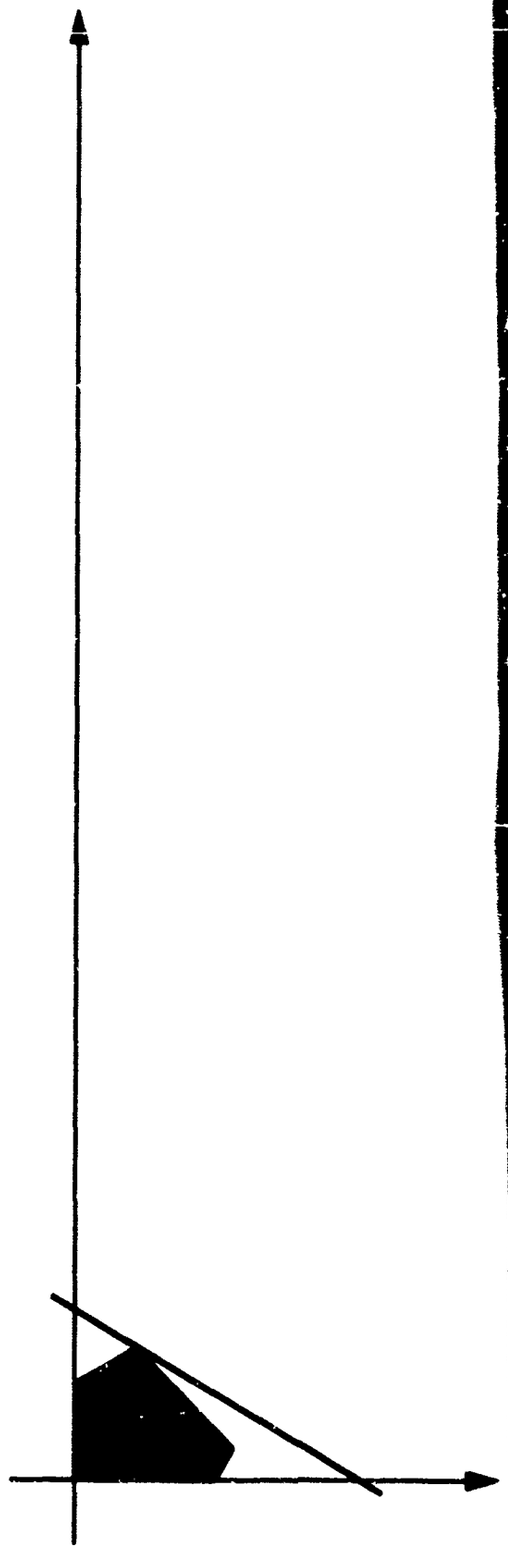
ORC 66-6
March 1966

OPTIMAL ASSIGNMENT OF COMPUTER STORAGE BY CHAIN DECOMPOSITION OF PARTIALLY ORDERED SETS

by

George B. Dantzig and Gary H. Reynolds



OPERATIONS RESEARCH CENTER

COLLEGE OF ENGINEERING

UNIVERSITY OF CALIFORNIA - BERKELEY

OPTIMAL ASSIGNMENT OF COMPUTER STORAGE BY CHAIN DECOMPOSITION OF PARTIALLY ORDERED SETS

by

George B. Dantzig and Gary H. Reynolds
Operations Research Center
University of California, Berkeley

March 1966 ORC 66-6

This research has been supported by the Office of Naval Research under Contract Nonr-222(83), and the National Institutes of Health under Grant GM-9606 and the National Science Foundation under Grant GP-4593 with the University of California. Reproduction in whole or in part is permitted for any purposes of the United States Government.

ABSTRACT

To save storage, a program is usually written so that each variable assumes several values. As a result, a program is usually difficult to understand and prone to errors. For an important class of programs, it will be shown that they can be written with complete freedom in the naming of variables; leaving the task of minimizing storage requirements to the computer itself.

Optimal Assignment of Computer Storage by Chain Decomposition of Partially Ordered Sets

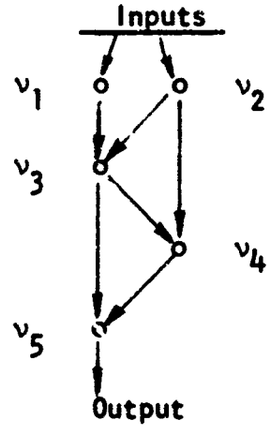
by

George B. Dantzig and Gary H. Reynolds

THE PROBLEM: Given a sequence of n steps, on the k-th step a value v_k is computed as a function F_k of the previously computed values $v_1, v_2, \ldots, v_{k-1}$. In general, only a subset of these is required to compute v_k. We are interested in finding the minimal number of locations in the memory of the computer to store the values v_i so that they will be available for computing the successive functions F_k. A saving in the number of locations occurs whenever a value v_k is stored in the same location as a previously computed v_i, which is no longer needed for step $k+1, k+2, \ldots, n$.

MOTIVATION: Consider the following trivial example:

Step	
1.	v_1 = given
2.	v_2 = given
3.	$v_3 = v_1 + v_2$
4.	$v_4 = v_2 + v_3$
5.	$v_5 = v_3 + v_4$
6. Output	$= v_5$



The arrows in the figure indicate the node values needed to compute $v_1, v_2, \ldots$. Thus v_3 requires v_1 and v_2, etc. The logical dependence is, accordingly,

$$\nu_1 = F_1(\text{constant given})$$

$$\nu_2 = F_2(\text{constant given})$$

$$\nu_3 = F_3(\nu_1, \nu_2)$$

$$\nu_4 = F_4(\nu_2, \nu_3)$$

$$\nu_5 = F_5(\nu_3, \nu_4)$$

$$\text{Output} = F_6(\nu_5) \quad .$$

A programmer who is a memory miser would notice the dependence of one step on another and would store ν_3 in the same location as ν_1 (since ν_1 is not needed after step 3), ν_4 in the same location as ν_2, and ν_5 in the same location as ν_3. Only two locations are needed which he calls, say, "x" and "y". He accordingly writes the following program:

Step	
1.	x = given input
2.	y = given input
3.	x = x+y
4.	y = y+x
5.	x = x+y
6.	output x

or some such nonsense, which we will refer to as "memory misering algebra."

This multiple use of the same symbol is a recognized cause of program error. It is one of the primary reasons why one programmer has the greatest difficulty in understanding a program written by another (or even one by himself). To avoid multiple use of the same symbol for memory misering, a programmer can make use of special instructions which will direct the machine language compiler to store the values of different symbols in the same location. This is of some help, but leaves the task of conserving storage location up to the programmer and again is subject to

error.

Our thesis is that memory misering is essentially clerical in nature, a task unworthy of the programmer's time. We will show for one important class of programs that the task of conservation of memory location can be done efficiently by the machine as part of its translation of a program into machine language.

SOLUTION: Define for each v_k an interval of storage. If v_k is last needed to compute F_ℓ, then its interval of time for storage is from step k+1 to $\ell_k = \ell$ and is denoted by

$$I(v_k) = [k+1, \ell_k] \quad .$$

We define an interval $I(v_k)$ as coming before another interval $I(v_{k'})$ when $\ell_k < k'+1$, which we write as

$$I(v_i) \prec I(v_{k'}) \quad \text{if} \quad \ell_k < k'+1 \quad .$$

The set of intervals forms a partially ordered set under this ordering relation. It is obviously transitive. No ordering is given between two overlapping intervals; such intervals are said to be unrelated. A subset of intervals $I(v_{j_1})$, $I(v_{j_2})$,..., $I(v_{j_s})$ is said to be completely ordered if

$$I(v_{j_1}) \prec I(v_{j_2}) \prec \ldots \prec I(v_{j_s}) \quad .$$

We will refer to such a completely ordered subset as a chain. Obviously, values $v_{j_1}, v_{j_2}, \ldots, v_{j_s}$ associated with the intervals in a chain may all be stored in the same storage location.

The problem of finding the minimal number of storage locations is thus the same as that of decomposing a partially ordered set into disjoint subsets, each of which is completely ordered. This is called a chain decomposition. A constructive procedure for doing this is given by one of the authors, joint with Alan Hoffman [1],

in connection with Dilworth's Theorem [2]. In our special application here to the partially ordered set of intervals, there is available, however, a much easier procedure. This can be found in Ford and Fulkerson [3]. Applied here, it yields:

RULE: Store v_k in the same location as any v_i not needed for any step after k.

It is obvious that the application of the rule provides a valid storage procedure and it is probably equally obvious that the rule yields a minimal number of storage locations. We will, nevertheless, give a formal proof.

Up to step k, let T_{k-1} be the subset of locations used to store the values $v_1, v_2, \ldots, v_{k-1}$. Let L be any location in the set T_{k-1}, and v_i the last value stored in L at the start of step k. Several values may have previously been stored in L, but v_i refers only to the last one stored in L up to step k; let I_L be the storage interval of this v_i.

It is clear that v_k cannot be stored in L if I_L overlaps with $I(v_k)$. If $I(v_k)$ overlaps with every interval I_L for all $L \epsilon T_{k-1}$, then it is necessary to increase the set of storage locations in order to store v_k. In this case, the number of storage locations in T_k has to be one greater than N_{k-1}, the number of locations in T_{k-1}. In general, $N_k = 1 + N_{k-1}$ or $N_k = N_{k-1}$. Let us suppose that on step k, there was a location $L \epsilon T_{k-1}$ such that $I(L)$ does not overlap with $I(v_k)$, but that a location $\bar{L}$ not in T_{k-1} was used instead for storing v_k. Note that on subsequent steps the values stored in L or $\bar{L}$ could be interchanged if on step k, location L were used in place of $\bar{L}$. This interchange never increases the count of the locations used and the count could even be decreased if $\bar{L}$ is never used and is dropped.

Thus we have shown that there always exists a minimal storage selection that always stores for each k the value v_k in T_{k-1} unless I_L for all $L \epsilon T_{k-1}$

overlaps with $I(\nu_k)$. We wish to show that any selection with this property is minimal. Let $k = k^*$ be the lowest index k such that $N_{k^*} = \text{Max } N_k$; then $1 + N_{k^*-1} = N_{k^*}$. Thus every interval I_L for LeT_{k^*-1} overlaps with that of $I(\nu_{k^*})$. But each such interval begins before $I(\nu_{k^*})$, hence all overlap with the value $k^* + 1$, the start of interval $I(\nu_{k^*})$. Thus all N_{k^*} intervals of T_{k^*} have the value $k^* + 1$ in common, and constitute a set of N_{k^*} unrelated intervals in the partially ordered set of intervals.

Note that N_{k^*} happens to be also equal to the number of storage locations selected to carry out the computations. Associated with each location LeT_{k^*} is the subset of values ν_i stored in L on steps $1,2,\ldots,n$. The intervals $I(\nu_i)$ of these ν_i are completely ordered, hence form a chain.

Thus for each LeT_{k^*} , there is associated a mutually exclusive chain, and every interval in the original partially ordered set belongs to one of these chains. Thus we can decompose the partially ordered set into N_{k^*} non-overlapping chains. Since it is obvious that each member of any group of unrelated elements must belong to different chains. the minimum chain-decomposition must always be greater or equal to the maximum number of unrelated elements. Hence, when N_{k^*} , the number of chains in some decomposition, happens to be the same as the number of elements in some set of unrelated elements, we conclude that this can only occur when the partially ordered set has been decomposed into a minimal number of chains. This completes our proof. The discussion just given is a paraphrase of the usual proof of sufficiency of the following:

DILWORTH'S THEOREM: The maximum number of unrelated elements in a partially ordered set is equal to the number of chains in a minimal decomposition.

APPLICATION IF THE NUMBER OF STEPS IS SMALL: The task of the compiler will be to set up a correspondence between location addresses and symbols used in the program.

If there are n steps and n is reasonably small, then the following procedure will accomplish the minimum storage of the program. Only if the program is to be executed many times would the method to save storage given below be worthwhile.

Set aside n locations A_k for recording ℓ_k, the last step for which v_k is needed for computation. Scan each step ℓ in turn and record ℓ in A_k if v_k is required on step ℓ to compute v_ℓ. The final value of ℓ recorded in each A_k is ℓ_k. Note that for any v_i which is not required on some subsequent step (such as v_n) the value in A_i is $\ell_i = 0$.

Set up a way of generating the names of up to $n-1$ addresses which will be called upon as required as a source of additional addresses for storing v_i. The addresses to be assigned for storing v_i will be stored in n locations $B_1, B_2, \ldots, B_n$ as follows: Generate an address and store in B_1, except store 0 if $\ell_1 = 0$. For each $k = 1,2,\ldots, n$, store 0 in B_k if $\ell_k = 0$; otherwise, the same address as in B_{i_0} where i_0 is the first $i_0 < k$ such that $0 < \ell_{i_0} \leq k$. If there exists no such i_0, then generate a new location address and store it in B_k. Note that 0 in B_k is to be interpreted as not requiring an address for v_k. To prevent the re-use of ℓ_{i_0} (since it is now superceded by ℓ_k), the value of ℓ_{i_0} in A_{i_0} is replaced by "$+\infty$" and the process is then iterated. Finally, assign the address in B_i to v_i.

IF THE NUMBER OF STEPS IS LARGE: A simple example will suffice to show a fundamental difficulty of the previous procedure when the number of steps is large or unspecified until execution time. The following routine (assuming no mistakes) can be used to (inefficiently) rearrange m numbers in ascending order:

Input $\{x_{11}, x_{12},\ldots,x_{1m}\}$

For $i = 2,3,\ldots,m$

$A_{i1} = x_{i-1,1}$,

for $j = 1,2,\ldots,(m-1)$

$x_{i,j} = \text{Min}[A_{i,j} , x_{i-1,j+1}]$

$A_{i,j+i} = \text{Max}[A_{i,j} , x_{i-1,j+1}]$

$x_{i,m} = A_{i,m}$

Output $\{x_{m1}, x_{m2},\ldots,x_{mm}\}$.

We will call this a generic algorithm because m is not specified until execution time. Here we wish to make a prior decision of what values are to be stored in the same location to be used whatever be the eventual value of m. This particular routing computes $2m^2 + m$ different values. For $m = 1{,}000$, say, it would not be practical to apply the method of the previous section. A little study shows that all the A_{ij} may be stored in a single location and all the vectors $\{x_{i,1}, x_{i,2},\ldots,x_{i,m}\}$ in the same m locations as $\{x_{i-1,1},\ldots,x_{i-1,m}\}$. Hence only $m+1$ memory locations are required by this routine to sort m numbers.

This illustrates the more important problem which we are working on, namely that of analyzing the structure of generic routines (i.e., those with unspecified parameters) to determine the minimal assignment to storage prior to specification.

REFERENCES

[1] Dantzig, G.B. and A.J. Hoffman, "Dilworth's Theorem on Partially Ordered Sets", Linear Inequalities and Related Systems, Annals of Mathematics Study 38, Princeton University Press, pp. 207-214 (1956).

[2] Dilworth, R.P., "A Decomposition Theorem for Partially Ordered Sets," Annals of Math 51, pp. 161-166 (1950).

[3] Ford, L.R., Jr., and D. R. Fulkerson, Flows in Networks, Princeton University Press (1962).

[4] Fulkerson, D.R., "Note on Dilworth's Decomposition Theorem for Partially Ordered Sets," Proc. Am. Math. Soc. 7, pp. 701-702 (1956).

Unclassified

Security Classification

DOCUMENT CONTROL DATA · R&D

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

1 ORIGINATING ACTIVITY (Corporate author) University of California, Berkeley	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3 REPORT TITLE

Optimal Assignment of Computer Storage by Chain Decomposition of Partially Ordered Sets

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)

Research Report

5 AUTHOR(S) (Last name, first name, initial)

Dantzig, George B. and Reynolds, Gary H.

6. REPORT DATE March 1966	7a. TOTAL NO. OF PAGES 9	7b. NO. OF REFS 4
8a CONTRACT OR GRANT NO. Nonr-222(83) b PROJECT NO. NR 047 033 c d	9a. ORIGINATOR'S REPORT NUMBER(S) ORC 66-6 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10 AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited.

11 SUPPLEMENTARY NOTES	12 SPONSORING MILITARY ACTIVITY Mathematical Sciences Division

13 ABSTRACT

To save storage, a program is usually written so that each variable assumes several values. This technique also leads to programming errors. For an important class of programs, it will be shown that they can be written with complete freedom in the naming of variables; leaving the task of minimizing storage requirements to the computer itself.

DD FORM 1 JAN 64 1473

Unclassified

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Storage Chain Decomposition						

INSTRUCTIONS

1. **ORIGINATING ACTIVITY:** Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. **REPORT SECURITY CLASSIFICATION:** Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. **GROUP:** Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. **REPORT TITLE:** Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. **DESCRIPTIVE NOTES:** 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.

5. **AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. **REPORT DATE:** Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. **TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. **NUMBER OF REFERENCES:** Enter the total number of references cited in the report.

8a. **CONTRACT OR GRANT NUMBER:** If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. **PROJECT NUMBER:** Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. **ORIGINATOR'S REPORT NUMBER(S):** Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. **OTHER REPORT NUMBER(S):** If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. **AVAILABILITY/LIMITATION NOTICES:** Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. **SUPPLEMENTARY NOTES:** Use for additional explanatory notes.

12. **SPONSORING MILITARY ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. **ABSTRACT:** Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS), (S), (C), or (U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. **KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.