Journal of the Korean
OR/MS Society
Vol. 11, No. 2, Oct. 1986

—Research Paper—

Optimization and Analysis of Nonserial Diverging Branch Systems in Dynamic Programming*

Chae Y. Lee**

Abstract

The focus of this paper is to develop the optimization procedures and analyze the complexities of the nonserial diverging branch systems in Dynamic Programming. The optimization procedure of the system is developed such that it helps to reduce the computational demands of the system. The complexity of the network is analyzed with the increasing number of nodes, branches and their connectedness to the main serial system. Determination of the optimal set of nodes for the main serial chain is also investigated.

1. Introduction

With the basic optimzation theory for the elementary nonserial systems by Aris et al. [1], the research on the nonserial network systems has attracted many researchers in the field of dynamic programming. Parker [8] considered production lines in a hardware manufacturing industry as a nonserial network system that contains branches and loops. Wong and Larson [11] examined the design and operation of natural gas transmission pipelines with a diverging branch system while Mays and Yen [6] applied the nonserial dynamic programming to a branched sewer system with converging branch structures. Optimization procedures and the associated complexity analyes for various types of loop structures are presented by Lee and Esogbue [5]. A dynamic programming approach to determine the optimum allocation of resources to the activities in a project network is due to Robinson [10]. Esogbue and Marks [2] studied several project scheduling and resource allocation problems of the CPM-cost variety in which the precedence relationships possess a nonserial structure.

Pope et al [9] presented a method for obtaining closed form solutions to nonserial dynamic programming problems with quadratic stage returns and linear transitions. They provided parametric solution tables both for the convex and nonconvex return functions and utilize these tables for the within stage optimizations.

In this paper, we present the optimization procedures and the complexity analysis for the multi-diverging branch systems. The branches are classified into levels. An algorithm that ef-

* This paper was partly supported by the Korean Science and Engineering Foundation.
** Dept. of Management Science, Korea Institute of Technolgy

fectively reduces the computational demand is developed. This paper is concluded with the determination of the optimal set of nodes that constitute the main serial chain. An illustrative example problem is also presented.

2. Optimization of the Single Diverging Branch System

We will first discuss about the optimization of a single diverging branch system. The computational demands of the procedure will also be analyzed. The examination of the multi-branch structures will then be followed.

A single diverging branch system is shown in Figure 1. The stage transformations and return functions for a main serial process (stage 1,..., s,...,N) and for a branch (stages 11,..., M1) are defined as follows (see Nemhauser [7]):

$$
\begin{aligned}
&x_{n-1} = t_n(x_n, d_n), \quad n = 1, \ldots, N \\
&r_n = r_n(x_n, d_n), \quad n = 1, \ldots, N \\
&x_{M1} = t_{s1}(x_s, d_s) \\
&x_{m-1,1} = t_{m1}(x_{m1}, d_{m1}), \quad m = 1, \ldots, M \\
&r_{m1} = r_{m1}(x_{m1}, d_1), \quad m = 1, \ldots, M
\end{aligned}
$$

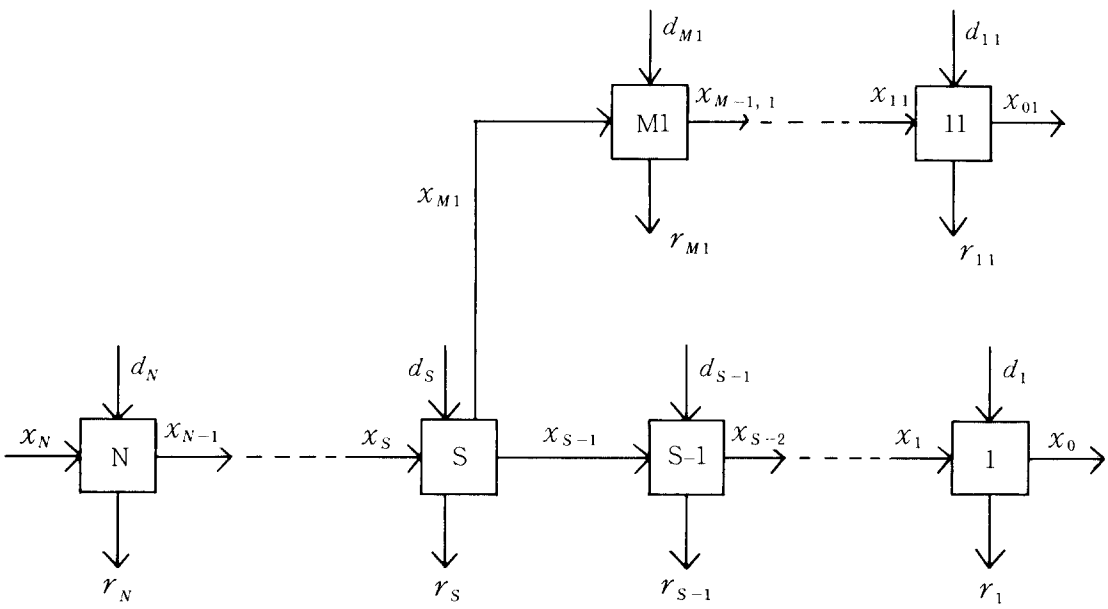

Figure 1. A Single Diverging Branch System

The optimization of the diverging branch system thus can be formulated as in the following mathematical form :

$$\max_{\substack{d_1,\ldots,d_N \\ d_{11},\ldots,d_{M1}}} \sum_{n=1}^{N} r_n(x_n,\ d_n) + \sum_{m=1}^{M} r_{m1}(x_{m1},\ d_{m1})$$

$$\text{s.t. } x_{n-1} = t_n(x_n,\ d_n) \qquad n = 1,\ldots,N$$
$$x_{m-1,\ 1} = t_{m1}(x_{m1},\ d_{m1}) \qquad m = 1,\ldots,M$$
$$x_{M1} = t_{s1}(x_s,\ d_s)$$
$$x_n \in X_n,\ x_{m1} \in x_{m1},\ d_n \in D_n,\ d_m \in D_{m1} \text{ for all } m,\ n$$

In the above formulation, x_n, x_{m1}, D_n, and D_{m1} are the constraint sets corresponding to the input and decision variables.

Note that by repeatedly substituting the input state variables x_{m1}, $m=1,\ldots,M$ using the transition functions, the decision vector$(d_{11},\ldots,d_{M1})$ in the diverging branch is determined independently of that in the main serial system. Therefore, the diverging branch can be optimized as a serial system yielding the optimal branch return $f_{M1}(x_{M1})$ with the optimal decisions d_{m1} (x_{m1}) for $m=1,\ldots,M$.

In the same manner, the decision vector$(d_1,\ldots,d_{s-1})$ of the main serial process is independent of that of the diverging branch. Thus, the main process can also be optimized as a serial system yielding $f_{s-1}(x_{s-1})$ at stage $s-1$ with $d_n(x_n)$ for $n=1,\ldots,s-1$. At stage s, the two branch returns $f_{M1}(x_{M1})$ and $f_{s-1}(x_{s-1})$ are combined with the stage return r_s. The optimization of the remaining stages are usual serial process.

Hence, the solution procedure for a single diverging branch system is given as follows :

1. Obtain the optimal branch return $f_{M1}(x_{M1})$ by serial procedures.
2. Obtain $(s-1)$ stage return $f_{s-1}(x_{s-1})$ by proceeding stages $1,\ldots,s-1$.
3. At stage s, combine the branch return with the return from the main chain as
   $f_s(x_s)=\max_{d_s}[r_s(x_s,\ d_s)+f_{s-1}(t_s(x_s,\ d_s)+f_{M1}(t_s,\ d_s))]$
4. Obtain the optimal system return $f_N(x_N)$ by proceeding stages $s+1,\ldots,N$.

The procedure repeatedly substitutes the stage return function with the one at the previous stage by employing the backward recursive equations. Computationally, it solves the recursive equation for each feasible state x_n by evaluating and selecting the decision that maximizes the computed return. In this manner the algorithm examines all the feasible states in each succeeding stage until the final stage is analyzed. The maximum cumulative total return among the feasible states in the terminal stage then gives the optimal system return. Thus, the diverging branch system is optimized with no more effort than the same sized serial system. The computational storage demand at each stage is one dimensional. For more detailed algorithm description see Lee [4].

3. Complexity Analysis of the Single Diverging Branch System

We examine the space and computational complexities of the single diverging branch algorithm

which is given in the previous section.

3.1 Space Complexity

To analyze the storage demand for the optimization of the single diverging branch system. We assume that the input and decision variables are integer-valued with the following lower and upper bounds:

$$L_n \leqq x_n \leqq U_n, \quad n=1,\ldots,N$$
$$L_{m1} \leqq x_{m1} \leqq U_{m1}, \quad m=1,\ldots,M.$$

We also define K_n and K_{m1} as follows:

$$K_n = U_n - L_n + 1, \quad K_{m1} = U_{m1} - L_{m1} + 1$$

Now, to simplify our notation in the complexity analysis we define the maximum discretization levels K and K_1 as

$$K = \max_n K_n, \qquad K_1 = \max_m K_{m1}.$$

Then, when we apply an exhaustive optimum searching method at each stage, the computational storage requirement of the single diverging branch system becomes

$$(M+2)K_1 + (N+2)K. \tag{1}$$

The above storage demand is itemized as follows:

1) To store the optimal decision for each input value at each of the $M+N$ stages the procedure requires $MK_1 + NK$ memory space.

2) The computation of the optimal branch return requires $2K_1$ memory space; Two storage spaces of size K_1 are repeatedly used in successive stages. The optimal $m1$-stage return f_{m1} at stage $m1$ will be stored at one memory space by adding the immediate return r_{m1} to the return $f_{m-1,1}$ which is stored at the other memory space.

3) To compute the optimal returns recursively at the main serial system the procedure needs $2K$ space.

If we assume $K_1 = K$, then the total storage requirement for this single diverging branch system becomes

$$(M+N+4)K. \tag{2}$$

3.2 Computational Complexity

The computational complexity is investigated by the number of elementary operations (addition and comparison) in the optimization procedure of the diverging branch system.

Notice that the optimization of the single diverging branch system has the same recursive equation as in the serial system except at the junction stage s. By assuming P and K discretizations of the decision and state variables respectively, the diverging branch algorithm needs PK comparisons at stages 11 and 1 of the system. No addition operations are performed at the stages. At stage s, two additions and one comparison are made for each discretized value of

x_s and d_s. Thus, stage s has $2PK$ additions and PK comparisons. All other stages have PK additions and PK comparisons. Therefore, the total number of operations required for the optimization of a diverging branch system is given as follows:

number of additions; $(N+M-1)PK$
number of comparisons; $(N+M)PK$
total number of operations; $(2(N+M)-1)PK$

This illustrates that the computational burden to solve a diverging branch system is linear to the number of stages in the main chain and the branch.

Table 1. Computational Demands of the Single Diverging Branch System— CPU Time in Seconds and Storage Words (numbers in the parentheses)

N=Number of stages in the main serial system	M=Number of stages in the diverging branch			
	3	8	13	18
5	· 176 (120)	· 214 (170)	· 250 (220)	· 291 (270)
10	· 237 (170)	· 272 (220)	· 316 (270)	· 350 (320)
15	· 308 (220)	· 339 (270)	· 381 (320)	· 420 (370)
20	· 361 (270)	· 398 (320)	· 438 (370)	· 482 (420)

Here, by implementing the optimization procedure into a FORTRAN code and running on the CYBER 170/855 at Georgia Institute of Technology, we illustrate the computational demands of the diverging branch system. The operating system was NOS, and the code was compiled using the FTN compiler in the OPT-2 optimizing mode. The code was run in BATCH mode from a time sharing terminal. Solution times were determined using the *CDC* timing routine SECOND. A set of problems with different number of stages both in the main serial process and in the diverging branch was solved. Table 1 shows the *CPU* time in second and the storage requirement (numbers in the parentheses) which is computed from the Equation (2) for each problem. In each case ten discretizations for all the input varibles were used.

For a fixed value of N, the *CPU* time increase is approximately linear to the number of stages in the diverging branch. The increase is also linear with respect to the number of stages in the main serial process for a fixed value of M.

4. Multi-Diverging Branch System

A multi-diverging branch system is shown in Figure 2. The system has D different diverging branches. Each branch has $M_k(k=1,..,D)$ stages and diverges from a stage s of the main serial chain. For the time being, to simplify our analysis we assume $s_k \neq s_{k'}$ for any two different branches k and k'.

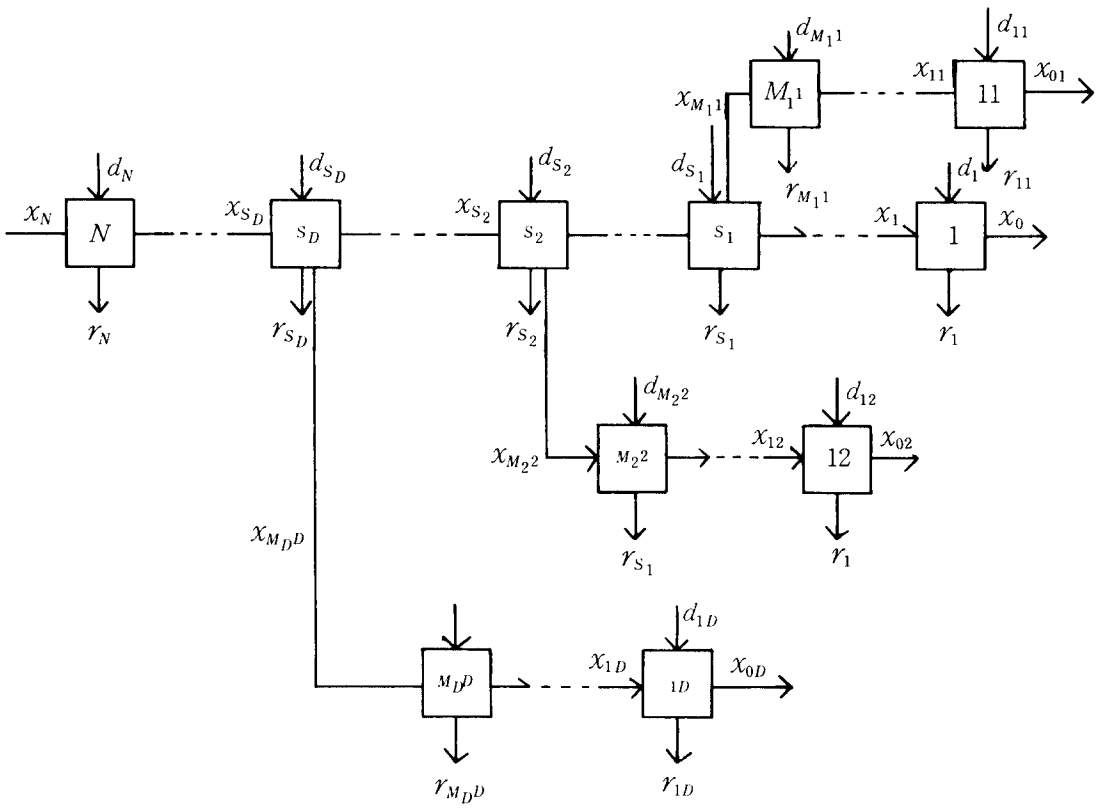

Figure 2. A Multi-diverging Branch System

As discussed in the single diverging branch system, the decisions at one diverging branch are independent of those in the main serial process and the other branches. As a consequence of this important fact, any multi-diverging branch system can be optimized as a one dimensional serial process (the optimal n-stage return function is represented with the input state variable x_n) regardless of the number of branches of the system. Hence, the optimization procedure for the single diverging branch system can be extended to this multidiverging branch structure.

However, the computational storage demand of this multidiverging branch system may be largely affected by the order of optimization. The importance of the order in which the subsystems should be optimized is well discussed in [3]. In relation to the multi-diverging branch system of Figure 2, Lee [4] considers the following two approaches.

The first approach optimizes the branches first with the optimal branch returns and decisions. Each branch return is then combined to the main serial process at the corresponding diverging stage s_k of the branch. In this case. due to the memory of the optimal return of each branch the storage demand increases with the number of branches in the system.

The second approach for the multi-diverging branch system optimizes each diverging branch and combines the optimal branch return when the optimization procedure for the main serial process has reached the corresponding diverging stage s_k of branch k. Thus, starting from the stage 1 of the main serial process, the optimization procedure continues as a serial process if a stage has no diverging branch. If a stage has a diverging branch, however, then the branch is optimized and the branch return is combined to the return from the main serial chain. This procedure effectively reduces the memory space required by the first approach to store the optimal return of each branch. When one branch return is combined at the corresponding diverging stage, the memory space used for the branch is successively used for the other branches.

However, a close examination reveals that the branches that do not diverge from the main serial system and that diverge from an identical node of the main trunk increase the complexity of a multi-diverging branch system. To generalize our analysis, we consider a more complex multi-diverging branch system as shown in Figure 3. We identify the branches as follows:

Branches of level 1: Branches that diverge from main serial chain (the branch of level 0).

Branches of level i: Branches that diverge from branches of level i-1.

In Figure 3, branches 1, 2 and 3 are classified as level 1, while branch 4 belongs to level 2.

Now, we consider the order of optimization. Clearly, any branch of each level can be treated as if it diverges from the main serial chain. In other words, when we optimize a branch of level i for $i=1,\ldots,L$, the branch of level $i-1$ is considered as the main serial chain. Hence, any complex multi-branch system can be examined as if it is a single level branch system (a system that has only branches of level 1).

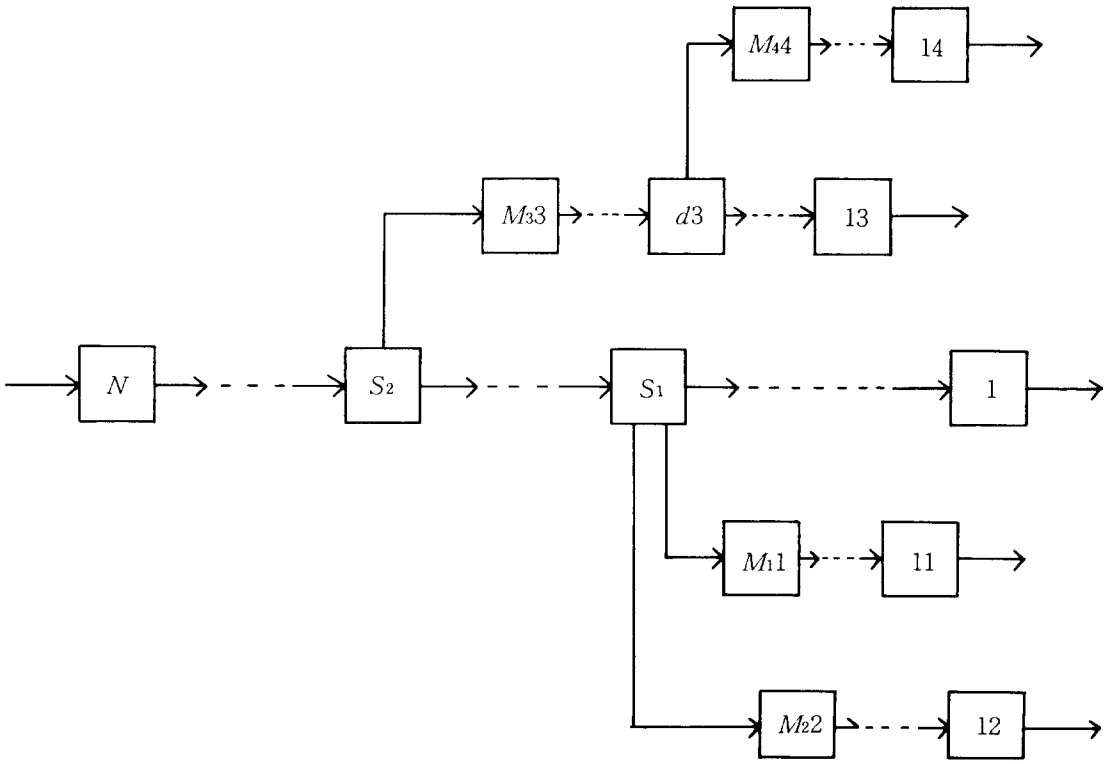

Figure 3. A Complex Multi-diverging Branch System

The optimization procedure starts from a branch of level L-1 that has branches of highest level L. If a branch of level L-1 has no diverging branches, then the branch is optimized when the procedure treats branches of level L-2. Otherwise, the branch of level L is combined into the branch of level L-1 as in the single diverging branch algorithm. This procedure continues until the optimization reaches the main serial chain of the system. The optimization algorithm for the complex multi-diverging branch system is now presented. The flowchart of the algorithm is shown in Figure 4.

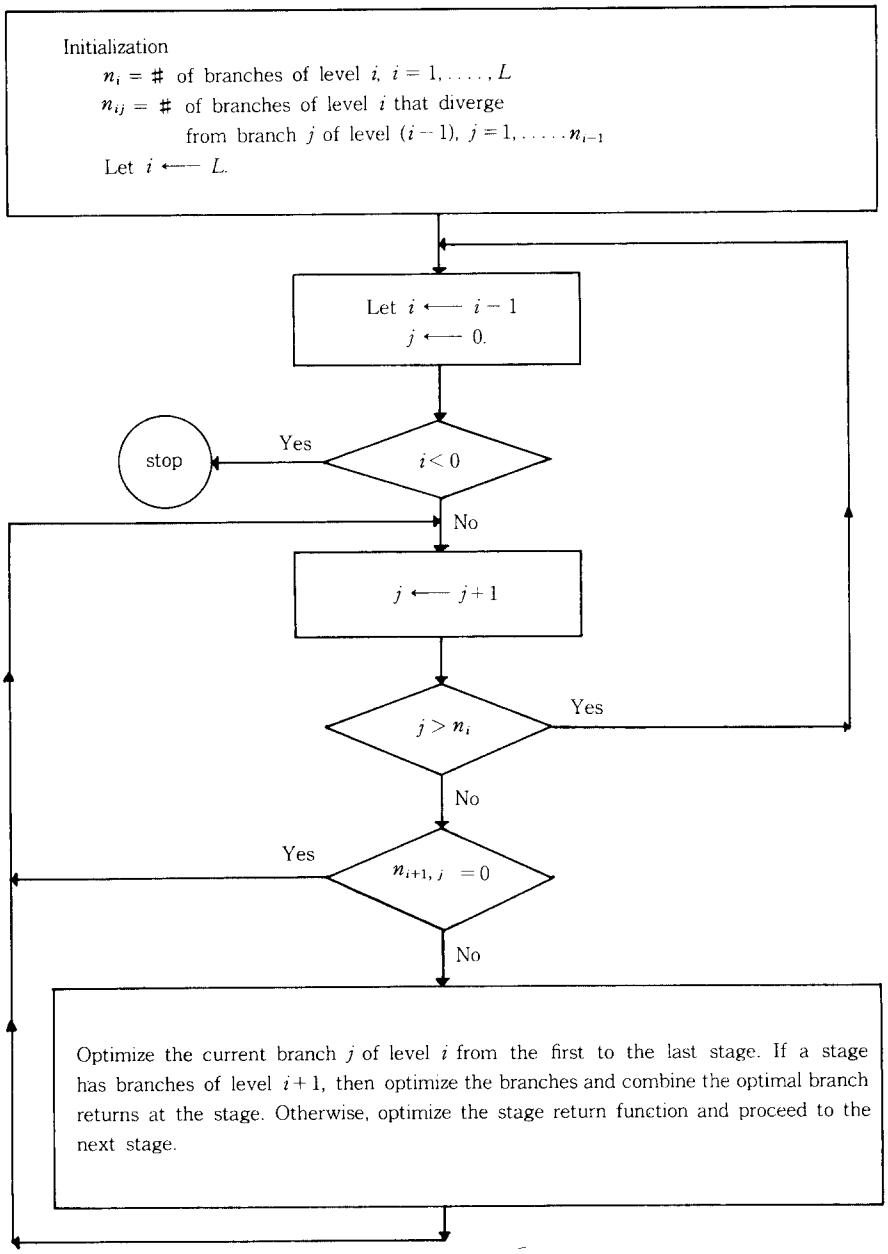

Figure 4. Flowchart of the Multi-diverging Branch Algorithm

Multi-Diverging Branch Algorithm

Initialization: Let n_i be the number of branches of level i for $i=1,\ldots,L$ and n_{ij} be the munber of branches of level i that diverge from branch j of level $i-1$, $j=1,\ldots,n_{i-1}$. Let $i=L$.

Step 1: Replace i with $i-1$ and let $j=0$. If $i<0$, then stop. The optimal system return is on hand. Otherwise, go to step 2.

Step 2: Replace j with $j+1$. If $j>n_i$, then go to step 1. Otherwise, If $n_{i+1,j}=0$ then repeat this step. If $n_{i+1,j}>0$ then go to step 3.

Step 3: Optimize the branch j of level i from the first stage to the last one. If a stage has branches of level $i+1$, then optimize the branches and combine the optimal branch returns at the stage. Otherwise, optimize the stage return function and proceed to the next stage. Go to step 2.

Consider a branch of level i that diverges from branch j of level $i-1$. If the branch of level i has no branches diverging from it, then the branch is optimized when the procedure reaches the diverging stage of the branch j. The optimal return of the branch of level i is absorbed directly into branch j of lower level. However, when many branches are diverging from an identical stage of branch j, then the optimal returns need to be stored before they are combined to the lower branch. Now, if the branch of level i has branches of level $i+1$ that diverge from it, then the combined branch return needs to be stored to be absorbed into the branch of level $i-1$. Thus, branches of level i that have branches of level $i+1$ and that diverge from an identical stage of the lower branch j require additional memory space (memory space required in addition to those for the optimal decisions and computation of the stage returns) to store the optimal branch return.

5. Sensitivity to the Complexity of Diverging Branches

When we apply the optimization procedure given in Section 4, a multi-diverging branch system calls for additional memory space.

We define

u_{ij} = number of branches of level i that diverge from branch j of level $i-1$ and that have branches of level $i+1$

v_{ij} = maximum mumber of branches of level i that diverge from an identical stage of branch j and that have no branches of level $i+1$.

Then the additional memory space to store the branch return in the multi-diverging branch system is determined by

$$Z=\max_{1\leq i\leq L}\ \{\ \max_{1\leq j\leq n_{i-1}}(u_{ij}+v_{ij}-1)\ \} \tag{3}$$

where L is the highest level of the branches in the system and n_{i-1} is the number of branches of level $i-1$. Since the main chain is considered as the only branch of level 0, we let $n_0=1$.

The additional memory space for branches of level i that diverge from branch j is given by

$u_{ij}+v_{ij}-1$. The last branch that is to be combined to an identical stage of the lower branch j does not need the additional space. It is absorbed directly into the lower branch. Also note that the memory space used at one level is also used for the other level. More specifically, the memory space used for branches of level i that diverge from branch j of level $i-1$ is successively used for branches of level i that diverge from another branch. Therefore, the maximum storage space required for branches of one level that diverge from a lower branch determines the additional memory space for the multi-diverging branch system. Hence, if we assume K as the maximum discretization of all the input variables, then the space complexity of the multi-diverging branch system is analyzed as follows:

1) To store the optimal decisions for K discretizations of the input variable at each stage, the procedure needs $K(\sum_{k=1}^{D} M_k + N)$ memory space.

2) The recursive computation of the stage returns at the main serial process requires $2K$ memory space.

3) Another $2K$ memory space is required to compute the optimal branch returns. This memory space is repeatedly used for all branches in a system by applying the optimization algorithm in Section 4.

4) Additional KZ memory space is required to store the optimal branch return.

Thus, the complexity of the multi-diverging branch algorithm is given by

$$K(\sum_{k=1}^{D} M_k + N + Z + 4). \tag{4}$$

In this study, the stages in the main serial process and the diverging branches are known before the optimization phase. In Section 6, we will discuss in detail the determination of the optimal set of nodes that contitute the main serial chain in a multi-diverging branch system.

6. Determination of the Best Main Serial Chain

In previous sections we have analyzed diverging branch systems, where the stages in the main serial chain as well as those in the branches are known before the optimization phase. However, when we deal with real world systems, it is probable that we need to decide the set of nodes (stages) that constitute the main serial system and branches. This calls for the segregation of a network system into subsystems such that it minimizes the computational storage demands for the optimization of the given network.

The optimization scheme for a multi-diverging branch system results in the following storage demand for a system with D different diverging branches:

$$K(\sum_{k=1}^{D} M_k + N + Z + 4) \tag{5}$$

with

$$Z = \max_{1 \le i \le L} \{ \max_{1 \le j \le n_{i-1}} (u_{ij} + v_{ij} - 1) \}, \tag{6}$$

where Z represents the maximum number of branches that requires one dimensional memory space to store the optimal branch return.

The above analysis is for a system where the stages of the main serial chain and the branches are known previously. We are now interested in determining the sets of nodes for the main trunk and branches such that the computational storage demand for the optimization of the system is minimized. Let N_l and Z_l be respectively the number of stages in the main serial system and the corresponding value of Z when branch ℓ is taken as the main serial process. Then Equation (5) becomes

$$K(\sum_{k=1,\, \neq l}^{D+1} M_k + N_l + Z_l + 4). \tag{7}$$

In Equation (7) the unknown main serial system is considered as another branch.

Definition. A set of nodes for the main serial system is said to be optimal, if it minimzes the storage demand for the optimization of a given nonserial network system.

Now, the following theorem holds to minimize the storage demand in the optimization of a multi-diverging branch system.

Theorem. Given a multi-diverging branch system with $D+1$ branches (including one main serial chain), if the input states of the system are discrete, then the optimal set of nodes for the main serial system is determined by a branch l^* such that

$$Z_{l^*} = \min_{l} [\max_{1 \leq i \leq L} \{\max_{1 \leq j \leq n_{i-1}} (u_{ij} + v_{ij} - 1)\}]. \tag{8}$$

Proof. Let the maximum discretization level of the input states be K, then the storage demand for the optimization of a multidiverging branch system with $D+1$ branches is given by Equation (7). In the equation, since the total number of stages, $\sum_{k=1,\, \neq l}^{D+1} M_k + N_l$, of the system is

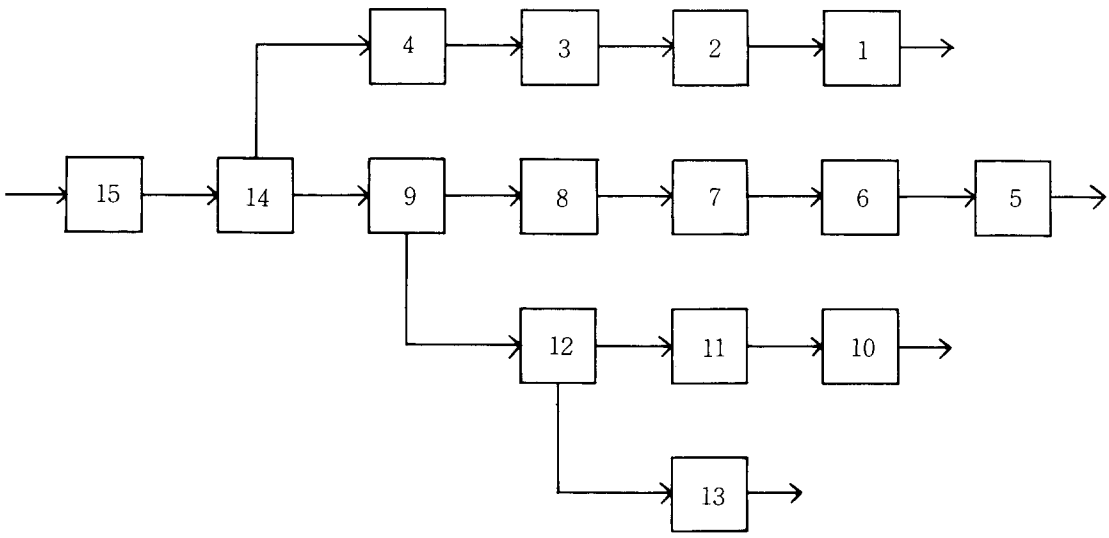

Figure 5. An Example of Multi-Diverging Branch System

fixed, the computational storge demand is dependent on Z_l. Therefore, from Equation (6), Equation (8) results. This completes the proof.

We will illustrate this theorem with the following example :

Example Consider a multi-diverging branch network given in Figure 5. The following four sets of nodes can be considered for the main serial system :

$$S_1 = \{1, 2, 3, 4, 14, 15\},$$
$$S_2 = \{5, 6, 7, 8, 9, 14, 15\},$$
$$S_3 = \{10, 11, 12, 9, 14, 15\},$$
and $$S_4 = \{13, 12, 9, 14, 15\}.$$

If we select S_2 as the main chain, then we have $u_{11}=1$ and $v_{11}=1$ which leads to $Z=1$. However, by taking S_3, we have $u_{11}=0$ and $v_{11}=1$ with $Z=0$. Table 2 gives the computation for each set of nodes for the main serial chain. Clearly, the optimal set of nodes for the main serial system is given by

$$S_3 = \{10, 11, 12, 9, 14, 15\}$$
or $$S_4 = \{13, 12, 9, 14, 15\}.$$

By taking S_3 or S_4 as the optimal set of nodes for the main serial system, we see that all the branches diverge from the main serial trunk.

Table 2. Determination of the Optimal Set of Nodes for the Main Serial System in the Multi-Diverging Branch System of Figure 5.

l	S_l	v_{ij}	u_{ij}	$Z_l = \max_{1\le i\le L} \{ \max_{1\le j\le n_{i-1}} (u_{ij}+v_{ij}-1) \}$
1	S_1	$u_{11}=1$	$v_{11}=1$	1
		$u_{11}=1$	$v_{21}=1$	
2	S_2	$u_{11}=1$	$v_{11}=1$	1
3	S_3	$u_{11}=0$	$v_{11}=1$	0*
4	S_4	$u_{11}=0$	$v_{11}=1$	0*

*represents the optimal case.

7. Conclusion

For the single diverging branch system, it is shown that the amount of memory space and the computing time increase linearly with respect to the number of stages either in the main chain or in the branch. For the multi-diverging branch system, an optimization procedure that effectively reduces the required storage demand is developed by classifying the branches into levels.

Finally, the sets of nodes that constitute the main serial chain and branches are examined based on the complexity analysis of the systems. The determination of the main serial system

is proved to be dependent on the number and the connectedness of the branches. To aid in the efficient analysis of more complex branch systems, one needs to consider the optimization of the multi-converging branch system. This issue is taken up in a forthcoming paper.

References

1. Aris, *R.*, G. L. Nemhauser and D. J. Wilde, "Optimization of Multistage Cycle and Branching Systems by Serial Procedures," *J. Am. Inst. Chem. Eng.*, Vol. 10, No. 6, 1964, pp. 913-919.
2. Esogbue, A. O. and Barry Marks, "Dynamic Programming Models of the Nonserial Critical Path-Cost Problem," *Management Science*, Vol. 24, No. 2, 1977, pp. 200-209.
3. Esogbue, A. O., "Dynamic Programming Algorithms and Analyses for Nonserial Networks," Technical Report, J-83-3, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Ga, 1983.
4. Lee, C. Y., "Analysis and Optimization of Complex Nonserial Dynamic Programming Network Systems," Ph. D. Dissertation. Georgia Institute of Technology, Atlanta, Georgia, 1985.
5. Lee, C. Y. and A. O. Esogbue, "Optimal Procedures for Dynamic Programs with Complex Loop Structures," *Journal of Mathematical Analysis and Applications*, Vol. 119, 1986, pp. 300-339.
6. Mays, L. W. and B. C. Yen, "Optimal Cost Design of Branched Sewer Systems," *Water Resources Research*. Vol. 11, 1975, pp. 37-47.
7. Nemhauser, G. L., *"Introduction to Dynamic Programming,"* Wiley, New York, 1976.
8. Parker, M. W., "Nonserial Multistage Systems-Analysis and Applications," Ph. D. Dissertation, University of Arkansas, 1969.
9. Pope, D. N., G. L. Curry and D. T. Phillips, "Closed Form Solutions to Nonserial, Nonconvex Quadratic Programming Problems Using Dynamic Programming." *Journal of Mathematical Analysis and Applications*, Vol. 86, 1982, pp. 628-647.
10. Robinson, D. R., "A Dynamic Programming Solution to Cost-Time Tradeoff for CPM," *Management Science*, Vol. 22, No 2, 1975, pp. 158-166
11. Wong, Peter and Robert Larson, "Optimization of Tree Structured Natural Gas Transmission Networks," *Journal of Mathematical Analysis and Applications*, Vol. 24, 1968, pp. 612-626.