
STRATEGIC PLANNING

Canada

#509725