International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014, pp.102-115
ISSN 2356-5608

Heuristics for Scheduling Independent Tasks on Heterogeneous Processors under Limited Makespan Constraint

Rim Somai & Zaher Mahjoub
Department of Computer Science
University of Tunis El Manar, Faculty of Sciences of Tunis
University Campus, 2092 El Manar II, Tunis Tunisia

*ABSTRACT*__ **Scheduling independents tasks on heterogeneous processors under resource constraints is classified as a hard combinatorial optimization problem for which several solving heuristics are known in the literature. We address here a particular instance of this problem where the constraint corresponds to a fixed unfeasible makespan i.e. a makespan that does not allow to schedule all the input tasks with the available processors. Our objective is to schedule an optimal task subset extracted from the input task set where optimality may be defined according to diverse criteria. For this purpose, we propose two constructive approaches leading to 2-phase heuristics. An experimental study is achieved in order to evaluate the performances of the designed heuristics and validate our contribution.**

Keywords—Heterogeneous environment, heuristic, independent tasks, makespan, parallel system, POC, processor, RCPSP, resource constraint, scheduling.

I. INTRODUCTION

A. *The RCPSP Problem*

The resource constraint scheduling problem, known as RCPSP, consists in minimizing the completion time (makespan) for scheduling a set of tasks when the resources are not continuously available to perform all the processing. The RCPSP is classified as NP-hard [1] and has many variants [12] depending particularly on (i) inter-task precedence constraints (GPRCP and PRCP-GPR variants), (ii) task preemption/non preemption (PRCPSP), (iii) resource availability pattern (constant, variable) … This problem has been extensively studied because of its many real world applications e.g. in chemical industry where it is known as Labor Constrained Scheduling Problem [3], in timetable managing for university classrooms and teachers [2], in industrial sectors such as textile workshops [7] or hybrid flow-shop [10]. It has to be underlined that there exist exact algorithms providing optimal solutions but requiring exponential time [11]. An alternative consists in designing polynomial time approximation algorithms or heuristics. In this paper, we address a specific variant of the RCPSP where the available resources, a set of heterogeneous processors i.e. of different speeds, cannot handle all the tasks during a limited prefixed makespan. Our aim is to schedule an optimal subset of the input tasks. The optimality may be either a maximal number of tasks or a maximal weight, the weight being the sum of the costs of the subset tasks.

B. *Brief overview*

In a recent study [12] we find an updated and exhaustive overview of the RCPSP with nine extensions and solving methods for each. For the RCPSP with independent tasks in which we are interested, fifteen approaches are cited such as hybrid genetic algorithms, integer/constraint programming, 0-1 linear programming, branch and bound technique, hybrid neural network approach, local search techniques, swarm optimization...

Other methods are also known in the literature such as hybridization of two classic methods i.e. particle swarm optimization and genetic algorithm [5]. A hyperheuristic based on a tabu search is proposed in [9].

It has to be underlined that the majority of the solving techniques which are of polynomial complexity determine only approximate solutions. However, techniques such as 0-1 linear programming and branch and bound determine exact solutions but are of exponential complexity, thus cannot be used for large size RCPSP's. For this important reason, we preferred designing polynomial time approximation algorithms that are able to determine good quality solutions in an acceptable time.

The remainder of the paper is organized as follows. In section 2, we propose our theore-tical contribution consisting in two approaches i.e. Limited Makespan Constraint Relaxation (LMCR) and Progressive Task Allocation (PTA). Section 3 is devoted to an experimental

International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014, pp.102-115
ISSN 2356-5608

study. Finally, we present some concluding remarks and perspectives in section 4.

II. SCHEDULING UNDER LIMITED RESOURCES

A. *Problem formulation*

The problem we address can be formulated as follows. Consider n independent tasks $T_1 \dots T_n$ with respective costs $c_1 \dots c_n$ and p heterogeneous processors $P_1 \dots P_p$ whose respective speeds are $s_1 \dots s_p$ where $s_i \geq s_{i+1}$ (i=1…p-1). Let sc be the sum of task costs and ss the sum of processor speeds. We have to schedule the n tasks under the constraint of limited duration (makespan) M. The makespan M is assumed to be unfeasible i.e. M < sc/ss. In other words, the n tasks cannot be scheduled in such a makespan. Only a subset could be. Indeed, the makespan of any scheduling of the n tasks with the p processors, denoted M(p), obviously satisfies the following inequality, provided that no processor is useless [13]:

$$M(p) \geq sc/ss \qquad (1)$$

Therefore, we have to determine a maximal (or optimal) task subset extracted from the set of n tasks that can be scheduled with the p available processors. Here, the task subset to be determined may be defined in various ways. It may be a maximum number of tasks. It may also correspond to a maximal subset weight, the weight being the sum of the costs of the subset tasks.

From inequality (1) we directly get :

$$sc \leq M(p)_{\times}ss \qquad (2)$$

Hence, we deduce that the total cost (weight) of tasks that can be processed in a period M, denoted sc(M), and the overall speed, denoted ss(M), required to schedule the n tasks in a period M satisfy the following relationships:

$$sc(M) \leq M_{\times}ss = sc_{max} \qquad (3)$$

$$ss(M) \geq sc/M = ss^{+}(M) \qquad (4)$$

We can then proceed in several ways as detailed below. First, it should be noted that a processor load, denoted λ, is defined as the sum of the costs of the tasks assigned to it divided by its speed. We detail below our two approaches by considering three cases that may occur in practice: (i) unsorted (or random) tasks, (ii) tasks with decreasing costs ($c_1 \geq c_2 \geq \dots \geq c_n$) and (iii) tasks with increasing costs ($c_1 \leq c_2 \leq \dots \leq c_n$). Indeed, each case has its own peculiarities.

B. *First Approach: Limited Makespan Constraint Relaxation (LMCR)*

The LMCR approach is a direct consequence of inequality (3). It consists in relaxing, i.e. not considering, the limited makespan constraint. Thus we begin by scheduling all the n tasks regardless of the resulting makespan. Let M(p) be the makespan obtained by using a scheduling algorithm, say *Sch*. Obviously, we have M(p) > M. It suffices now to first identify the processors whose loads do not exceed M. Such processors will be called feasible. As to the remaining processors, called unfeasible, we eliminate for each of them as many tasks as necessary so that the new load does not exceed M. The eliminated tasks will be called redundant. Remark that the elimination procedure may be done in various ways as follows. Let us first denote by $T_{i,1} \dots T_{i,ki}$ the k_i tasks whose costs are $c_{i,1} \dots c_{i,ki}$ that are assigned to the unfeasible processor P_i. Let λ_i be its load i.e. $(c_{i,1} + c_{i,2} + \dots + c_{i,ki})/s_i$.

A first straightforward procedure consists in formulating, for each unfeasible processor, say P_i, the elimination of its redundant tasks as a problem of extracting from its k_i tasks a subset of k_{i*} ($<k_i$) tasks such that the sum of their costs divided by s_i, denoted λ_{i*}, is the closest to M without exceeding it. We may proceed here in several ways as follows.

(a) A trivial way consists in choosing the first *feasible* k_{i*} tasks. Obviously, this procedure can be achieved in an $O(k_i)$ time in the worst case. We easily deduce that the overall complexity (i.e. to treat all the unfeasible processors) is O(np) in the worst case since $k_i \leq n$, i=1…p.

(b) A variant of the above alternative is to first sort the k_i tasks $T_{i,1} \dots T_{i,ki}$ in increasing (resp. decreasing) cost, then extract the first k_{i*} tasks i.e. such that the corresponding load λ_{i*} is the closest to M without exceeding it. Clearly, the increasing (resp. decreasing) sort would maximize (resp. minimize) k_{i*} thus the total number of tasks scheduled by P_i. This variant is obviously of complexity $O(k_i \log k_i)$ at most. The overall complexity (i.e. to treat all the unfeasible processors) will then be O(nplogn) in the worst case.

(c) Another alternative consists in extracting, with no presorting, an optimal sublist of non necessarily consecutive tasks. Unfortunately, this extraction is a hard combinatorial optimization problem (POC) thus an exponential time is required to determine an optimal solution. However, polynomial approximation algorithms for solving this problem are known in the literature [4]. We have to add that an approximate solution may be easily determined when the task costs are initially in decreasing (resp. increasing) order. In such case, the deleted tasks will be those with smaller (resp. larger) costs. Thus, as previously seen in (b), this procedure would minimize (resp. maximize) the number of scheduled tasks and would maximize (resp. minimize) the total sum of scheduled task costs. Remark that as will be seen in section 3 when describing the experiments we achieved, the decreasing order maximizes, in most cases, the total weight i.e. the sum of the costs of the task subset.

Concerning the above mentioned scheduling algorithm, denoted *Sch*, it may be one among the standard algorithms known in the literature i.e. List Scheduling (LS), Shortest Processing Time (SPT), Longest Processing Time (LPT) with their two versions. Their complexities are as follows. For LS, it is O(np) (may be reduced to O(nlogp)) and for both SPT and LPT, it is O (n(logn+logp)) [6].

We have to add that, once the task elimination procedure is done, a second and improving phase may be considered. It consists in reallocating the eliminated tasks by first identifying the non saturated processors i.e. whose loads do not exceed the fixed makespan M. Let n* (< n) be the number of scheduled tasks. The reallocation procedure of the n-n* remaining tasks may be done in several ways as follows.

(d) The current task is allocated to the first non saturated processor that fits it i.e. whose new load does not exceed M. The procedure is iterated until no processor can receive a new task. Its complexity is clearly O((n-n*)p) in the worst case i.e. at most O(np).

(e) The current task is allocated to the best non saturated processor that fits it i.e. whose new load will be the closest to M without exceeding it. The procedure is iterated until no processor can receive a new task. Here the complexity is the same as above.

(f) The n-n* tasks are first sorted in increasing or decreasing costs and one of the two previous procedures is used. The complexity is here O((n-n*)(log(n-n*)+p) in the worst case i.e. at most O(n(logn+p)).

It is easy to remark that the three proposed ways (d), (e) and (f) are in fact based on standard heuristics known for solving a variant of the 1D Bin Packing problem where the number of bins is fixed and their capacities are different [7].

The LMCR approach is illustrated with the following example where n=10 ; p=3 ; costs: 7,4,5,1,10,6,3,2,8,9 (hence sc=55) ; speeds : 1,0.5,0.25 (hence ss=1.75) ; M=23<sc/ss=55/1.75= =31.43.
Thus $sc(M) \leq sc_{max}=M_{\times}ss=23_{\times}1.75=40.25$ and $ss(M) \geq ss^{+}(M)=sc/M=55/23=2.39$. We apply our approach by using algorithms LS, SPT and LPT. The different results are then compared.

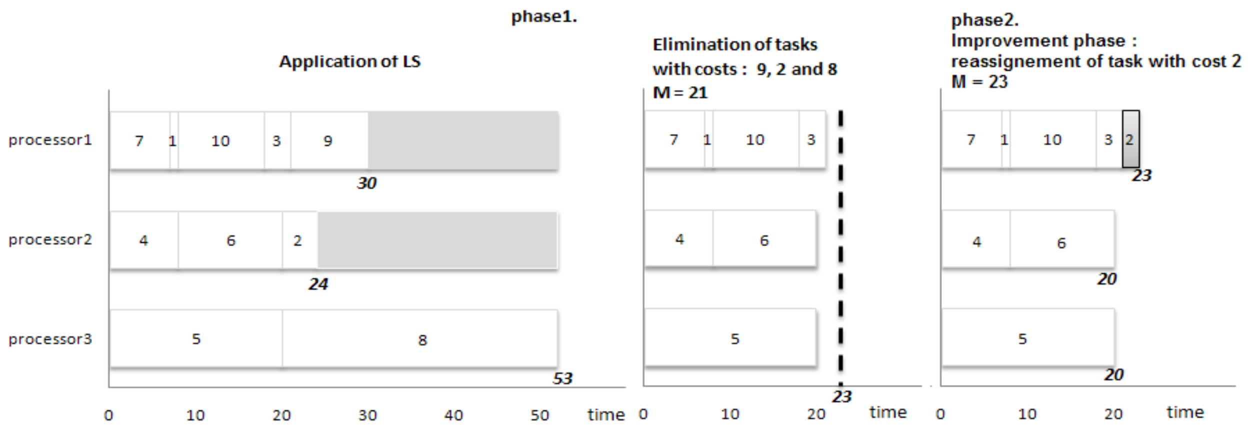

Fig 1. LMCR approach using LS-version 1

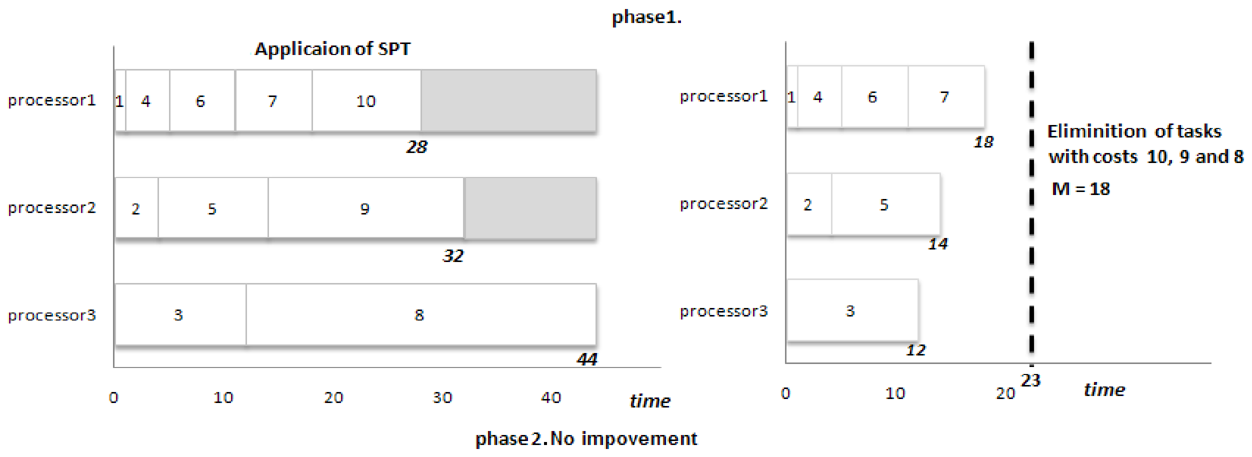

Fig 2. LMCR approach using SPT-version 1

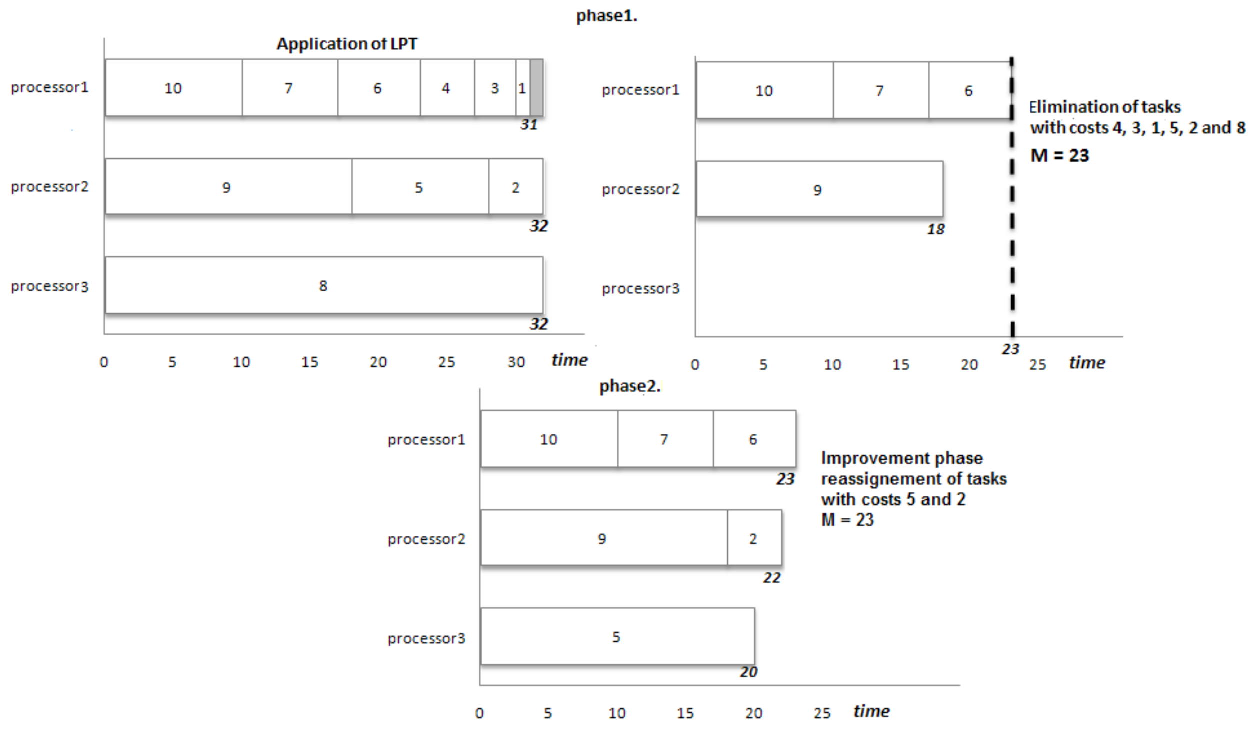

Fig 3. LMCR Approach using LPT-version 1

The next table gives, for each algorithm, n* the number of scheduled tasks, their weight sc* (sum of their costs) and the final makespan M* (≤ M=23). We also give the corresponding ratios n*/n, sc*/sc, M*/M and sc*/scmax (%).

TABLE I. RESULTS COMPARISON - LMCR

Sch	LS	SPT	LPT
n*	8	7	6
sc*	38	28	39
M*	23	18	23
n*/n	80	70	60
sc*/sc	69.1	50.9	70.9
M*/M	100	78.3	100
sc*/ sc_{max}	94.40	69.13	96.89

We remark that LS gave the maximum number of tasks (8 out of 10) but neither the maximum nor the minimum weight. LPT gave the minimum number of tasks (6 out of 10) and the maximum weight (39 out of 55). Both LS and LPT reached the fixed makespan. As to SPT, it gave the worst results.

On the other hand, we may define the efficiency E of each algorithm by the ratio sc*/ scmax. We remark that it is equal to 94.40% for LS, 69.13 % for SPT and 96.89 for LPT (the best).

C. Second Approach: Progressive Task allocation (PTA)

This approach consists first in applying a scheduling algorithm Sch (LS, SPT or LPT) to schedule the whole n tasks such that the (transient) load of each processor does not exceed M.

Assigning a task Ti to a processor Pj considered by Sch as the first feasible candidate is done under the assumption that its new load does not exceed M i.e. λj_new ≤ M with λj_new = λj_old + ci/sj. If this condition is not satisfied, this task will be assigned to the second feasible processor (according to Sch) with the same previous assumption. If there is no feasible processor at all, the task in question will be definitively removed (eliminated).

Once this phase is terminated, an improvement phase may be considered as in the previous LMCR approach. It consists in reallocating the eliminated tasks according to one among the alternatives (d), (e) or (f) seen in the LMCR approach.

As to the overall complexity of PTA, in addition to that of *Sch* (either O(nlogp) or O(n(logn+logp)), we have to add either O(np) or O(n(logn+p)) in the worst case. This gives a worst case complexity of O(np) to O(n(logn+p)).

The PTA approach is illustrated with the previous example. The results are recapitulated in the following table. We can remark that quite similar results to LMCR are obtained.

TABLE II. RESULTS COMPARISON - PTA

	LS	SPT	LPT
n*	8	7	7
sc*	38	28	39
M*	23	18	23

To conclude this section, we recapitulate the two approaches with their alternatives and complexities below.

TABLE III. RECAPITULATIVE TABLE OF APPROACHES

Approach	**Phases**		**Alternatives**	**Complexity**
LMCR	1	Apply a scheduling algorithm *Sch*		Complexity of *Sch*
		Identify unfeasible processors		O(p)
		Elimination of non assigned tasks	(a)	O(np)
			(b)	O(nplogn)
			(c)	exponential
	2	Improvement reallocation	(d)	O(np)
			(e)	
			(f)	O(n(logn+p))
PTA	1	Apply *Sch* without exceeding M by choosing an adequate processor and eliminate unfeasible tasks		Complexity of *Sch*
	2	Improvement reallocation	as in LMCR	

III. EXPERIMENTATIONS AND COMPARISONS

A. *Introduction*

In order to validate our theoretical contribution, we present in this section an experimental study achieved on a series of random data. Our algorithms were coded in Python 2.6 under Windows XP SP3. The target machine is an Intel Toshiba with 1.60 GHz clock and 1 GB RAM.

We precise that we choosed 9 values for n in the range [10 500]. For each n, we generated two sets of task costs : identical costs and different costs with three cases i.e. random costs (rac), increasing costs (ic) and decreasing costs (dc). On the other hand, for each value of n, we choosed three values for p (varying with n). For each value of p, we choosed two values for $m=\max(s_i)/\min(s_i)$ i.e. 5 and 10 then randomly generated the speeds. We mention that the processors were sorted by decreasing speeds. Concerning the choice of the values of both p and M, we have to precise that is was done as follows. We first experimentally determine $M(p^+)$ i.e. the makespan required to schedule the n tasks with p^+ processors by using algorithm LPT-version 2 (LPTb). We then choose $p < p^+$ and $M < M(p)$. In the experiments presented below, we used the LS algorithm with its two versions LS1 and LSb i.e. LS with back filling.

B. *Numerical results*

In addition to the previously defined parameters n, sc, p, ss, M and m, we denote by ss+(M)=sc/M (> ss) the minimal global processor speed sum required to process the n tasks in M time, $sc_{max}=M*ss$ the maximal weight (sum of costs of the scheduled tasks) that can be reached, n* (<n) the number of scheduled tasks, sc* the sum of scheduled tasks costs, the efficiency $E=100(sc^*/ sc_{max})$, the n-ratio nr=100(n*/n) and the cost-ratio cr=100(sc*/sc).

We give below excerpts of the numerous results restricted to n = 200 and sc = 9718. Twelve tests are presented.

(a) LMCR

We precise that we used alternative (a) in phase 1 and alternative (d) in phase 2.

International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014, pp.102-115
ISSN 2356-5608

TABLE IV. LMCR WITH LS FOR N = 200 AND SC=9718

p	ss	M	$ss^+(M)$	sc_{max}	**rac**					**ic**					**dc**				
					n*	nr	sc*	cr	E	n*	nr	sc*	cr	E	n*	nr	sc*	cr	E
16	64	66.08	150.3	4229.56	103	51.5	4156	42.76	98.26	126	63	3710	38.17	87.71	72	36	4218	43.40	99.72
37	250.4	33.55	295.3	8402.31	176	88	7778	80.03	92.56	172	86	7080	72.85	84.26	158	79	8320	85.61	99.02
42	163.5	42.08	236.9	6880.62	158	79	6548	67.38	95.16	152	76	5454	56.12	79.26	130	65	6679	68.72	97.06
55	380.1	22.78	444	8657.83	175	87.5	7653	78.75	88.39	166	83	6562	67.52	75.79	167	83.5	8483	87.29	97.98
177	541.3	21.25	557.7	11502.62	69	34.5	2128	21.89	18.50	148	74	5156	53.05	44.82	200	100	9718	100	84.48
122	881	10.21	1062	9001.52	69	34.5	2122	21.83	23.57	144	72	4874	50.15	54.14	27	13.5	1755	18.05	19.49

TABLE V. LMCR WITH LSB FOR N = 200 AND SC=9718

p	ss	M	$ss^+(M)$	sc_{max}	**rac**					**ic**					**dc**				
					n*	nr	sc*	cr	E	n*	nr	sc*	cr	E	n*	nr	sc*	cr	E
16	64	66.08	150.3	4229.56	101	50.5	4209	43.31	99.51	126	63	3710	38.17	87.71	73	36.5	4209	43.31	99.51
37	250.4	33.55	295.3	8402.31	180	90	8242	84.81	98.09	167	83.5	6642	68.34	79.04	151	75.5	8320	85.61	99.02
42	163.5	42.08	236.9	6880.62	153	76.5	6703	68.97	97.41	150	75	5300	54.5	77.02	130	65	6724	69.19	97.72
55	380.1	22.78	444	8657.83	183	91.5	8175	84.12	94.42	162	81	6223	64.03	71.87	161	80.5	8542	87.89	98.66
177	541.3	21.25	557.7	11502.62	172	86	7165	73.72	62.29	137	68.5	4399	45.26	38.24	200	100	9718	100	84.48
122	881	10.21	1062	9001.52	171	85.5	7083	72.88	78.68	134	67	4205	43.27	46.71	170	85	8575	88.23	95.26

By comparing the obtained values for nr and cr ratios with the two versions LS and LSb, we remark that, in most cases, LSb is better. This is due to the fact that LSb tries to fill holes i.e. saturate the processors. Thus LSb will be more likely to handle a larger number of tasks.

Concerning nr, we notice that for the case where the tasks are sorted by decreasing cost (dc), we obtained the worst results. This can be explained by the fact that the allocation of tasks with larger costs is done first. This leads to reach more quickly the threshold M i.e. the processors are (nearly) saturated with a small number of tasks. The results obtained with LMCR and its different versions are generally enough satisfactory because we could schedule in most cases practically all the tasks.

(b) PTA

Here alternative (d) was used in phase 2.

TABLE VI. PTA WITH LS1 FOR N = 200 AND SC=9718.

p	ss	M	$ss^+(M)$	sc_{max}	**rac**					**ic**					**dc**				
					n*	nr	sc*	cr	E	n*	nr	sc*	cr	E	n*	nr	sc*	cr	E
16	64	66.08	150.3	4229.56	105	52.5	4161	42.81	98.37	126	63	3710	38.17	87.71	55	27.5	4222	43.44	99.82
37	250.4	33.55	295.3	8402.31	174	87	7782	80.07	92.61	170	85	6901	71.01	82.13	138	69	8378	86.21	99.71
42	163.5	42.08	236.9	6880.62	152	76	6293	64.75	91.45	151	75.5	5373	55.28	78.08	116	58	6849	70.47	99.54
55	380.1	22.77	444	8657.83	176	88	7860	80.88	90.78	165	82.5	6471	66.58	74.74	155	77.5	8608	88.57	99.42
177	541.3	21.25	557.7	11502.62	163	81.5	6415	66.01	55.76	148	74	5156	53.05	44.82	200	100	9718	100	84.48
122	881	10.21	1062	9001.52	162	81	6333	65.16	70.35	144	72	4874	50.1 5	54.14	145	72.5	8425	86.69	93.59

TABLE VII. PTA WITH LSB FOR N = 200 AND SC=9718

p	ss	M	$ss^+(M)$	sc_{max}	**rac**					**ic**					**dc**				
					n*	nr	sc*	cr	E	n*	nr	sc*	cr	E	n*	nr	sc*	cr	E
16	64	66.08	150.3	4229.56	108	54	4121	42.40	97.43	126	63	3710	38.17	87.71	55	27.5	4222	43.44	99.82
37	250.4	33.55	295.3	8402.31	171	85.5	7556	77.75	89.92	167	83.5	6642	68.34	79.04	144	72	8383	86.26	99.77
42	163.5	42.08	236.9	6880.62	153	76.5	6214	63.94	90.31	150	75	5300	54.53	77.02	116	58	6849	70.47	99.54
55	380.1	22.77	444	8657.83	171	85.5	7365	75.78	85.06	162	81	6223	64.03	71.87	154	77	8606	88.55	99.40
177	541.3	21.25	557.7	11502.62	153	76.5	5690	58.55	49.46	137	68.5	4399	45.26	38.24	200	100	9718	100	84.48
122	881	10.21	1062	9001.52	148	74	5336	54.90	59.27	134	67	4205	43.27	46.71	80	40	5348	55.03	96.44

We can remark that, regardless of task costs configuration (rac, ic, dc), LSb gives better results than LS. On the other hand, LS in the rac and dc cases always gives a minimum number of scheduled tasks.

We can explain this bad performance in the dc case, as seen for LMCR, by the fact that here tasks with larger costs are firstly scheduled i.e. the processors are (nearly) saturated with a small number of tasks.

If we examinate the results for cr ratio, we notice that the best and worst results are not always obtained with the same versions as it is the case for the nr ratio. The two versions (LS and LSb) with tasks in the ic case often provide the smallest weights (sc*). Indeed, these versions allocate lower cost tasks to the least loaded processors. Therefore, it is less likely to increase the weight of scheduled tasks. This behavior leads to increase the nr ratio (so n*) at the expense of the cr ratio (so sc*). In the rac case (random costs), the best results for n* and sc* seem to depend more on the task costs than on the algorithm itself.

C. *Inter-approaches comparative study*

For a total of 114 tests and each task cost case (rac, ic, dc), we specify in table 8 the number of times (%) where an algorithm gave the best result in the two approaches. We precise that a result obtained by a given algorithm is considered the best when it corresponds to the highest values for n* or sc*. For instance, in Table 8, LS1-rac giving 14.91 (%) for n* means that it was the best in 14.91 % of the 114 cases (i.e. 17/114).

We also give the efficiency variation interval (E interval) reached by an algorithm in the 114 tests and the number of times (%) where the efficiency exceeded 70% (E>70%).

TABLE VIII. RATIOS (%) OF THE BEST ALGORITHMS

Version	**LMCR**				**PTA**			
	*n**	*sc**	*E interval*	*E >70%*	*n**	*sc**	*E interval*	*E >70%*
LS1-rac	14.91	6.14	[7.08 99.84]	73.01	**49.12**	5.26	[55.76 99.73]	98.47
LS1-ic	32.45	0.87	[43.97 97.75]	59.52	29.82	0.87	[43.97 97.20]	53.50
LS1-dc	14.03	36.84	[0. 99.94]	92.06	28.94	**90.35**	[82.02 99.93]	100
LSb-rac	**37.71**	21.05	[60.65 99.86]	93.65	14.91	6.14	[49.46 99.86]	82.45
LSb- ic	26.31	0.87	[35.87 97.75]	50.79	25.43	0.87	[35.87 97.20]	50.87
LSb-dc	23.68	**63.15**	**[81.51 99.94]**	**100**	24.56	72.80	**[82.02 99.93]**	**100**

We remark that the two approaches give enough similar results for n* and sc*. LS1-ic gives good values for n* but bad ones for sc*. We also notice that in order to maximize n*, we have better using LS1-ic version in the two approaches. But, to maximize sc*, LSb-dc in LMCR as well as LS1-dc in PTA are the best. Concerning the efficiency, it was larger than 70% in more than half of the cases and the LSb-dc is the best.

In order to better appreciate the performances of the different algorithms used in the two approaches, we present in the following figures the cost ratio cr and efficiency E profiles where cr and E sorted in increasing order (notice that sorting E and cr does not lead to the same permutations).

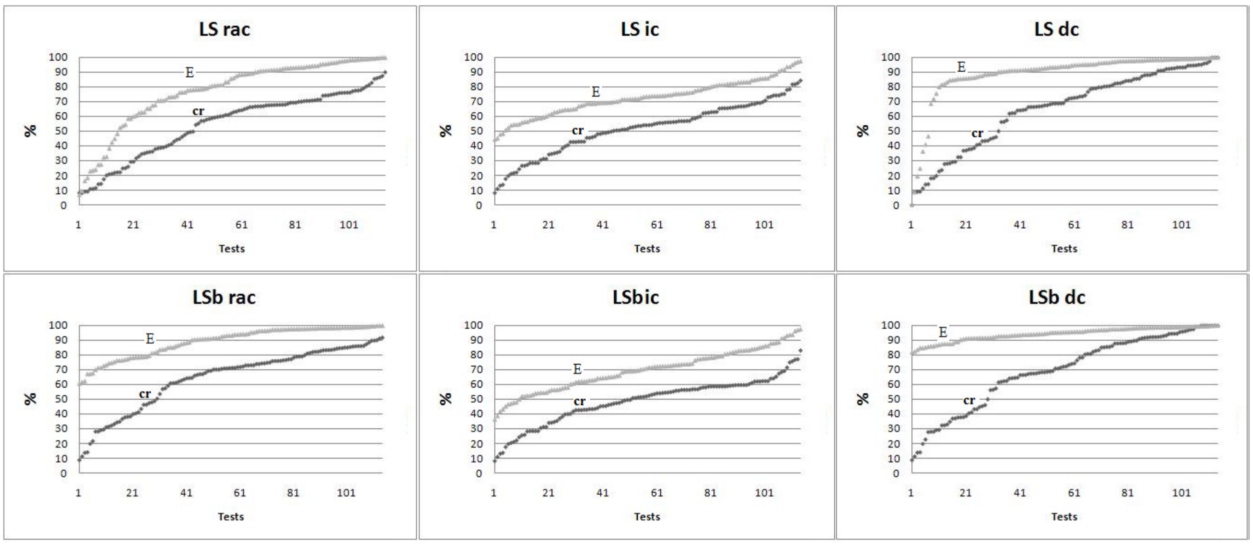

Fig 4. Efficiency (E) and cr profiles for algorithm versions in LMCR approach

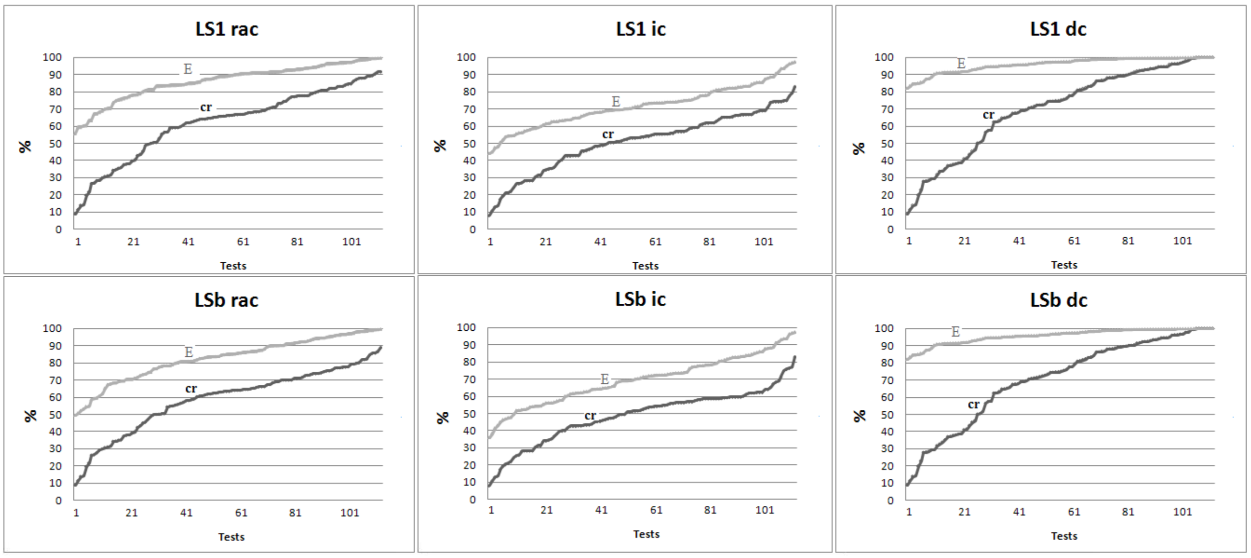

Fig 5. Efficiency (E) and cr profiles for algorithm versions in PTA approach

We can remark that in LMCR as well as in PTA, we have the following :

- For both LS-ic and LSb-ic, the efficiency and cr profiles follow enough similar features i.e. the *curves* are translatable.
- For both LS-rac and LSb-rac, the efficiency and cr profiles follow somewhat dissimilar features.
- For both LS-dc and LSb-dc, the efficiency and cr profiles follow quite dissimilar features since the difference between cr and E is large (resp. small) when their values are small (resp. large).

IV. CONCLUSION

We proposed in this paper two approaches for the determination of approximate solutions for the COP of scheduling independent tasks on heterogeneous processors under limited makespan constraint. Each approach involved several alternatives. The objective was to schedule a subset of tasks optimizing a specific criterion which is either the size (number of tasks) or maximum weight (sum of task costs). A series of experiments could validate our contribution and establish accurate comparisons between the two approaches.

This work leads us to precise some interesting perspectives we intend to study in the future. We may particularly cite the following:

- Design and experiment other alternative approaches for solving the addressed problem
- Generalize our study to the case of preemptive scheduling
- Parallelize the designed algorithms in order to process problems of larger sizes in reduced time

REFERENCES

[1] Blazewicz, J., Lenstra, J. K., & Kan, A.H.G.R. (1983), Scheduling projects to resource constraints: Classification and complexity. *Discrete Applied Mathematics,* 5(1), 11-24.

[2] Brucker, Peter & Knust, S. (2001). Resource-constrained project scheduling and timetabling, *Practice and Theory of Automated Timetabling* 3, 277-293.

[3] Cavalcante, C., Cristina, C.B., Cavalcante, Victor, F., Ribeiro, Celso, C., De Souza & Cid, C. (2002), Parallel Cooperative Approaches for the Labor Constrained Scheduling Problem, *Essays and Surveys in Metaheuristics*, 15, 201-225.

[4] Daniel, S. (1977), Hirschberg Algorithms for the Longest Common Subsequence Problem, *Journal of the ACM*, 24(4), 664-675.

[5] Eunice, L.A., (2009), Ordonnancement de projet sous contraintes de ressources à l'aide d'un algorithme génétique à croisement hybride de type OER, Tech. Rpt, Quebec University at Chicoutimi, Canada.

[6] Eyraud, L. (2006), Théorie et pratique de l'ordonnancement d'applications sur les systèmes distribués, Doctoral thesis, INP Grenoble, France.

[7] Kone O. (2009), Nouvelles approches pour la résolution du problème d'ordonnancement de projet à moyens limités, Doctoral thesis, University of Toulouse, France.

[8] Korf, R.E. (2002), A New algorithm for optimal bin packing, American Association for Artificial Intelligence Proceedings, AAAI-02, 731-736.

[9] Koulinas, G.K., Anagnostopoulos, K.P. (2013), A new tabu search-based hyper-heuristic algorithm for solving construction leveling problems with limited resource availabilities, *Automation in Construction*, 31, 169–175.

[10] Lahimer, A., Lopez, P., & Haouari, M. (2011), Ordonnancement d'atelier de type flow shop hybride avec tâches multiprocesseurs, Proc. *Congrès International de Génie Industriel (CIGI),* Saint Sauveur, Canada.

[11] Malapert, A. (2011), Techniques d'ordonnancement d'atelier et de fournées basées sur la programmation par contraintes, Doctoral thesis, ENSTIM, Nantes, France.

[12] Orji, I.M.J., & Wei, S. (2013), Project Scheduling Under Resource Constraints: A Recent Survey Ifeyinwa, *International Journal of Engineering Research & Technology (IJERT)*, 2(2).

[13] Somai, R. (2013), Optimisation des ressources dans les ordonnancements de tâches indépendantes en milieu hétérogène, Master thesis, University of Tunis El Manar, Faculty of Sciences of Tunis, Tunis, Tunisia

International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014, pp.102-115
ISSN 2356-5608