Relatório Técnico

Núcleo de Computação Eletrônica

On the Efficiency of a Genetic Algorithm for the Multiprocessor Scheduling Problem

Ricardo C. Corrêa

NCE - 07/97

Universidade Federal do Rio de Janeiro

On the Efficiency of a Genetic Algorithm for the Multiprocessor Scheduling Problem

*Ricardo C. Corrêa**
Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro
Caixa Postal 2324
CEP 20001-970 RJ
Brazil
correa@nce.ufrj.br

Abstract

In the *multiprocessor scheduling problem* a given program is to be scheduled in a given multiprocessor system such that the program's execution time is minimized. This problem being very hard to solve exactly, many heuristic methods for finding a suboptimal schedule exist. An efficient genetic algorithm which introduces some knowledge about the scheduling problem represented by the use of a list heuristic in the *crossover* and *mutation* genetic operations was recently proposed [3]. In this paper, we investigate the efficiency of this genetic algorithm from a theoretical point of view. In particular, we demonstrate the ability of the knowledge-augmented crossover operator to generate all the space of feasible solutions.

Keywords: Multiprocessors, scheduling problems, list heuristics for scheduling problems, genetic algorithms, NP-hard, optimization.

*Partially supported by the brazilian agency FAPERJ.

1 Introduction

Let a *(homogeneous) multiprocessor system* be a set $\mathcal{P} = \{p_1, \cdots, p_m\}$ of m identical processors, $m > 1$. Each processor has its own memory, and each pair of processors communicate exclusively by message passing through a completely connected interconnection network in which all links are identical. Each processor can execute at most one task at a time and task preemption is not allowed. While computing, a processor can communicate through one or several of its links. Additionaly, let a *parallel program* be a set of communicating tasks to be scheduled to the multiprocessor system under a number of precedence constraints, which is described by an acyclic digraph $\mathcal{D} = (\mathcal{T}, A)$. The vertices represent the set $\mathcal{T} = \{t_1, \ldots, t_n\}$ of tasks and each arc represents the precedence relation between two tasks. To each task is associated a cost, representing its execution time on any processor. In addition, to every arc $(t_{i_1}, t_{i_2}) \in A$ there is an associated positive weight representing the transfer time of the message sent by t_{i_1} to t_{i_2} using any link of the multiprocessor system. If both message source and destination are scheduled to the same processor, then the cost associated to this arc becomes null.

Given a parallel program to be executed on a given multiprocessor system, the *multiprocessor scheduling problem* (MSP) consists of finding a task schedule that *minimizes* the execution time of the parallel program. Considering the communications and the precedence constraints between tasks, it follows that different schedules of each task satisfying the precendence constraints lead to different execution times of the parallel program (see for instance [1, 2, 4, 7, 8] and references therein). This problem is known to be NP-hard. Since an exhaustive search is often unrrealistic, most of the attempts to efficiently solve the MSP has been done on fast heuristic methods to find *suboptimal solutions*, i.e., solutions whose optimality cannot be guaranteed. In other words, the purpose of such heuristic methods is to be able to determine a good solution, even when the instance size leads the exhaustive search to be too long. The most studied heuristic methods for multiprocessor scheduling problems are the so called *list heuristic* [4].

Another heuristic method used in the scheduling problem context is the meta-heuristic known as *genetic algorithms* [5, 6]. In this paper, we study the theoretical efficiency of the genetic algorithm proposed in [3] in which knowledge is integrated inside the crossover and mutation operators. In particular, we demonstrate that the crossover operator of this algorithm is able to generate all the search space. In [3], this is argued to be a significant improvement over previous algorithms.

The remaining sections are organized as follows. In Section 2, we precisely specify the space

of feasible schedules of the MSP and recall the principles of genetic algorithms. The operators of the particular genetic algorithm analized in this paper are described in Section 3. This algorithm is then analyzed in Section 4, where the main results of this paper are stated and demonstrated. Finally, Section 5 is devoted to the conclusions and perspectives.

2 Preliminaries

In this section, we define more formally the search space of the MSP and the principles of genetic algorithms.

2.1 Search space

Let a *schedule* be a vector $s = \{s_1, \cdots, s_n\}$, where $s_j = \{t_{i_1}, \cdots, t_{i_{n_j}}\}$, i.e., s_j is the set of the n_j tasks scheduled to p_j. For each task $t_{i_l} \in s_j$, l represents its execution rank in p_j under the schedule s. Further, for each task t_i, we denote $p(t_i, s)$ and $r(t_i, s)$, respectively, the processor and the rank in this processor of t_i under the schedule s. The execution time yielded by a schedule is called *makespan.* We consider uniquely the schedules whose computation of the introduction dates for the tasks is done using a list algorithm. They follow a *list heuristic* whose principle is to schedule each task t_i to $p(t_i, s)$ according to its rank $r(t_i, s)$. In addition, the task is scheduled as soon as possible depending on the schedule of its immediate predecessors in $\mathcal{D}$.

A list heuristic builds a schedule step by step. At each step, the tasks that can be scheduled (called *free tasks*) are those whose all predecessors have already been scheduled. Then, we choose one of such tasks, say t_i, according to a certain rule R_1. Additionaly, we choose a processor, say p_j, to which t_i will be scheduled according to another rule R_2. We then schedule t_i to p_j as soon as possible. This algorithm finishes when all tasks have been scheduled. At an iteration k of this algorithm, let $R(k)$ be the set of tasks remaining to be scheduled, and $F(k)$ the set of free tasks from $R(k)$. Initially, $R(0) = \mathcal{T}$ and $F(0) = \{t_1\}$. Thus, at an iteration $k > 0$, we choose a task from $F(k)$, we take it out from both $R(k)$ and $F(k)$, and we schedule it to $p(t_i, s)$, as soon as possible. This algorithms finishes when $F(k) = \emptyset$.

We define that a schedule s is *feasible* if and only if the above algorithm that constructs s finishes at iteration $k = n$. This means that all tasks could be scheduled since exactly one task is scheduled at each iteration. It is clear that the schedule obtained is minimal with respect to the makespan. The search space is composed of all feasible schedules.

2.2 Genetic algorithms

A genetic algorithm is a guided random search method where elements (called *individuals*) in a given set of solutions (called *population*) are randomly combined and modified. It starts with an initial population randomly generated that evolves iteratively through generations in order to improve the *fitness* of its individuals and until some termination condition is achieved. The fitness of an individual s_1 is said to be *better* than the fitness of another individual s_2 if the solution corresponding to s_1 is closer to an optimal solution than s_2. In our case, the fitness of an individual is defined as the difference between its makespan and the one of the individuals having the largest makespan in the population. Notice that the best individual correspond to the one having the smallest makespan and the largest fitness. In what follows, we review the operators that compose an iteration of a genetic algorithm.

The *selection* operator allows the algorithm to take biased decisions favoring good individuals when changing generations. Starting from a population P_1, this transformation is implemented iteratively by generating a new population P_2 of the same size as P_1, as follows. Initially, the best individual of P_1 is replicated, with only one copy kept in P_1 and the other inserted in P_2. Then, at each iteration, we randomly select an individual $s_1 \in P_1$ according to its fitness. Then, s_1 is duplicated into a new individual s'_1, and s_1 is kept in P_1 while s'_1 is inserted into P_2. This process is repeated until P_2 reaches the size of P_1. Notice that, using this scheme, each individual can be selected more than once or not at all. Thus, some of the good individuals are replicated, while some of the bad individuals are removed. As a consequence, after the selection, the population is likely to be "dominated" by good individuals.

Genetic algorithms are based on the principles that crossing two individuals can result on offsprings that are better than both parents, and that a slight mutation of an individual can also generate a better individual. The *crossover* takes two individuals of a population as input and generates two new individuals, by crossing the parents characteristics. Hence, the offsprings keep some of the characteristics of the parents. The *mutation* randomly transforms an individual that was also randomly chosen.

The structure of the algorithm is a loop composed of a selection followed by a sequence of crossovers and a sequence of mutations. Let the population be randomly divided in pairs of individuals. The sequence of crossovers corresponds to the crossover of each of such pairs. After the crossovers, each individual of the new population is mutated with some (low) probability. This

probability is fixed at the begining of the execution and is constant. Moreover, the termination condition may be the number of iterations, execution time, results stability, etc.

3 A combined genetic-list algorithm

In this section, the combined genetic-list algorithm presented in [3] is described. In this combined algorithm, knowledge about the scheduling problem is integrated into the crossover and mutation operators based on a list heuristic.

3.1 Coding of solutions

The coding of an individual s is composed of m strings $\{s_1, s_2, \cdots, s_m\}$. There is a one to one correspondance between processors and strings, where each string represents the tasks scheduled to some specific processor. Each string s_j represents the tasks scheduled to processor p_j in s, and these tasks appear in s_j in the order of their execution in the schedule s. It is easy to see that this encoding scheme using strings may represent schedules not satisfying the precedence constraints. For this reason, a method that guarantees that all strings in the initial population or produced by crossovers or mutations will correspond to feasible schedules is used, as indicated in what follows.

3.2 Initial population

Each individual of the initial population is randomly genererated using a list heuristic with the following rules:

I-1. R_1: choose a task at random.

I-2. R_2: choose a processor at random.

Since a list heuristic is used, all individuals in the initial population correspond to feasible schedules.

3.3 Genetic operators

The knowledge about the scheduling problem is integrated into the genetic operators as described in the following.

3.3.1 Selection

Recall the principle of a selection operation discussed in Subsection 2.2. In what follows, we present the "roulette wheel" principle used to randomly select an individual from P_1. In its implementation, each individual is assigned an interval, whose length is proportional to its fitness. For instance, task t_i is assigned to the interval $[1, fitness(t_1)]$, task t_2 is assigned to the interval $[fitness(t_1) + 1, fitness(t_1) + fitness(t_2)]$ and so on. A number between 1 and $\sum_{i=1}^{n} fitness(t_i)$ is drawn at random. An individual is then selected if the randomly drawn number belongs to its interval. Thus, the better the fitness of an individual, the better the odds of it being selected.

3.3.2 Crossover

Let s_1 and s_2 be two individuals which should generate two offsprings. The first step consists of separating the two individuals into two parts. In order to describe the crossover and the mutation operators, we need a precedence relation that stems from the precedences implyed by the tasks scheduled to the same processor in a given schedule, say s, and is defined as follows.

$$\begin{aligned} A(s) = A \cup \{(t_{i_1}, t_{i_2}) \mid \ & (t_{i_1}, t_{i_2}) \notin A, \\ & p(t_{i_1}, s) = p(t_{i_2}, s) \text{ and} \\ & r(t_{i_1}, s) = r(t_{i_2}, s) - 1\}, \end{aligned} \tag{1}$$

where the arcs that belongs to $A(s)$ but not to A have cost zero. We denote $\mathcal{D}(s)$ the digraph $(\mathcal{T}, A(s))$. We also define the relation A^+ as the transitive closure of A, and analogously, $A^+(s)$ as the transitive closure of $A(s)$. From these definitions, a schedule s is feasible if and only if $\mathcal{D}(s)$ is acyclic [3].

Let s_1 and s_2 be two individuals which should generate two offsprings. The first step consists of separating the two individuals into two parts. Call a subset V' of tasks verifying the following property

if $t \in V'$ then all predecessors of t also belong to V'

a *closed task set.* In order to ensure consistency, a partition V_1, V_2 of the tasks is determined such that V_1 is a closed task set. To do so, define the digraph $(\mathcal{T}, A(s_1) \cup A(s_2))$ representing the dependencies stemming from the task digraph as well as from the two schedules s_1 and s_2. Then, let $T = \mathcal{T}$. We execute the following steps while $T \neq \emptyset$.

C-1. Choose randomly a task $t_i \in T$ and $V = V_j$, $j = 1$ or 2.

C-2. If $V = V_1$ then

$$V_1 \leftarrow V_1 \cup \{t_i\} \cup \{t_{i'} : t_{i'} \in T \text{ and } (t_{i'}, t_i) \in (A^+(s_1) \cup A^+(s_2))\} \quad (2)$$

else

$$V_2 \leftarrow V_2 \cup \{t_i\} \cup \{t_{i'} : t_{i'} \in T \text{ and } (t_i, t_{i'}) \in (A^+(s_1) \cup A^+(s_2))\} \quad (3)$$

C-3. Delete all tasks inserted into V_1 or V_2 from T.

In (2), t_i and all of its predecessors that remain in T are inserted in V_1. Equivalently, t_i and all of its successors that remain in T are inserted in V_2 in (3). It is not difficult to see that V_1 and V_2 correspond to the required partition when $T = \emptyset$.

Finally, the crossover of s_1 and s_2 generates the two offsprings s_1' and s_2' from s_1 and s_2. Let the scheduling s_1' be the same as s_1 for all tasks in V_1. On the other hand, the remaining tasks (those in V_2) are scheduled according to a list heuristic run over the graph $\mathcal{D}(s_2)$, called *earliest date/most immediate successors first* (ED/MISF), defined by the following rules.

R_1: compute the minimal introduction date of each free task. This is computed in function of the precedence constraints and in function of the schedule of tasks previously scheduled. Choose the task with smallest introduction date, say t_i. In case of several possibilities, choose the one with more successors. In case of several possibilities, choose at random.

R_2: choose a processor at random among the processors where the task t_i can be scheduled as soon as possible.

The generation of s_2' is analogous, with the tasks in V_2 being scheduled under the constraints in $\mathcal{D}(s_1)$. Feasibility of both s_1 and s_2 is guaranteed since both $\mathcal{D}(s_1')$ and $\mathcal{D}(s_2')$ are acyclic.

3.3.3 Mutation

Let s be an individual to which the operator mutation is to be applied. The first step in a mutation is to construct the digraph $\mathcal{D}(s) = (\mathcal{T}, A(s))$. Then, the new individual is formed by using a list heuristic. The rules used are the same as the crossover, where R_1 is modified such that the minimal introduction dates of the tasks are computed exclusively in function of the precedence constraints. This can be performed just once at the beginning of the operation.

4 Analysis of CGL

In this section, we analyze positive features of CGL related to its ability to generate all feasible schedules. The first such a feature is mentioned in [3] and is pointed out in the following lemma. It says that the new initial population generation scheme garantees that every feasible schedule can be constructed with a list heuristic and I-1 and I-2 define a random list heuristic.

Lemma 1 *Let s be a feasible schedule and P be an initial population generated with I-1 and I-2. Then, the probability of P to contain s is greater than zero.*

Proof. We prove the lemma if we show that every feasible schedule s can be generated with the random list heuristic. In this sense, consider the partially ordered set $(\mathcal{T}, A(s))$. Then, let us construct a linear extension of $(\mathcal{T}, A(s))$ by adding to $A(s)$ the pairs (t_{i_1}, t_{i_2}) such that the introduction date of t_{i_1} in s is less than or equal to the introduction date of t_{i_2} in s. If some tasks remain incomparable, add the pairs (t_{i_1}, t_{i_2}) such that $i_1 < i_2$. It is clear that this total order of the tasks can be randomly generated with some probability greater than zero. Therefoe, each task t_i can be randomly scheduled to $p(t_i, s)$ with some probability greater than zero. Consequently, by the definition of the linear extension of $(\mathcal{T}, A(s))$, the rank of each t_i is $r(t_i, s)$ and the lemma follows. □

A stronger result than Lemma 1 concerning the ability of CGL to generate all feasible solutions can be stated, indicating that the crossover operator is also able to generate every feasible schedule. Before stating this stronger result, let us define some more notation. Since any feasible schedule s can be generated by a list heuristic, let Σ_L be a sequence of task schedulings of a list heuristic L that generates s. Each element in this sequence corresponds to the task and to the processor chosen in an iteration of L. Such a sequence Σ_L of task schedulings *covers* a processor p_j if there exists a scheduling of some task on p_j in Σ_L. Given a closed task set V_1 and two list heuristics L_1 and L_2, define $\Sigma_{L_1,L_2}(V_1, V_2)$ as a sequence of task schedulings in which the first $|V_1|$ elements correspond to the scheduling of the tasks in V_1 according to L_1, while the remaining elements correspond to the scheduling of the tasks in $V_2 = V - V_1$ according to L_2. This definition is illustrated in Figure 1. The following fact states some necessary conditions to every feasible schedule may be generated by an ED/MISF heuristic.

Fact 1 *Let s be a feasible schedule, L_1 be a list heuristic that generates s and L_2 be an ED/MISF list heuristic. Then, $\Sigma_{L_1} = \Sigma_{L_1,L_2}(V_1, V_2)$ if the following conditions hold at the begining of each*

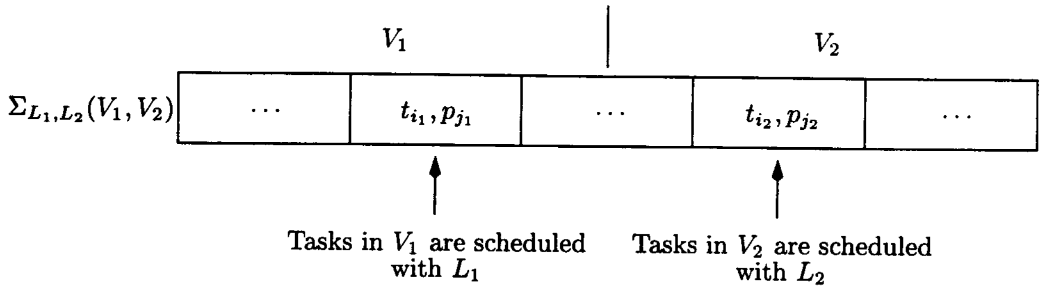

Figure 1: Sequence of tasks obtained with two different list heuristics.

iteration k of L_1 (suppose that task $t_i \in V_2$ is scheduled on p_j in this iteration):

F-1. for every free task $t_{i'} \in F(k)$, the minimum introduction date of $t_{i'}$ is greater than or equal to the introduction date of t_i on p_j, and

F-2. the introduction date of t_i on p_j is minimum over all processors and all tasks in $F(k)$.

It follows from the definition of ED/MISF that the conditions above are both necessary for some ED/MISF execution to choose the same task (condition F-1) and processor (condition F-2) than L_1 at each iteration. Now, suppose that $\Sigma_L \neq \Sigma_{L,ED/MISF}(V_1, V_2)$, for some list heuristic L. In the following lemma we state, based on Fact 1, that the probability of $\Sigma_L = \Sigma_{L,ED/MISF}(V_1, V_2)$ becomes greater than zero if we add some arcs to the DAG when using the ED/MISF heuristic to schedule the tasks in V_2. In order to state this result more formally, we need some more notation. Let $\Sigma_L(V_1)$ and $\Sigma_L(V_2)$ be the sequence of task schedulings defined by the tasks in V_1 and V_2, respectively, and their order in Σ_L. Equivalently, $\Sigma_{L,ED/MISF}(V_1, V_2, V_1)$ and $\Sigma_{L,ED/MISF}(V_1, V_2, V_2)$ stand for the sequence of task schedulings defined by the tasks in V_1 and V_2, respectively, and their order in $\Sigma_{L,ED/MISF}(V_1, V_2)$. We suppose that $\Sigma_L(V_1) = \Sigma_{L,ED/MISF}(V_1, V_2, V_1)$ and that $\Sigma_L(V_1)$ covers all processors, and we determine the arcs that must be included in the DAG in order to obtain $\Sigma_L(V_2) = \Sigma_{L,ED/MISF}(V_1, V_2, V_2)$ with some probability greater than zero. Notice that if any arc is not included in the DAG, then

$$Pr\left((\Sigma_L(V_2) = \Sigma_{L,ED/MISF}(V_1, V_2, V_2)) \mid (\Sigma_L(V_1) = \Sigma_{L,ED/MISF}(V_1, V_2, V_1))\right) = 0$$

because we have supposed $\Sigma_L \neq \Sigma_{L,ED/MISF}(V_1, V_2)$. This situation is illustrated in Figure 2.

Lemma 2 *Consider an extended DAG defined from the original one including the arcs (t_{i_1}, t_{i_2}) such that t_{i_2} belongs to V_2 and one of the following conditions holds:*

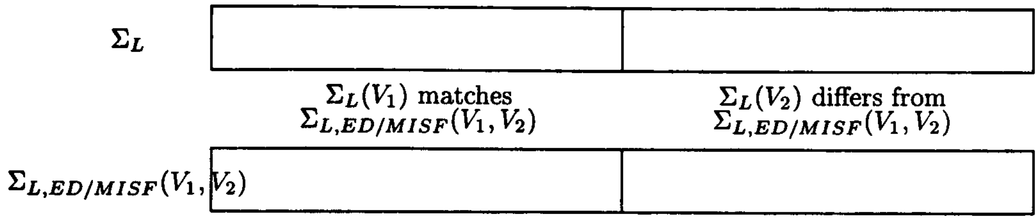

Figure 2: Two sequences of task schedulings that match for a closed task set V_1 but differ for $V - V_1$.

L-1. $a_{i_1} = a_{i_2} - 1$ *and* $b_{i_1} > b_{i_2}$*, where* a_{i_1} *and* a_{i_2} *are the positions of* t_{i_1} *and* t_{i_2} *in* Σ_L*, respectively, and* b_{i_1} *and* b_{i_2} *are the positions of* t_{i_1} *and* t_{i_2} *in* $\Sigma_{L,ED/MISF}(V_1, V_2)$*, respectively; or*

L-2. $p(t_{i_1}, s) = p(t_{i_2}, s)$ *and* $p(t_{i_2}, s_1) \neq p(t_{i_2}, s)$*, where* s *and* s_1 *stand for the schedules corresponding to* Σ_L *and* $\Sigma_{L,ED/MISF}(V_1, V_2)$*, respectively.*

Then,

$$Pr\left((\Sigma_L(V_2) = \Sigma_{L,ED/MISF}(V_1, V_2, V_2)) \mid (\Sigma_L(V_1) = \Sigma_{L,ED/MISF}(V_1, V_2, V_1))\right) > 0,$$

when $ED/MISF$ *is applied using the extended DAG in* $\Sigma_{L,ED/MISF}(V_1, V_2, V_2)$*.*

The position of a task t in a sequence Σ is the number of task schedulings appearing in Σ before t. Figures 3 and 4 illustrate two situations corresponding to Lemma 2.

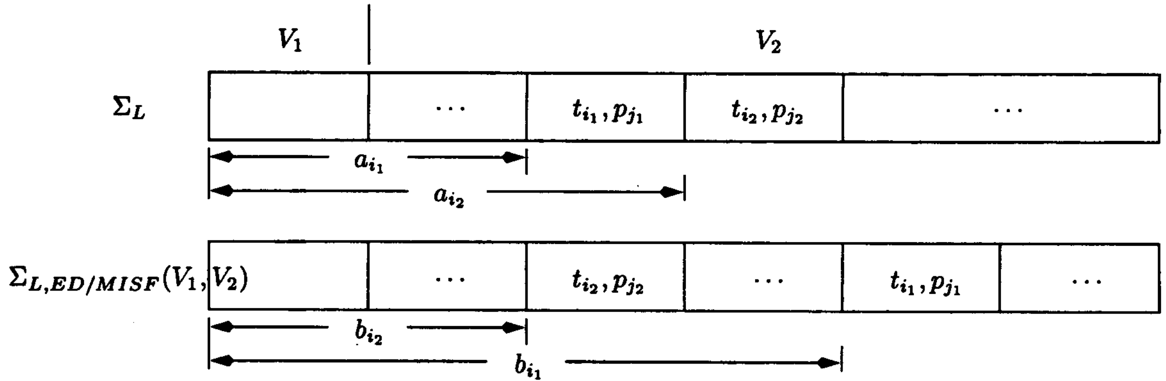

Figure 3: The arc (t_{i_1}, t_{i_2}) does not belong to the DAG but it is included in the extended DAG due to condition L-2 of Lemma 2.

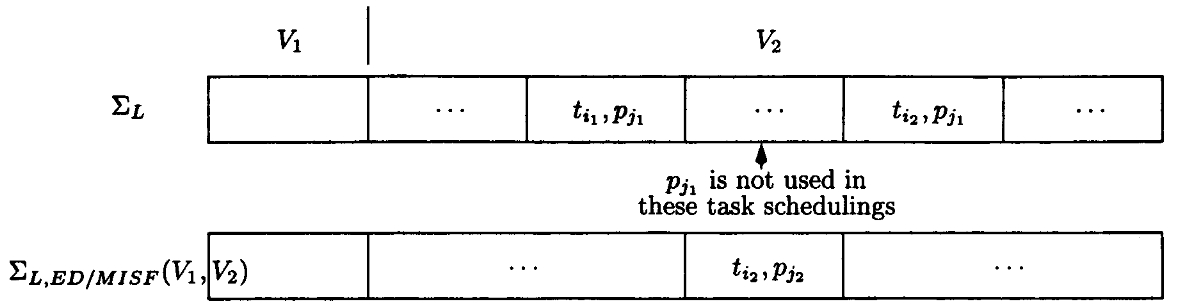

Figure 4: The arc (t_{i_1}, t_{i_2}) does not belong to the DAG but it is included in the extended DAG due to condition L-1 of Lemma 2.

Proof. We prove that Σ_L and $\Sigma_{L,ED/MISF}(V_1, V_2)$ are equivalent with some probability if we prove that, with some probability, $a_i = b_i$ and $p(t_i, s) = p(t_i, s_1)$ for every $t_i \in V_2$ when $ED/MISF$ is applied using the extended DAG in $\Sigma_{L,ED/MISF}(V_1, V_2)$. Let us show that $a_i = b_i$ for all i with probability 1 by contradiction. Let a_i be the smallest index such that $a_i \neq b_i$. In this case, there exists a task $t_{i'} \in V_2$ such that $b_{i'} = a_i$, $b_{i'} < b_i$ and $a_{i'} > a_i$. Then, two cases are possible. First, $a_i = a_{i'} - 1$ which by condition L-1 of the lemma yields that $(t_i, t_{i'})$ belongs to the extended graph, which is a contradiction with $b_{i'} < b_i$. Otherwise, $a_i < a_{i'} - 1$, and let $t_{i''}$ be a task such that $a_{i''} = a_{i'} - 1$. Since $b_{i'} < b_{i''}$, condition L-1 forces $(t_{i''}, t_{i'})$ to belong to the extended graph, which is a contradiction.

Finally, the probability we search for is given by

$$1 - Pr(p(t_i, s) \neq p(t_i, s_1), \text{ for some } t_i \in \mathcal{T})$$

which is equivalent to

$$Pr(p(t_i, s) = p(t_i, s_1), \text{ for all } t_i \in \mathcal{T}).$$

In order to prove that this latter probability is greater than zero, suppose by contradiction a task t_i such that $p(t_i, s) \neq p(t_i, s_1)$. Since $t_i \in V_2$ and $\Sigma_L(V_1)$ covers all processors, then there exists a task $t_{i'}$ such that $a_{i'} < a_i$ and $p(t_{i'}, s) = p(t_i, s)$. Suppose that $p(t_{i'}, s) = p(t_{i'}, s_1)$ (otherwise, take $t_i = t_{i'}$). Consequently, by condition L-2, $(t_{i'}, t_i)$ belongs to the extended graph. Consider now the iteration k of the ED/MISF heuristic in which t_i is scheduled on $p(t_i, s_1)$. Since $(t_{i'}, t_i)$ belongs to the extended graph and $p(t_{i'}, s_1) = p(t_{i'}, s)$, then t_i may be scheduled on $p(t_i, s)$ instead of on $p(t_i, s_1)$ without increasing its introduction date. This yields that the k-th element of $\Sigma_{L,ED/MISF}(V_1, V_2)$

may be modified to t_i scheduled on $p(t_i, s)$, which is a contradiction with the assumption that such a sequence does not exist. □

Based on this lemma and on Fact 1, we can state the following theorem, which says that a CGL execution also may generate all feasible schedules in any generation.

Theorem 1 *Let s be a feasible schedule and P be a generation after the initial population in a CGL execution. Then, the probability of P to contain s given that the generation previous to P does not contain s is greater than zero.*

Proof. The proof is by induction on the generations. For the initial population $P = P_0$, the Lemma 1 applies. Suppose that the theorem is valid for any feasible schedule and any generation $P = P_{i-1}$, for all $i > 0$. We investigate the probability of the generation P_i to contain any feasible schedule, say s. Let L be a list heuristic that generates s. We concentrate the proof on the crossover by supposing that any mutation does not occur between P_{i-1} and P_i, this without loss of generality. So, P_i is obtained from P_{i-1} by a selection followed by crossovers.

Suppose two individuals s_1 and s_2 in P_{i-1} which crossover. By the induction hypothesis, such s_1 and s_2 exist in P_{i-1} with some probability greater than zero, but we have that $s_1, s_2 \neq s$. Also suppose a partition V_1, V_2 of V such that:

I. there is no dependency from a task in V_2 to a task in V_1 in the digraph $(\mathcal{T}, A(s_1) \cup A(s_2))$;

II. the first $|V_1|$ tasks in Σ_L are exactly those in V_1; and

III. V_1 covers all processors.

Let L_1 be the list heuristic that generates s_1, and assume that $\Sigma_L(V_1) = \Sigma_{L_1}(V_1)$. The remaining of the proof is divided into two parts. We show in the first part that there exists a feasible schedule s_2 such that $A(s_2)$ extends A with the arcs needed to find the desired equivalence between Σ_L and $\Sigma_{L,ED/MISF}(V_1, V_2)$, this based on Lemma 2. For the second part, we must show that, for every arc (t_{i_1}, t_{i_2}) added to the DAG as above, t_{i_1} and t_{i_2} are scheduled on the same processor and with the same rank in s_2 as in s with some probability greater than zero. This because of the rules defined to the crossover operator.

For the first part, we exhibit an s_2 which corresponds to the above requirements, based on s. Let us consider the following algorithm, which builds a sequence Σ_{L_2} of task schedulings step by step.

1. Set $\Sigma L_2 = \Sigma_L(s)$. The processor of some positions may be altered in the remaining steps.

2. For every position k from $|V_1|+1$ until n, if the tasks in position k in Σ_L and $\Sigma_{L,ED/MISF}(V_1, V_2)$ differ then modify ΣL_2 as follows. Let t_i be the task at position k in $\Sigma_{L,ED/MISF}(V_1, V_2)$. Set the processor at position k to the processor at the position previous to the position of t_i in Σ_L.

3. For every position k from $|V_1|+1$ until n, if the tasks in position k in Σ_L and $\Sigma_{L,ED/MISF}(V_1, V_2)$ match but the processors differ then modify ΣL_2 as follows. Let t_i be the task scheduled on the same processor and immediately before the task at position k in Σ_L. Set the processor at position k in ΣL_2 to the processor on which t_i is scheduled in ΣL_2.

Clearly, s_2 is feasible since the order of the tasks in ΣL_2 respects the order of the tasks in Σ_L. By the induction hypothesis, s_2 may be generated in P_{i-1}. We conclude the proof of the theorem showing that $A(s_2)$ is such that every arc added to the DAG following L-1 and L-2 of Lemma 2 belongs to $A(s_2)$. Notice that this fact yields the desired equivalence between Σ_L and $\Sigma_{L,ED/MISF}(V_1, V_2)$ with some probability greater than zero since $\Sigma_L(V_1) = \Sigma_{L_2}(V_1)$.

Consider an arc of the extended DAG included because of L-1. Then, in the algorithm of ΣL_2, step 2 is executed at $k = a_{i_1}$, being t_{i_1} and t_{i_2} the tasks at positions k and $k+1$ in Σ_L, respectively. Consequently, t_{i_1} and t_{i_2} are scheduled on the same processor, which yields $(t_{i_1}, t_{i_2}) \in A(s_2)$ (see Figure 3. The arcs of the extended DAG included because of L-2 belong to $A(s_2)$ due to step 3 of the algorithm that builds ΣL_2. □

5 Conclusion

As empirically tested in [3], the integration of knowledge about the multiprocessor scheduling problem, through the use of a list heuristic in knowledge-augmented crossovers and mutations, helps to dramatically improve the quality of the solutions that can be obtained with both a pure genetic and a pure list approaches. One of the arguments for this improvement used in [3] is the ability of CGL to consider all feasible schedules in the search. In this paper, we have shown that this property is indeed valid for the randomly generated initial population, as well as schedules generated by crossovers.

References

[1] T. Casavant and J. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2), February 1988.

[2] E. Coffman. *Computer and Job-Shop Scheduling Theory.* Wiley, New York, 1976.

[3] R. Corrêa, A. Ferreira, and P. Rebreyend. Integrating list heuristics into genetic algorithms for multiprocessor scheduling. In *IEEE Symposium on Parallel and Distributed Processing*, New Orleans, USA, October 1996.

[4] M. Cosnard and D. Trystram. *Parallel Algorithms and Architectures.* International Thomson Computer Press, 1995.

[5] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Mass., 1989.

[6] J. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Mass., 1992.

[7] B. Malloy, E. Lloyd, and M. Soffa. Scheduling DAG's for asynchronous multiprocessor execution. *IEEE Transactions on Parallel and Distributed Systems*, 5(5), May 1994.

[8] M. Norman and P. Thanisch. Models of machines and computations for mapping in multicomputers. *ACM Computer Surveys*, 25(9):263–302, Sep 1993.