Parallel Computing 48 (2015) 1–14

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Parallel Local Search to schedule communicating tasks on identical processors

Tatjana Davidović[a,*], Teodor Gabriel Crainic[b,c]

[a] Mathematical Institute, Serbian Academy of Science and Arts, Kneza Mihaila 36, 11001 Belgrade, P.O. Box 367, Serbia
[b] Department of Management and Technology, School of Management, Université du Québec à Montréal, Montréal, QC, Canada
[c] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), C.P.6128, Succ.Centre-ville, Montréal, Canada H3C3J7

ARTICLE INFO

Article history:
Received 7 September 2013
Revised 30 March 2015
Accepted 5 April 2015
Available online 9 April 2015

Keywords:
Distributed memory multiprocessors
Load balancing
Neighborhood decomposition
Directed acyclic graph
Meta-heuristics

ABSTRACT

This paper reports on the analysis of parallelization strategies for Local Search (LS) when the neighborhood size varies throughout the search. The Multiprocessor Scheduling Problem with Communication Delays (MSPCD) is used as benchmark for illustrating the methodology and results. The dynamic load distribution strategy implemented within a supervisor–worker framework is shown to offer the best performance. Experimental results on several sets of instances with up to 500 tasks show excellent speedups (super-linear in most cases) while preserving the quality of the final solution. The proposed parallel LS is incorporated into Multi-start Local Search and Variable Neighborhood Search meta-heuristic frameworks to analyze its efficiency in a more complex environment. The comparison between the sequential and parallel versions of each meta-heuristic, using various numbers of processors, shows improvement in the solution quality within proportionally smaller CPU time.

1. Introduction

Local Search (*LS*) is a well-known methodological tool for finding good suboptimal solutions for combinatorial optimization problems. It is widely used to improve the quality of solutions obtained by constructive heuristics. More importantly, it serves as a basic building block for a wide variety of meta-heuristic methods, neighborhood-based ones in particular. Although meta-heuristics have proven efficient for many combinatorial optimization problems and real-life applications, there are still problems that cannot be treated effectively in a reasonable amount of time due to their complexity or the large size of the practical instances, or both. As the recent literature shows [7], the parallelization of search procedures offers a promising approach to increase the efficiency of heuristic and meta-heuristic methods.

LS is generally computationally intensive, therefore, it has to be very efficient, especially within a meta-heuristic framework where it is called upon very frequently. Parallel strategies for LS procedures have been developing for more than 20 years [28] together with a variety of computational paradigms [7,39]. However, each particular application of LS is specific and deserves special treatment in order to obtain the most efficient implementation. Therefore, the main objective of this paper is to develop efficient parallelization strategies for the special case of LS procedures addressing a major class of combinatorial optimization problems.

* Corresponding author. Tel.: +381113349072; fax: +381112186105.
E-mail addresses: tanjad@mi.sanu.ac.rs (T. Davidović), TeodorGabriel.Crainic@cirrelt.net (T.G. Crainic).
URL: http://www.mi.sanu.ac.rs/t~anjad (T. Davidović)

http://dx.doi.org/10.1016/j.parco.2015.04.002
0167-8191/© 2015 Elsevier B.V. All rights reserved.

We address the *Multiprocessor Scheduling Problem with Communication Delays* (*MSPCD*). This is an essential problem not only in computer science, but also in various applications involving scheduling of multiple resources (robotics, aircraft control, etc.). Identifying good solutions for this problem may be extremely hard, even when advanced meta-heuristic strategies are deployed [12]. The same paper also showed that LS accounts for the main part of the computational burden in meta-heuristic search, with 99% of the execution time devoted to constructing and evaluating neighbors. The aim of this paper therefore is to provide an efficient parallel LS method for improving the existing scheduling results for MSPCD.

LS evolves within a search environment comprised of neighborhoods of highly variable sizes when addressing MSPCD, as detailed in Section 2. It has been shown, e.g., [1,6,7], that any static parallelization of such a dynamical search process yields an inefficient utilization of available resources. We propose a parallelization strategy involving the dynamic fine-grained partition of the neighborhood, which leads to an efficient load-balanced parallel execution of LS. Moreover, we show experimentally that the proposed parallel LS procedure brings remarkable improvements to the performance of the associated meta-heuristics for MSPCD.

The main contributions of this paper are: (1) identification and analysis of the issues related to the parallelization of LS in the presence of neighborhoods of variable size, for which static decomposition is inefficient; (2) development of appropriate parallelization strategies for this type of LS procedure; (3) experimental evaluation of various parallel LS strategies; and (4) improvement of the best known results for MSPCD. Note that, the proposed parallelization strategies do not depend on the particular application and can be easily adapted to different combinatorial optimization problems.

The paper is organized as follows. Section 2 recalls MSPCD and the sequential, permutation-based LS procedure. The proposed parallelization strategies are described in Section 3, while the implementation details are given in Section 4. Section 5 contains the results of extensive experimentation with the proposed parallel LS procedures. The experimental evaluations of Variable Neighborhood Search (VNS) and Multistart Local Search (MLS) with the selected parallel LS procedure are described in Section 6. We conclude in Section 7.

2. MSPCD definition and sequential LS

The MSPCD can be described as follows. Tasks (or jobs) have to be executed on a multiprocessor system containing several identical processors, and we have to decide where and when each task will be executed, such that the total completion time (makespan) is minimum. The duration of each task is known as well as the precedence relations among tasks, i.e., what tasks should be completed before some other could begin. When dependent tasks are executed on different processors, the data transfer time (or communication delay) is also known. Task preemption and duplication (redundant executions) are not allowed. Two mathematical programming formulations, one based on task ordering and the other inspired by rectangle packing, are detailed in [11]. Recent developments on exact formulations are presented in [37,38]. We recall next the combinatorial formulation of MSPCD for which the sequential LS procedure is developed in [12]. The second subsection provides a brief overview of the sequential LS, including the data structures and neighborhood definitions.

2.1. Multiprocessor scheduling with communication delays

The tasks to be scheduled are represented by a directed acyclic graph (DAG) [9,21,36], called *task graph* (*TG*), defined as a tuple $\mathcal{G} = (T, L, E, C)$. Here $T = \{t_1, \ldots, t_n\}$ stands for the set of tasks; $L = \{l_1, \ldots, l_n\}$ represents the computation times of the tasks (execution times, lengths, durations); $E = \{e_{ij} \mid t_i, t_j \in T\}$ describes the set of communication edges; and $C = \{c_{ij} \mid e_{ij} \in E\}$ is the set of edge communication costs. The set E defines precedence relations between tasks. A task cannot start its execution unless all its predecessors are completed and all relevant data are available. The communication cost $c_{ij} \in C$ represents the amount of data transferred between tasks t_i and t_j when they are executed on different processors. When both tasks are scheduled to the same processor no communication is required. An example of the task graph containing 12 nodes (tasks) is given in Fig. 1. The numbers within the circles represent the task indexes. Weights on the task-graph edges represent the communication cost. Task durations are listed in the table presented below the task graph.

The multiprocessor architecture $\mathcal{M}$ is assumed to contain p identical processors with their own local memories, communicating by exchanging messages through bidirectional links of the same capacity. This architecture is modeled by a *distance matrix* $D = [d_{kl}]_{p \times p}$ [9,13], where the element (k, l) represents the minimum distance between the processors p_k and p_l calculated as the number of links along the shortest path between the two processors. The distance matrix is therefore symmetric with zero diagonal elements. A multiprocessor system containing four processors connected in a ring architecture and the corresponding distance matrix are presented in Fig. 2.

The scheduling of TG $\mathcal{G}$ onto $\mathcal{M}$ consists in determining the index of the processor p_k and starting time instant S_i for each task t_i in TG in such a way as to minimize some objective function. The usual objective function (that we use in this paper as well) is the completion time $C_{max} = \max_{t_i \in T}\{S_i + l_i\}$ of the scheduled task graph (referred to as *makespan* or schedule length) [4]. The starting time S_i of a task t_i depends on the completion times of its predecessors and the amount of required communication. The communication time between t_i assigned to processor p_k and t_j assigned to processor p_l is computed as $\gamma_{ij}^{kl} = c_{ij} \cdot d_{kl} \cdot \text{ipc}$, where ipc represents the inter-processor communication speed (the communication-to-computation-ratio (CCR) from [23]). If $l = k$, then $d_{lk} = 0$, implying $\gamma_{ij}^{kl} = 0$. The optimal solution of the task graph represented in Fig. 1 on the multiprocessor architecture given in Fig. 2 is illustrated by the Gantt chart in Fig. 3.