A Data Dependent Loop Scheduler Based on Genetic Algorithms

Jose Aguilar

CEMISID, Dpto. de Computación,
Facultad de Ingeniería.
Universidad de Los Andes
Mérida, VENEZUELA.

Ernst L. Leiss

Department of Computer Science
University of Houston
Houston, TX 77204-3475. USA

1 INTRODUCTION

In this paper, we study the problem of scheduling a set of *n* nested loops, with loop carried dependencies, on distributed systems. We represent the data dependencies among tasks in loops using the *loop task graph,* whose nodes represent the tasks on different iterations and arcs the dependence relationship between tasks. We also need a scheduling approach that can exploit parallelism within each iteration and among different loop iterations. We solve this problem using the loop unrolling technique and the critical path concept. Finally, we propose a scheduling algorithm based on genetic algorithms. We define a set of specific genetic operators for this problem.

2 OUR SCHEDULING LOOP TASK GRAPH APPROACH

Our algorithm follows the next steps: i) Calculate the complete loop unrolling u={b_1-1, …, b_n-1} for the Loop Task Graph G in order to determine the set of *M* tasks on the loop, where *N* is the set of tasks on the *n* nested loops (N=|V|). ii) Call our scheduling algorithm based on the GA. In general, to obtain a good scheduling we need to minimize the execution time of each task of the loop. The parallel execution time (Tp) of the loop on *K* processors, according to a given assignment $\Pi(g)$ of the tasks of the loop, where *g* is one of the possible task assignments (g=1, ..., K^M), is the earliest time at which a given task $T_j^{I^1 \ldots I^n} \in TS$ will terminate.

$$Tp(\prod(g)) = \max_{T_j^{I^1 \ldots I^n} \in TS} \left\{ T_j^{I^1 \ldots I^n}(\prod(g)) \right\}$$

$$\forall j = 1, ..., N,\ \forall I^1 = 1, ..., b^1, \ \ldots, \ \forall I^n = 1, ..., b^n$$

The instant at which $T_j^{I1 \ldots In}$ will terminate is:

$$T_j^{I^1 \ldots I^n}(\prod(g)) = T_j^{I^1 \ldots I^n} + \max\left\{ X_i^{I'^1 \ldots I'^n}(\prod(g)) \right\} + \left| \begin{array}{l} \max_{Ti \in R} \left\{ T_i^{I'^1 \ldots I'^n}(\prod(g)) + \lambda_{ij}^{I^1 \ldots I^n, I'^1 \ldots I'^n}(\prod(g)) \right\} \\ - \max\left\{ X_i^{I'^1 \ldots I'^n}(\prod(g)) \right\} \end{array} \right|_{if\ (COND=true)}$$

COND determines the overhead due to the communication time of the predecessor task of Tj that has finished last when this time plus its completion time is bigger than the completion time of the last task executed on the same processor where Tj will start its execution. $X_i^{I1 \ldots In}(\Pi(g))$ gives the completion time of the tasks already executed on the processor where Tj has been allocated. $\lambda_{ij}^{I1 \ldots In}(\Pi(g))$ is the communication time when two tasks are allocated on different processors. The objective function (OF) is:

$$OF = \min_g \{ Tp(\prod(g)) \} = \min_g \left\{ \max_{T_j^{I^1 \ldots I^n} \in ST} \left\{ T_j^{I^1 \ldots I^n}(\prod(g)) \right\} \right\}$$

We define a heuristic approach based on genetic algorithms (GA) to solve the data dependent loop scheduling problem. The GA applied in our problem follows the next procedure: we define a space of research of *M*n* individuals. An individual represents the assignment of the different nodes of the replicated task graph G_u that represents the complete loop unrolling of the original loop task graph G. We number the set of tasks of the replicated task graph G_u from 1 to M. In this way, each individual is composed of *M* elements. The *ith* element has two values: a value between *1* and *K*, indicating the processor where task *i* will be allocated, and another value indicating its execution order on this specific processor. According to the execution order, we can find a deadlock (a task is scheduled before its predecessor). The cost for the individuals that represent this situation will be infinite. We use the following specific genetic operators:

- *Mutation of the Processor Allocation (MA)*: this operator defines a new allocation *k* for a given task *i*.
- *Permutation of the Execution Order (PEO):* For a given value of the processor field, we permute the execution order of the tasks assigned to it.
- *Permutation of the Allocation (PA):* For a given value of the execution order field, we permute the processor where they will be assigned.
- *Partial Crossover (PC):* This operator takes the part of two individuals that correspond to a given execution order or processor; it compares if they are similar, then it exchanges this specific part of information between them. This operator is extended for a set of execution orders or set of processors.

Data oriented genetic operators for one-machine scheduling problems

Marie-Claude Portmann
MACSI team of INRIA Lorraine and LORIA
Ecole des Mines de Nancy, Parc de Saurupt
54042 Nancy Cedex, France

Mohamed-Ali Aloulou
MACSI team of INRIA Lorraine and LORIA
Ecole des Mines de Nancy, Parc de Saurupt
54042 Nancy Cedex, France

We consider the one-machine scheduling problem denoted by $1/S_{sd}/\sum w_i T_i$ in the Graham's notation, i.e. the one-machine problem with identical release dates, sequence dependent set-up times and weighted sum of tardiness as performance measure. We use the classical "permutation" encoding which gives operation order. We propose new genetic operators called "data oriented" that use not only the permutation (order of the operations) contained in the chromosome but also an analysis of the corresponding solution. We consider here two data oriented processes.

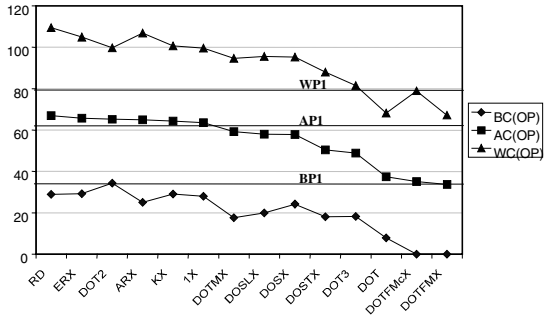
1. Set-up time improvement process, denoted by DOS, consists in studying the operation sequence corresponding to one mate and to find the longest sequence in which the set-up time average is minimal. It uses both the contents of the chromosome and the instance data.

2. Tardiness improvement process, denoted by DOT, consists in sorting the operations in a new order that depends on their processing times P_j, their tardiness penalties W_j and the operation tardiness values T_j corresponding to the mate solution. For each operation j of the mate, a measure $K(j)$ is given by the following equation:

$$K(j) = W_j * (T_j + 1 / P_j)$$

Clearly, the most important term is T_j. $1/P_j$ is important only when two operations have the same weighted tardiness.

Several operators were built with these improvement processes and compared to some known genetic crossovers. The comparison is made on a population of 900 mates generated randomly. Many families of problem instances were experimented and for each one the results can be visualized by a graph (see the following figure) giving, for each operator OP, the best value BC(OP), the worst value WC(OP) and the average value AC(OP) of the created offspring. It is also interesting to show the percentage of good solutions (Gd) obtained from couples of good, medium (Md) or bad (Bd) mates. These results are shown in the table below. We use the following notations: DO for Data Oriented, T for Tardiness, S for Set-up times, X for crossover, F (resp. M) for DO applied to Father (resp. Mother), 2 (resp. 3) for DO applied twice (resp. three times) and RD for random. 1X, KX, ERX and ARX are classical genetic crossovers.



→ Gd	Gd →	Md →	Bd →
DOTFMX	82,5 %	93,6 %	95,6 %
DOT	82,5 %	85,4 %	90,9 %
DOSX	81,2 %	34,2 %	11,1 %
DOTMX	80,0 %	61,5 %	38,6 %
DOSLX	80.0 %	36,6 %	11,4 %
1X	67,5 %	16,9 %	2,2 %
DOSTX	52,5 %	32,1 %	14,6 %
KX	51,2 %	18,4 %	2,1 %
DOTFMcX	48,7 %	93,6 %	95,6 %
DOT3	37,5 %	48,1 %	56,2 %
ARX	26,2 %	16,0 %	5,3 %
ERX	16,2 %	13,9 %	5,7 %
DOT2	13,7 %	9,6 %	5,3 %
RD	6,2 %	7,2 %	7,5 %

The results show that data oriented operators are globally better than classical ones especially when applied to rather bad chromosomes (applying them twice, DOT2 gives results similar to random). Hence, they can be used in a genetic algorithm in order to improve population quality and/or accelerate algorithm convergence.

References

L. Davis, *ICGA'1985*, 136-140.

M.C Portmann, *WPPC'1996,* I-XXIV.

M.C Portmann, A. Vignier, *GECCO'2000,* 331-338.

Y. Whitley et al, *ICGA'1989,* 133-140.

A Comparison of Genetic Algorithms and Tabu Search Techniques for the Resource-Constrained Project Scheduling Problem

Manuel Vázquez
Technology of Industrial Automation
PDVSA Petróleo y Gas
Caracas Venezuela
vazquezm@pdvsa.com

Abstract

The Resource Constrained Scheduling Project Problem (RCPSP) models a large number of real world problems. This paper examines various Genetic Algorithms (GAs) and Tabu Search (TS) techniques for the RCPSP. These algorithms are combined with heuristic-based techniques that produce feasible schedules. The objective of this research is to evaluate the effectiveness of heuristic-based approaches to this problem. Algorithms are compared against the state-of-the-art techniques for the RCPSP on Project Scheduling Problem Library benchmark instances. Two algorithms **TS_shifts** and **OB_RS** that produce near optimal solutions were created.

1 Introduction and new algorithms

In its most general form, the RCPSP can be summarized as: Given a set of activities, a set of resources, and a measurement of performance, what is the best way to assign the resources to the activities such that the performance is maximized?

It is well known that the RCPSP belongs to the class of $NP-$hard optimization problems. Since practical applications require solving large problems, heuristic-based approaches for the RCPSP have been developed. Optimal procedures also have been developed for solving the RCPSP. The results presented here are focused on heuristic-based algorithms, which are better suited for practical applications. The criteria utilized to determine the effectiveness of the algorithms are based on the capacity of producing near optimal solutions, minimizing the number of schedules generated and minimizing the solutions diversity across different runs. Two new heuristic-based algorithms were created.

1.1 OB_RS and TS_shifts

OB_RS is a generational GA based on Order-Based operators that were applied succesfully by Vázquez and Whitley [2000] to the static and dynamic Job Shop Scheduling Problem. **OB_RS** uses an activity list representation and its selection mechanism is proportional selection.
In **TS_shifts** the neighborhood is formed picking an activity a randomly. A random subset from the set of valid positions to move a is calculated. The maximum number of moves is set by default to 1/4 of the number of activities of the RCPSP instance. The tabu list stores the position (and the iteration) to where an activity was previously moved.

2 Conclusions

TS_shifts, is a viable alternative for the RCPSP, but its results are diverse (high variance among the makespan of the generated solutions) and suffer scalability problems. **OB_RS** did not show scalability problems, consumes less computational resources than **TS_shifts** and has small variance.**OB_RS** looks like a good candidate to be applied in real-time scheduling and in dynamic scheduling applications.

Acknowledgments

This work was sponsored by Petróleos de Venezuela, PDVSA.

References

[Vázquez, et al., 2000a]: M. Vázquez and L.D. Whitley. "A Comparison of Genetic Algorithms for the Dynamic Job Shop Scheduling Problem". Genetic and Evolutionary Computation Conference, GECCO 2000.