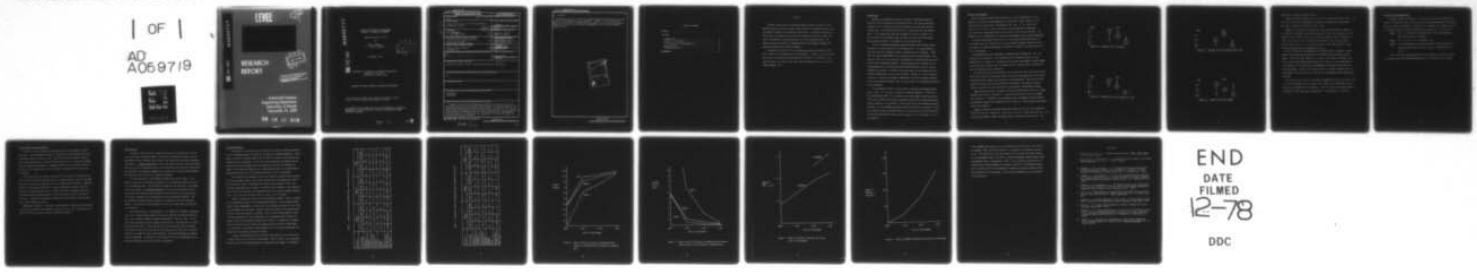
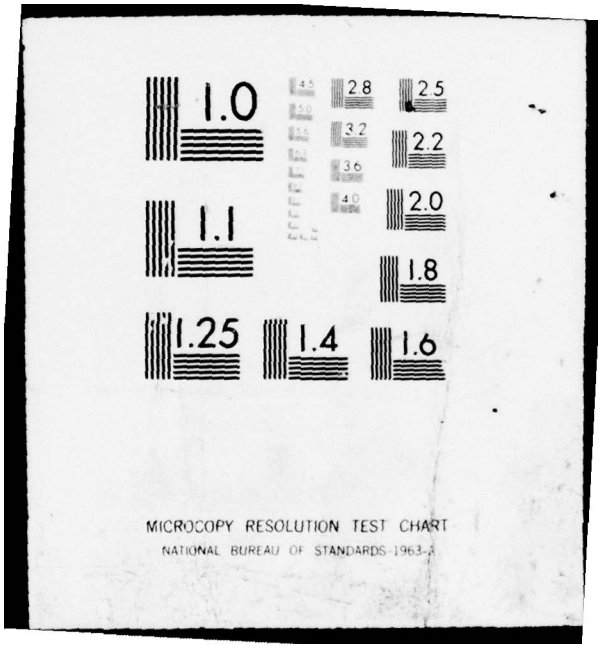
AD-A059 719 FLORIDA UNIV GAINESVILLE DEPT OF INDUSTRIAL AND SYS--ETC F/G 12/2
A NOTE ON THE VALUE OF INTERCHANGE METHODS IN SCHEDULING PROBLE--ETC(U)
SEP 78 T J HODGSON, C S LOVELAND N00014-76-C-0096

UNCLASSIFIED RR-78-12 NL

| OF |
AD
A059719



END
DATE
FILMED
12-78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL II

(12) 45

RESEARCH REPORT

Industrial & Systems
Engineering Department
University of Florida
Gainesville, FL. 32611

A NOTE ON THE VALUE OF INTERCHANGE METHODS IN SCHEDULING PROBLEMS

Research Report No. 78-12

by

Thom J. Hodgson
C. Stafford Loveland

September, 1978

Department of Industrial and Systems Engineering
University of Florida
Gainesville, Florida 32611

This research was supported in part by the Office of Naval Research, under contract number N00014-76-C-0096.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 78-12	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): A Note on the Value of Interchange Methods in Scheduling Problems,	5. TYPE OF REPORT & PERIOD COVERED: Technical
6. PERFORMING ORG. REPORT NUMBER: 78-12	
7. AUTHOR(s): Thom J. Hodgson, C. Stafford Loveland	8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0096
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Industrial and Systems Engineering, University of Florida, Gainesville, Florida 32611	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 20061102A14D Rsch in & Appl of Applied Math.
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Arlington, VA	12. REPORT DATE: September, 1978
	13. NUMBER OF PAGES: 22
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Research rept,	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited. (12) 23p.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

N/A

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Scheduling
Heuristics

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

Consider a multi-machine scheduling problem in which the jobs (of unit duration) may have non-zero release times, monotonic increasing deferral costs, and general precedence relationships between them. In general, efficient optimal solution techniques do not exist for problems of this type, and, typically realistically sized problems must be solved using heuristics. ~~We present~~ an efficient method for implementing job interchange techniques for improving heuristically derived schedules. is presented

→ next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

404 399

LB

20. (cont'd)

The method is tested on over 200 randomly generated (NP-complete) problems. 98.5% of the problems are solved optimally. Finally, it is noted that the quality of the solution technique does not appear to be limited by computation costs, but rather by the (one time) developmental cost of the interchange computer code.

TABLE OF CONTENTS

Abstract

Consider a multi-machine scheduling problem in which the jobs (of unit duration) may have non-zero release times, monotonic increasing deferral costs, and general precedence relationships between them. In general, efficient optimal solution techniques do not exist for problems of this type, and, typically, realistically sized problems must be solved using heuristics. We present an efficient method for implementing job interchange techniques for improving heuristically derived schedules.

The method is tested on over 200 randomly generated (NP-complete) problems. 98.5% of the problems are solved optimally. Finally, it is noted that the quality of the solution technique does not appear to be limited by computation costs, but rather by the (one time) developmental cost of the interchange computer code.

Introduction

Consider a multi-machine (parallel processor) scheduling problem in which the jobs have monotonic increasing deferral costs, may be partially ordered by a set of general precedence constraints, and may have non-zero release times. The scheduling objective is a function of the job deferral costs. It is assumed that jobs are scheduled for discrete periods (i.e., all jobs have unit processing time) and that the setup time either takes place between periods or is included in the processing time.

This is a reasonably general problem description and includes several problems that have been shown to be NP-complete [8]. For many problems of this type, the best available solution procedure may be a heuristic. Often there is room for improvement in the solutions found by these methods. The purpose of this note is to present a methodology for improving heuristic solutions to these scheduling problems. The methodology used is simply that of interchanging jobs in the heuristically derived schedule to see if a better schedule can be found. In theory, this could result in the evaluation of all possible combinations of jobs in the schedule. However, it is not necessary, typically, to evaluate all possible combinations to find the optimal schedule (or a nearly optimal schedule), and, if one is clever, the computational effort can be minimized.

In succeeding sections, we first define a multiple interchange methodology which can be used to search for improved schedules. We show how the computational effort for the higher order interchanges (involving several jobs) can be reduced, and develop a general structure for an open ended interchange procedure. The procedure is then tested on over 200 randomly generated problems. For the randomly generated problem two results are apparent: first, the procedure is very effective (98.5% of the problems were solved optimally); and second, computational effort does not appear to be the limiting factor to the approach.

Multiple Interchanges

After creating a schedule with a heuristic, it can be beneficial to try to improve the solution by interchanging two or more jobs. What follows is a description of the forms which interchanges may take. It is seen that some of the forms of three and four-way interchanges may be decomposed into lower order interchanges each of which improve the objective function. The mechanics of the decomposition of three-way interchanges receive closer examination.

A two-way interchange of jobs 1 and 2 in a schedule would replace job 1 on its machine with job 2 and place job 1 on the machine formerly occupied by job 2. An example of a two-way interchange is presented in Figure 1B in which jobs 1 and 2 are interchanged.

The standard three-way interchange is demonstrated in Figure 1A. Job 1 replaces job 3 in period t3. Job 3 replaces job 2 in period t2 which, in turn, replaces job 1 in period t1. The only other three-way interchange (a mirror image) occurs when job 1 replaces job 2 in period t2; job 2 replaces job 3 in period t3; and job 3 replaces job 1 in period t1.

The standard three-way interchange can be divided into four cases each of which improve the solution. Of those four cases, three can be decomposed into two two-way interchanges which improve the solution at each interchange. Thus, only one case requires an actual three-way interchange be performed to accomplish its goals. (As will be seen later, this can result in considerable computational savings.)

Case 1 involves either a precedence constraint between jobs 2 and 3 or a higher deferral cost for job 2 than job 3. In this case both jobs 2 and 3 have higher costs than job 1. Therefore, the interchange can be decomposed into the two-way interchange of Figure 1B, followed by that of Figure 1C. Both two-way interchanges improve the solution.

Case 2 involves no precedence constraint between jobs 2 and 3 and a higher deferral cost for job 3 than job 2. Since job 3 also has a higher cost than job 1, the two-way interchange of Figure 1D would be more advantageous and would save the

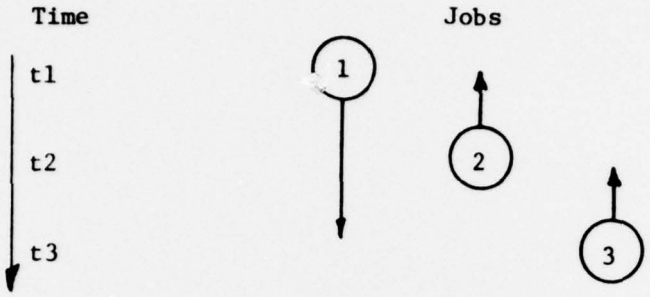


Figure 1A. Standard 3-way interchange

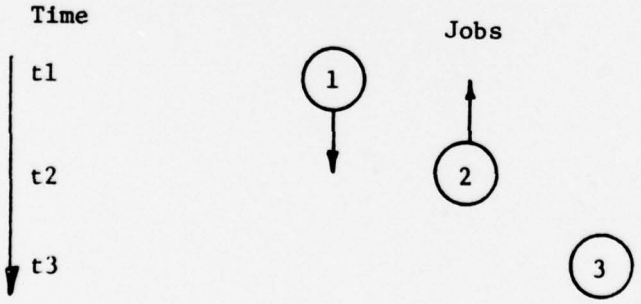


Figure 1B. Standard 3-way interchange-first step

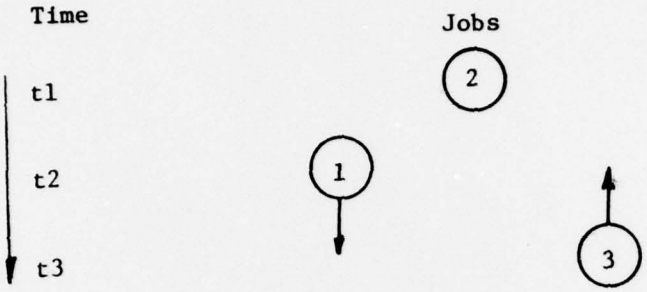


Figure 1C. Standard 3-way interchange-second step

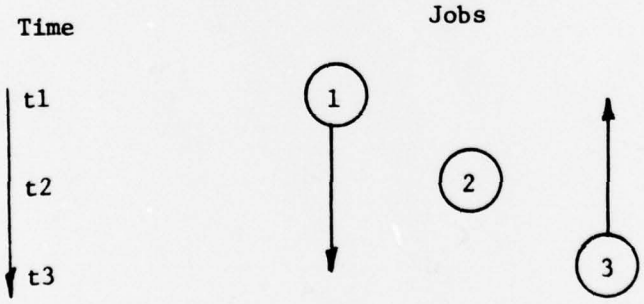


Figure 1D. Standard 2-way interchange

inevitable interchange of jobs 2 and 3.

Case 3 involves a deferral cost for job 3 no greater than that of job 1. In this case, the two-way interchange of Figure 1B accomplishes the improvement and saves a possible interchange of jobs 1 and 3.

Case 4 is the genuine non-decomposable three-way interchange of Figure 1A. It requires a precedence constraint between jobs 2 and 3 and a deferral cost no higher for job 2 than for job 1, or it requires a deferral cost no higher for jobs 2 and 3 than for job 1 with the sum of deferral costs for jobs 2 and 3 greater than for job 1. It is this case and its mirror image which the three-way interchange routine is programmed to detect.

Where the three-way interchange has two standard forms (including a mirror image), the four-way interchange has five standard forms (including a mirror image). Two of the forms, and part of a third, can be shown to decompose into sets of two-way interchanges which improve the solution at each interchange, or a combination of two and three-way interchanges which improve the solution at each interchange. Clearly, this analysis could be extended to higher levels of interchange if one was willing to evaluate the increasingly complex combinatoric structure.

The result of all of this is that when programming the search for three, four, or k-way interchanges that will improve the solution, many combinations can be explicitly eliminated from consideration. This will reduce the computational requirements to less than complete enumeration. The question is: How much?

A Multiple Interchange Method

It appears from the discussion of the previous section that there may be a savings in computation time to be realized from decomposing multiple interchanges into lower order interchanges. A logical way to implement this approach when the interchange procedures have been developed up to some level k is as follows:

Step 1. Set i = 2 and make all nondecomposable i-way interchanges of jobs which improve the solution.

Step 2. If $i < k$, set $i = i+1$. Otherwise, stop.

Step 3. If a nondecomposable i-way interchange which would improve the solution exists, make it and go to Step 1. Otherwise, to to Step 2.

By the time the interchange method has been completed, there are no two-way through k-way interchange improvements that can be made in the solution.

A Test Problem for Experimentation

In order to test the multiple interchange method, an experimental problem was chosen. The experimental problem is a parallel processor problem with serial precedences placed on groups of jobs. Each job i has an availability time a_i and a linear deferral cost c_i which differs from job to job. Lenstra [8] showed that the problem is NP-complete. Elmaghraby and Sarin [3] developed bounds on a heuristic for a relaxed version of the problem. Loveland [9] also dealt with the problem.

Other papers deal with further relaxations of the problem in which each job has the same availability time and there is a single machine. Horn [6] considers the case in which the precedences are sets of arborescences (forests). Adolphson and Hu [1] solve Horn's problem for a single arborescence in worst case time $O(n \log n)$ and shorten Horn's proof. Lawler [7] solves a problem in which the precedences involve parallel series of jobs and realizes a worst case time of $O(n \log n)$. Sidney [10] develops an algorithm which decomposes and solves general, acyclic networks of jobs.

Hodgson and Loveland [4, 5] address a multi-machine problem which has similar structure but minimizes the completion time of the latest job. Martin-Vega, and Ratliff [2] survey parallel processor scheduling problems.

The Heuristics

In order to have an initial feasible solution as a starting point for testing the multiple interchange method, two heuristic techniques are used. The two individual heuristics analyze the strings of jobs defined by the serial precedence constraints. A feasible substring of such a job string consists of the first available job on that string and the rest of the consecutive jobs from that string which are available to the machines immediately succeeding that first available machine. Each heuristic uses a form of the feasible substring.

The first heuristic, called SCHED1, schedules one job at a time. It is based on a measure of the potential penalty of not scheduling a given job on the first available machine. The calculation is made for the first job, if available, of each job string. The potential penalty is the sum of the deferral costs of those jobs on the feasible substring which would be forced into a later period, if the first available job is scheduled on the second available machine. The job having the greatest potential penalty is scheduled on the first available machine. The analysis is then repeated for each job string, until all the jobs are scheduled.

The second heuristic, called SCHED2, is an adaptation of Sidney's algorithm [10] for the single-machine problem in which all jobs have the same availability time. This heuristic schedules complete substrings of jobs at a time. It considers each substring of every feasible substring (i.e., the first job, the first and second jobs, etc.). Since each job has a unit processing time, the figure of interest is the ratio of the sum of the deferral costs of the job in the substring to the number of jobs in the substring. The substring having the highest ratio is scheduled. The analysis is then repeated for all the substrings of the feasible substrings, until all the jobs are scheduled.

Testing and Results

To evaluate the efficiency and the potential contribution towards optimality in the interchange method, over 200 test problems were randomly generated. Availability times were randomly spaced over the first 25 periods. The problems were divided into two categories: in the first each problem had two machines and 30 jobs; and the second category had four machines and 50 jobs per problem (the number of jobs used was limited by the computational limits of an enumeration algorithm which was used to find optimal solutions to the test problems).

The heuristics were applied to each test problem to provide starting points for the interchange method. Two, three, and four-way interchanges were used on each solution in order to determine the relative effectiveness of each. The statistics gathered were the percent of problems solved optimally, the average percent error for all problems, the maximum percent error, and the average execution time. (Note that the execution time is in units of 10^{-2} seconds).

Table 1 contains the results of the two-machine problems. Table 2 contains the corresponding results of the four-machine problems. The interchange method shows a steady improvement in accuracy for the individual heuristics as the level of interchanges used increases. Figures 2 and 3 are based on the combined data from the two and four-machine problems. Both of these figures appear to indicate that regardless of its starting point the interchange method rapidly (in terms of level of interchange) becomes quite accurate. In addition, by taking the best solution for each problem the joint results also show an improvement. (In other words, using several starting points for the multiple interchange procedure helps in getting around local minima).

Figure 4 demonstrates that the execution time appears to be no worse than a linear function of the level of interchange. Figure 5 shows that the developmental cost of the interchange method (as measured by the number of statements

Table 1. Statistics from 93 problems having 2 machines and 30 jobs

Heuristic	SCHED 1				SCHED 2				JOINT			
Interchange	4 WAY	3 WAY	2 WAY	NONE	4 WAY	3 WAY	2 WAY	NONE	4 WAY	3 WAY	2 WAY	NONE
Percent of Problems Solved Optimally	96.8	89.3	69.9	0.0	97.9	94.6	85.0	41.9	98.9	98.9	93.6	41.9
Average Percent Error for all Problems	0.18	0.75	3.86	83.7	0.14	0.41	1.199	5.44	0.097	0.097	0.310	5.44
Maximum Percent Error	9.01	12.3	45.7	284.	9.01	11.97	31.3	32.2	9.01	9.01	11.7	32.2
Average Times For all Problems (10^{-2} SEC)	.56898	.49025	.38601	.1825	.52930	.45089	.29701	.2290	1.0983	.94114	.68302	.4115

Table 2. Statistics from 115 problems having 4 machines and 50 jobs

Heuristic	SCHED 1				SCHED 2				JOINT			
Interchange	4 WAY	3 WAY	2 WAY	NONE	4 WAY	3 WAY	2 WAY	NONE	4 WAY	3 WAY	2 WAY	NONE
Percent of Problems Solved Optimally	93.0	66.1	55.7	0.0	96.5	85.2	79.1	33.9	98.3	88.7	85.2	33.9
Average Percent Error for All Problems	0.53	3.46	8.93	92.5	0.068	0.76	1.03	6.03	0.018	0.299	0.703	5.91
Maximum Percent Error	19.4	26.6	73.6	379.	4.17	34.9	34.9	42.9	1.79	4.69	34.9	42.9
Average Time For All Problems (10^{-2} SEC)	.8348	.6740	---	---	.7238	.6003	---	---	1.5586	1.2743	---	---

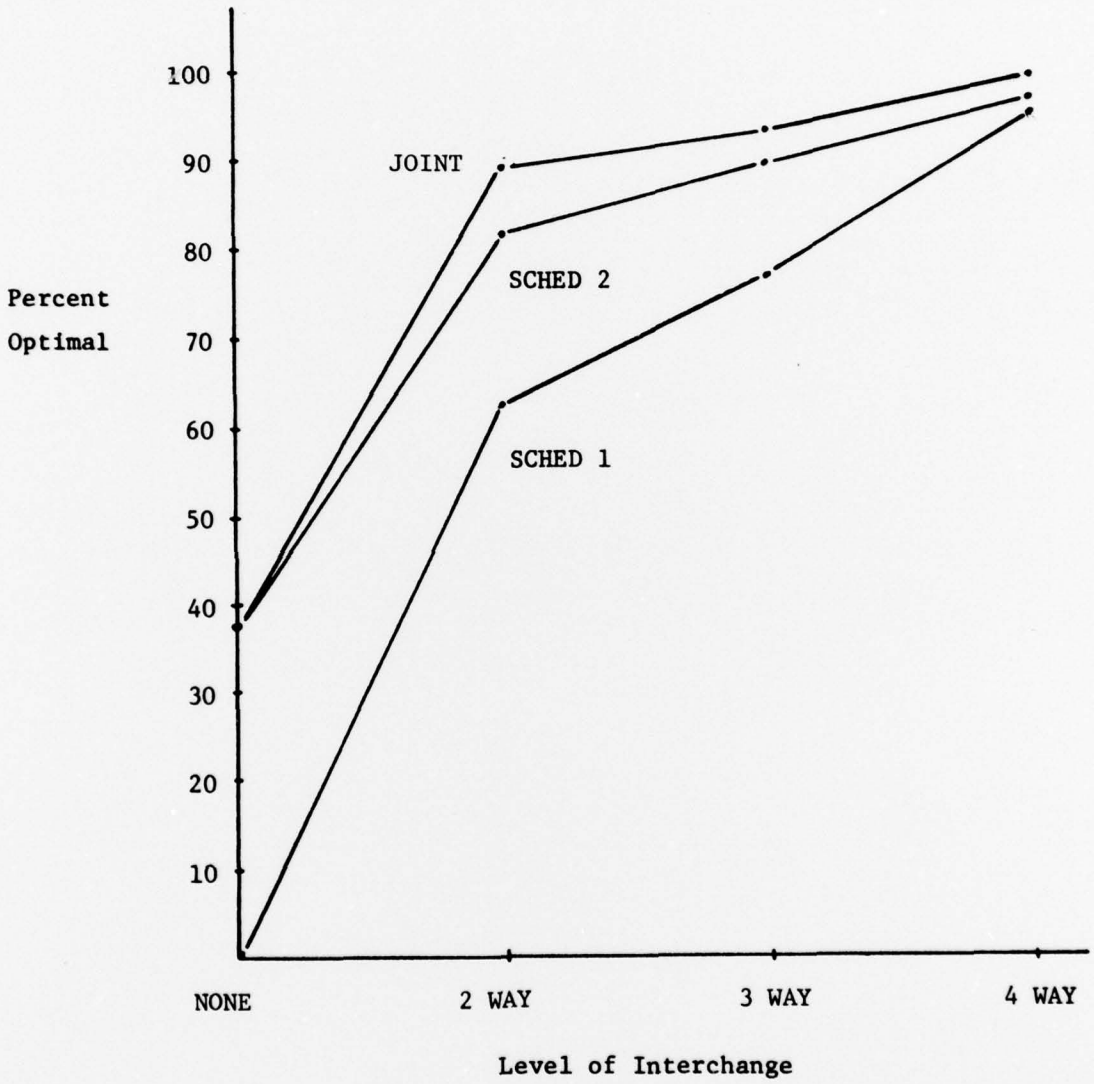


Figure 2. Graph of level of multiple interchange versus percent of problems solved optimally for combined data

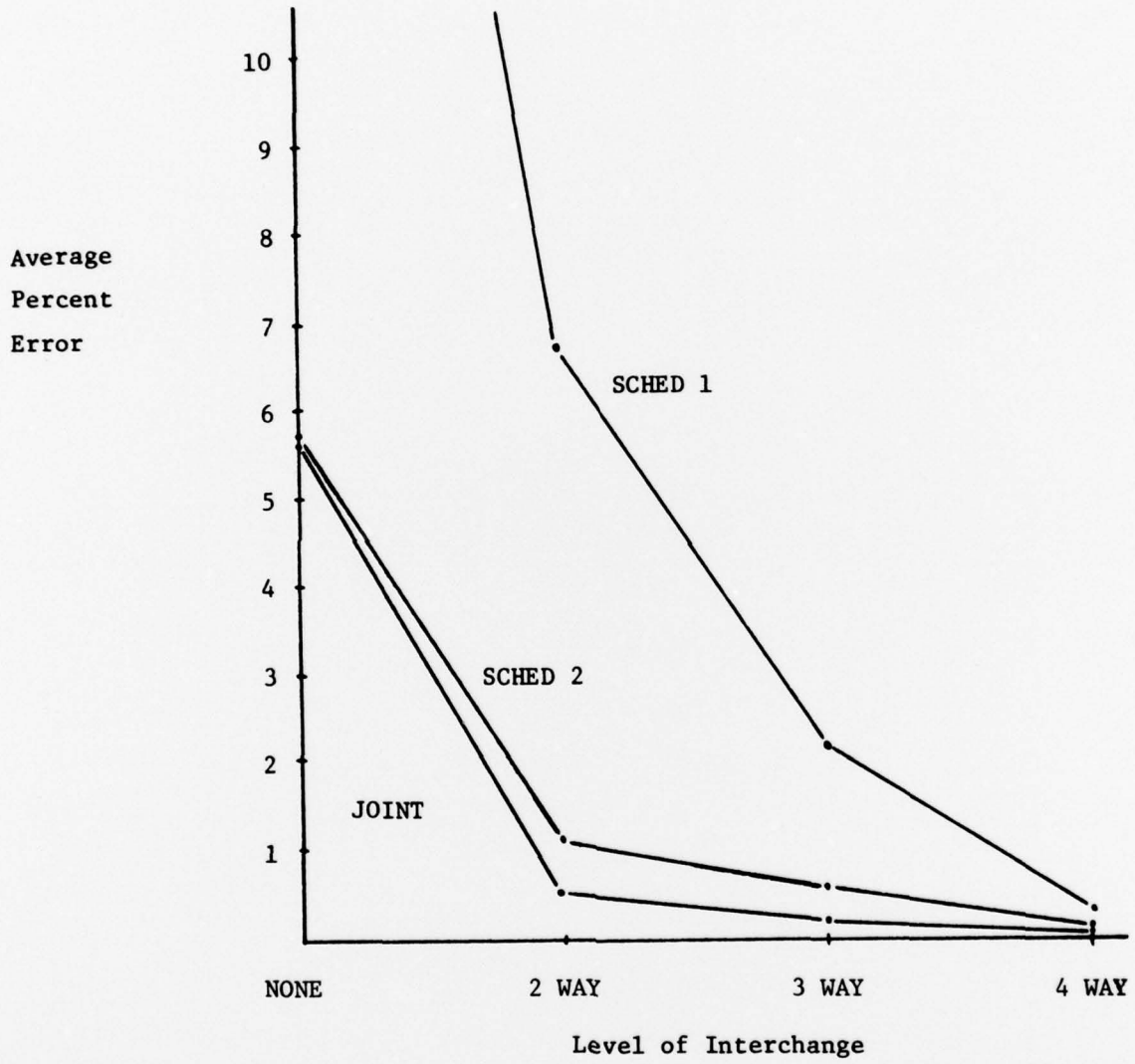


Figure 3. Graph of level of multiple interchange versus average percent error for all problems for combined data

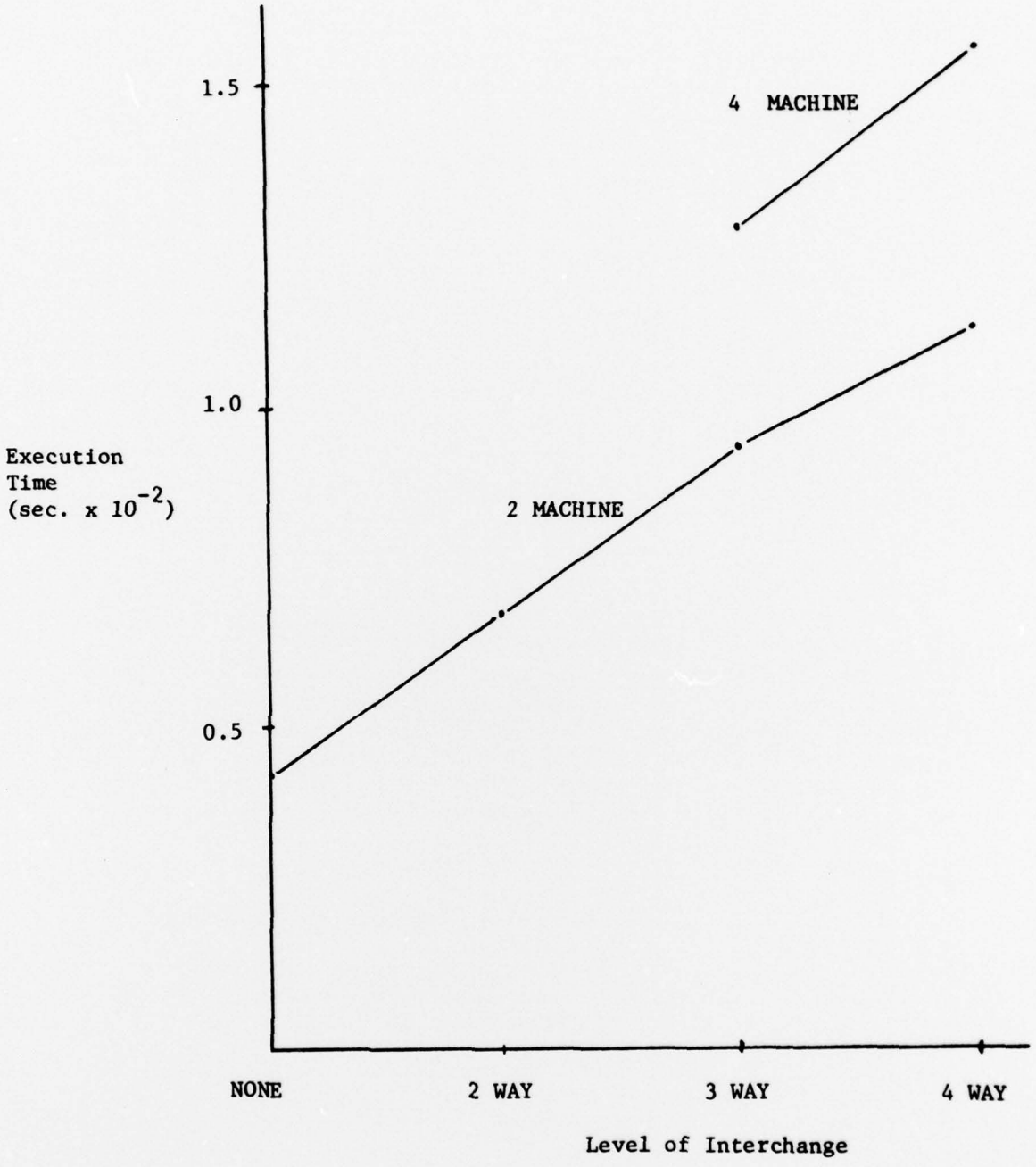


Figure 4. Graph of heuristic execution time versus level of interchange

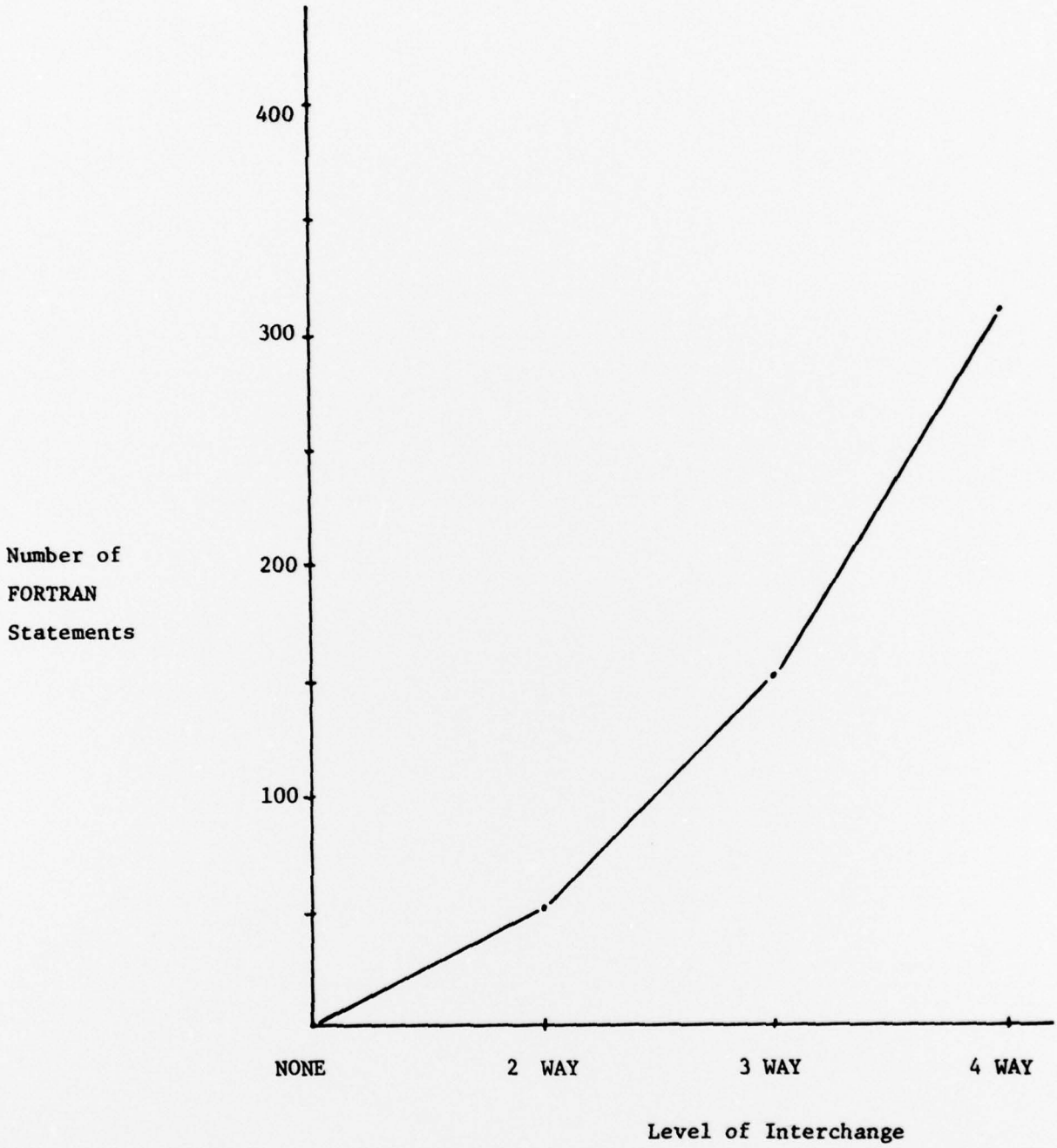


Figure 5. Graph of FORTRAN statements versus level of interchange

in the FORTRAN code) appears to be an increasing convex function of the level of interchange. While the form of Figure 5 is certainly to be expected, Figure 4 is not. The indication is that the quality of the solution may be more dependent on developmental effort (in terms of coding increasingly complex higher order interchanges) than on computational costs. If so, this may point the way for further work in the development of a general structure for interchange methods, and for further work in the development of bounds on heuristic solutions which have undergone k-way interchange. (Note that the FORTRAN code can be obtained from the authors).

References

1. Adolphson, D. and Hu, T. C., "Optimal Linear Ordering," SIAM J. Appl. Math., Vol. 25 (1973), 403-423.

2. Martin-Vega, L., and Ratliff, H. D., "Scheduling Rules For Parallel Processors" AIIE Transactions, Vol. 9, No. 4, (1977), 330-337.

3. Elmaghraby, S. E. and Sarin, S. C., "On Scheduling Precedence-Related Jobs on Parallel Machines: Bounds on the Performance of a Heuristic," North Carolina State University, Research Report No. 117, Raleigh, N.C. (1977).

4. Hodgson, T. J. and Loveland, C. S., "A Partial Lagrange Multiplier Approach to a Resource Constrained C.P.M. Problem," University of Florida, Industrial and Systems Engineering Department, Research Report 76-11, Gainesville, Fla. (1976).

5. Hodgson, T. J. and Loveland, C. S., "An Analytic Approach for a Capacitated C.P.M. Problem," University of Florida, Industrial and Systems Engineering Department, Research Report 76-12, Gainesville, Fla. (1976).

6. Horn, W. A., "Single-Machine Job Sequencing with Treelike Precedence Ordering and Linear Delay Penalties," SIAM J. Appl. Math., Vol. 23 (1972), 189-202.

7. Lawler, E. L., "Optimal Sequencing of Jobs Subject to Series Parallel Prededence Constraints," the Mathematical Centre, Amsterdam, Netherlands (1975).

8. Lenstra, J. K., "Sequencing by Enumerative Methods," Mathematical Centre, Amsterdam, Netherlands (1976).

9. Loveland, C. S., "Solution Approaches to a Multi-Stage, Multi-Machine, Multi-Product Production Scheduling Problem," University of Florida, Industrial and Systems Engineering Department, Ph.D. Dissertation, Gainesville, Fla. (1978).

10. Sidney, J. B., "Decomposition Algorithms for Single Machine Sequencing with Precedence Relations and Deferral Costs," Operations Research, Vol. 23, (1975), 283-298.