Effectiveness of heuristics and simulated annealing for the scheduling of concurrent tasks. An empirical comparison

Christophe Coroyer, Zhen Liu

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1379

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes

EFFECTIVENESS OF HEURISTICS AND SIMULATED ANNEALING FOR THE SCHEDULING OF CONCURRENT TASKS - AN EMPIRICAL COMPARISON

Christophe COROYER
Zhen LIU

Janvier 1991

RR-1379

Efficacité des Heuristiques et du Recuit Simulé pour l'Ordonnancement de Tâches Concurrentes — Une Comparaison Empirique

Christophe COROYER*

I3S-LISAN
CNRS, URA 1376
Bât. 4, rue A. Einstein
06560 Valbonne, France

Zhen LIU

INRIA
Centre Sophia Antipolis
2004 route des Lucioles
06565 Valbonne, France

le 10 Janvier 1991

Résumé: Cet article concerne l'ordonnancement de tâches concurrentes liées par des contraintes de précédence. Il est bien connu que le problème consistant à trouver l'ordonnancement qui minimise le temps total d'achèvement de cet ensemble de tâches dans un système parallèle est NP-complet. Dans cet article, nous étudions à la fois l'efficacité moyenne (au sens de la qualité des solutions) et la complexité moyenne en temps de 27 heuristiques et de 7 algorithmes de recuit simulé, utilisés pour la minimisation de ce temps total d'achèvement. Des résultats expérimentaux mettent en évidence que lorsqu'ils convergent, les algorithmes de recuit simulé sont très efficaces comparés aux heuristiques. On montre aussi que certaines heuristiques sont assez efficaces en moyenne, et que les heuristiques, quand elles sont utilisées conjointement, ont un comportement qualitatif comparable à celui du recuit simulé tout en étant beaucoup plus rapides.

Mots-Clés: Ordonnancement, heuristiques, recuit simulé, temps total d'achèvement, comparaison empirique, traitement parallèle, contraintes de précédence.

*Ces Travaux ont été effectués pendant que cet auteur était à l'INRIA, Centre Sophia Antipolis.

Effectiveness of Heuristics and Simulated Annealing for the Scheduling of Concurrent Tasks — An Empirical Comparison

Christophe COROYER*

I3S–LISAN
CNRS, URA 1376
Bât. 4, rue A. Einstein
06560 Valbonne, France

Zhen LIU

INRIA
Centre Sophia Antipolis
2004 route des Lucioles
06565 Valbonne, France

January 10, 1991

Abstract: It is well-known that the scheduling of concurrent tasks with precedence constraints in parallel systems in order to minimize the makespan is NP-complete. We study both the average effectiveness and the average efficiency of 27 heuristics and 7 simulated annealing algorithms used for the minimization of makespan. It is shown, by a computational experiment, that the simulated annealing algorithms are very effective compared with the heuristics, provided these algorithms converge. It turns out that some heuristics are quite effective on average, and that the heuristics, provided they are used together, have a qualitative behavior not much worse than that of the simulated annealing and are much more efficient.

Keywords: Scheduling, heuristics, simulated annealing, makespan, empirical comparison, parallel processing, precedence constraints.

*The work was performed when this author was with INRIA, Centre Sophia Antipolis.

1 Introduction and Problem Description

Consider the following scheduling problem: there are N parallel tasks and K parallel processors, indexed from 1 to N and 1 to K, respectively. A processor can execute at most one task at a time, and a task can only be assigned to one processor. The execution of the tasks is nonpreemptive, and is constrained by some precedence relations which can be described by a task graph $G = (V, E)$, where $V = \{1, 2, \cdots, N\}$ is the set of vertices corresponding to the tasks, and E the set of directed edges representing precedence relations between tasks. For all $i, j \in V$, relation $(i, j) \in E$ implies that task j can only start after task i has finished. The objective is to minimize the makespan, i.e. the total processing time of the tasks, by assigning the tasks to the processors and by ordering the execution. The task assignment may have constraints, viz. a task may only be assigned to a subset of processors. Two extreme cases are considered in this paper: 1) the assignment of the tasks is predefined so that each task can only be assigned to a predefined processor; 2) the task assignment has no constraints so that each task can be assigned to any of the processors.

Studies on the optimal scheduling have been carried out for particular cases. Hu [7] studied precedence relations of in-tree structure, i.e. a task has at most one successor, and there is only one task (root of the tree) without a successor. It is shown that the HLF (Highest Level First) policy is optimal. Coffman and Graham [2] found an efficient algorithm for scheduling identical tasks with arbitrary precedence constraints on two parallel processors. Coffman and Liu [3] analyzed the out-forest structure and found optimal scheduling when all the subtrees in the out-forest have an embedding relation.

Algorithms leading to optimal solutions of the general model can also be found in the literature (e.g. Ramamoorthy et al. [13]). However, it is well known that optimal solutions are NP-complete [10]. In fact, when there are no precedence relations, this boils down to the classical job-shop problem in the theory of operations research.

Various heuristics have been proposed in the literature, see e.g. Ramamoorthy et al. [13], Lo [12], Liu and Labetoulle [11]. Worst-case analysis of the heuristics (see Graham [4]) shows that the ratio of the makespan of any priority-based heuristic to the optimal solution is bounded by $2 - 1/K$.

The technique of Simulated Annealing (SA), which is an extension of the Monte Carlo method, is used to find approximate solutions to combinatorial optimization problems [8], [9]. In particular, this can also be applied to our scheduling problem.

In this paper, we are interested in the average effectiveness (i.e. the average quality of the solutions) and the average efficiency (i.e. the average time complexity) of the heuristics and the SA algorithms, as well as the comparison of their effectiveness and efficiency. We carry out a computational experiment on 27 heuristics and 7 homogeneous and inhomogeneous SA approaches. It is shown, by a computational experiment, that the SA algorithms are very effective compared with the heuristics, provided the temperature is decreasing slowly enough so that these algorithms converge. It turns out, unlike the bound provided by the worst case analysis, that some heuristics are quite effective on average. Another observation is that the heuristics, when they are used together, i.e. for every problem all the heuristics are tried and the best solution is chosen, have a qualitative behavior not much worse than that of the SA and are much more efficient.

Earlier empirical comparisons of heuristics were performed by Adam et al. [1] where 4 heuristics were tested. Our investigation differs from that of Talbot et al. [15] where the heuristics for minimizing the number of processors were considered.

The paper is organized as follows. In Section 2, we describe the heuristics that are considered in the paper. In Section 3, we discuss how the technique of SA are used for solving the scheduling problem. In Sections 4 and 5, we present empirical comparison results among heuristics and between heuristics and SA algorithms, respectively.

2 Priority-Based Heuristics

We consider a class of priority-based heuristic algorithms for the scheduling problem. Since the tasks that we deal with have precedence constraints, it is natural to consider the tasks to be scheduled according to the partial order defined on them. Hence in our scheduling algorithm, only those tasks which have no predecessors in the task graph, or whose predecessors have already been scheduled by the algorithm, will be taken into consideration as candidates for scheduling.

The basic idea of priority-based heuristic approach consists of choosing tasks one by one from the candidates according to their priorities. As soon as a processor is idle, we assign the candidate task with the highest priority onto it, provided the task has access to the processor. If there is no candidate, the processor remains idle. More precisely, let C denote the set of candidate tasks, P_i the processing time of task i, and T_j the available time of processor j, i.e. the time when processor j finishes the tasks already assigned on it, and can begin to process another task. This schema can be written in a PASCAL-like language as follows.

Generic Priority-Based Heuristic

C = set of tasks without predecessors;
for $k = 1, \cdots, K$ **do** $T_k = 0$;
repeat
  $H = \min_{1 \le k \le K} T_k$;
  repeat
    $S = \{j \mid T_j = H\}$;
    for all $j \in S$:
      if the last finished task of processor j has successors which have
        no unfinished predecessors at time H
      then put these successors into C;
      if C is empty **then**
        $H = \min_{1 \le k \le K} \max(H, T_k)$;
        for all $j \in S$ **do** $T_j = H$;
  until C is not empty;
  for all $j \in S$:
    assign the task i onto processor j such that i has the highest priority
      among the tasks in C which have access to processor j;
    remove i from C;
    update T_j by $T_j + P_i$;
until all the tasks are scheduled.

Different heuristics can be obtained by using this schema together with some priority criteria. In general, one can distinguish two types of priorities: static and dynamic rules. Static rules consist of associating with each task a priority which can be determined before the scheduling process, while dynamic rules determine the priorities of tasks during the scheduling process. When a tie occurs, more priority criteria can be used, so that one can construct hybrid priorities.

In the remainder of this section, we present some examples of priority rules. Each of the priority rules generates a heuristic. Denote by $p(i)$ and $s(i)$ the sets of predecessors and successors of task i in G, respectively.

Static priority rules.

- **Largest Processing Time** (LPT)

  One of the simplest criteria that can be used as a priority rule is the task processing time P_i. The LPT policy requires that the task whose processing time is the largest among the candidates be scheduled first.

- **Smallest Processing Time** (SPT)

  On the contrary to LPT policy, the Smallest Processing Time policy requires that the task whose processing time is the smallest among the candidates be scheduled first.

- **Highest Level** (HL)

  In the above two heuristic algorithms, the structure of the task graph is not taken into consideration. In the HL policy, we assign a priority to a task with its **level** in the task graph. The level of task i, denoted by $l(i)$, is defined recursively as follows:

$$l(i) = 0, \qquad \forall i: \quad s(i) = \emptyset;$$

$$l(i) = \max_{j \in s(i)} l(j) + 1$$

  Highest Level policy consists of giving high priorities to tasks of high levels.

- **Most Sons** (MS)

  Another way to take into account the task graph structure is to define the priority of task i as the cardinality of the set of successors of i, $|s(i)|$. This yields the so called Most Sons (MS) policy which schedules the candidate task with most sons first.

- **Most Descendants** (MD)

  Define further $d(i)$ be the set of descendants of task i in G.

$$d(i) = \emptyset, \qquad \forall i: \quad s(i) = \emptyset,$$

$$d(i) = \bigcup_{j \in s(i)} (\{j\} \cup d(j))$$

  Similar to the Most Sons policy, the Most Descendants policy takes the number of descendants of task i, $|d(i)|$, as the priority criterion.

- **Longest Path** (LP)

  In the above three algorithms, the task processing time is not used. We now design some policies which take into account both task processing times and task graph structure.

  The first approach is to modify Highest Level policy. We associate a priority to a task by the **length** of its longest path, which will simply be referred to as the length, in the task graph that begins with the task. The length of task i, denoted by $L(i)$, is defined recursively as follows:

$$L(i) = P_i, \qquad \forall i: \quad s(i) = \emptyset,$$

$$L(i) = \max_{j \in s(i)} L(j) + P_i.$$

The LP policy then assigns the task with longest path among the candidates to the first available processor.

- **Successors' Weight** (SW)

  Another possible approach is to define a weight $W(i)$ for task i. This can be defined on the set of successors:

  $$W(i) = P_i + \sum_{j \in s(i)} P_j$$

  The SW policy takes this weight as priority: the heavier the task is, the higher priority the task has.

- **Descendants' Weight** (DW)

  The weight of a task can also be defined on the set of descendants:

  $$W(i) = P_i + \sum_{j \in d(i)} P_j$$

  The DW policy associates high priorities to heavy tasks.

Dynamic priority rules.

- **Earliest Enabled** (EE)

  In the Earliest Enabled policy, tasks which are first enabled, i.e. the ones whose predecessors are complete earliest, are given the highest priorities. In order to define the enabled time of task i, denoted by $E(i)$, we need the notion of completion time of the task, referred to as $C(i)$, which is the time when the processing of the task is finished by one of the processors according to some scheduling policy.

  $$E(i) = 0, \qquad \forall i: \quad p(i) = \emptyset,$$

  $$E(i) = \max_{j \in p(i)} C(j).$$

  Hence, in Earliest Enabled policy, priorities are not determined all at once. They are computed gradually in the scheduling algorithm.

- **Least Schedule Flexibility** (LSF)

The criterion Least Schedule Flexibility reflects the criticalness of the task in the task graph. The LSF policy studied in this paper is a simplified version of an algorithm proposed by Liu and Labetoulle [11].

Before describing the priority rule, we define some notation:

- $r(i)$: the location of scheduled task i, i.e. the index of the processor on which task i is assigned.
- FT_i: the completion time of task i provided it is already scheduled.
- CT_i: the estimated minimum completion time of task i.
- ET_i: the estimated earliest time when task i could start execution.
- LT_i: the estimated latest time when task i should start execution so as not to increase the makespan.
- $SF_i = LT_i - ET_i$: the **Schedule Flexibility** of task i.

The LSF heuristic consists of associating tasks with the least schedule flexibilities the highest priorities. The schedule flexibilities are recomputed, whenever some tasks are scheduled, by the following algorithm.

Algorithm for Estimating the Schedule Flexibility

$\mathcal{T} = \{i | p(i) = \emptyset\}$;
$np[i] = |p(i)|$;
$ns[i] = |s(i)|$;
repeat
  take i from $\mathcal{T}$;
  if task i is already scheduled **then**
    $ET_i = FT_i - P_i$;
    $CT_i = FT_i$;
  else if $p(i) = \emptyset$ **then**
      $ET_i = 0$;
      $CT_i = P_i$;
    else
        $ET_i = \max(\max_{v \in p(i)} CT_v, \min_{1 \le j \le K} T_j)$;
        $CT_i = ET_i + P_i$;
  for every $v \in s(i)$ **do**
    $np[v] = np[v] - 1$;
    if $np[v] = 0$ **then** add v into $\mathcal{T}$;
until $\mathcal{T} = \emptyset$;
$\mathcal{T} = \{i | s(i) = \emptyset\}$;

repeat
take i from $\mathcal{T}$;
if $s(i) = \emptyset$
then $LT_i = ET_i$
else $LT_i = \min_{v \in s(i)} LT_v - P_i$;
for every $v \in p(i)$ **do**
$ns[v] = ns[v] - 1$;
if $ns[v] = 0$ **then** add v into $\mathcal{T}$;
until $\mathcal{T} = \emptyset$.

Hybrid priority rules.

When using priority based algorithms, one frequently encounters the situation where several candidates have the same priority, i.e. a tie occurs. If such is the case, one can either arbitrarily choose one of the candidates or introduce more priority criteria. The latter yields hybrid heuristics. The following list contains some examples of such heuristics, the first of which was proposed by Ramamoorthy et al. [13], and others are simply some combinations of the static and dynamic priority rules presented above.

- **Highest Level Essential Task** (HLET)

  The Highest Level Essential Task policy is basically the same as the Highest Level policy. However, when a tie occurs, we select an *essential* task to schedule. Task i is said to be essential, at level l, if :

  $$i \in (E_l \bigcap L_l)$$

  where E_l and L_l are defined as follows: in the task graph, tasks having no predecessors belong to E_1. We remove those tasks from the graph and we obtain E_2 by taking vertices which have no more predecessors and so on. The set L_l consists of vertices of the same level in the task graph, L_1 being the set of tasks of highest level.

- **Highest Level Largest Processing Time** (HLLPT)

  With HLLPT we always select the task with the largest processing time amongst those which have the highest level.

- **Highest Level Smallest Processing Time** (HLSPT)

  The policy HLSPT can be interpreted in a similar way.

- **Most Sons Longest Path** (MSLP)

  In MSLP the cardinality of the set of successors is taken as the first criterion, and the path length as the second one.

- **Most Descendants Longest Path** (MDLP)

  The policy MDLP differs from the previous one in that the cardinality of the set of descendants is the first priority criterion.

- **Most Descendants Least Schedule Flexibility** (MDLSF)

  The policy MDLSF differs from the previous one in that the schedule flexibility is used for breaking ties.

- **Longest Path Largest Processing Time** (LPLPT)

  The policy LPLPT chooses the candidates with the longest path length, and the tie is broken by selecting the task with the largest processing time.

- **Longest Path Smallest Processing Time** (LPSPT)

  The policy LPSPT is similar to LPLPT except that the tie is broken by taking the task with the smallest processing time.

- **Longest Path Most Sons** (LPMS)

  The policy LPMS takes the path length as the first criterion and the cardinality of the set of successors as the solution to ties.

- **Longest Path Most Descendants** (LPMD)

  The policy LPMD replaces the cardinality of the set of successors in the previous policy by that of descendants for breaking ties.

- **Longest Path Least Schedule Flexibility** (LPLSF)

  The policy LPLSF is the same as LPMD except that the schedule flexibility is used for breaking ties.

- **Least Schedule Flexibility Highest Level** (LSFHL)

  The policy LSFHL is a combination of the policies of Least Schedule Flexibility and Highest Level, where the latter is used to break ties.

- **Least Schedule Flexibility Descendants' Weight** (LSFDW)

  The policy LSFDW is similar to the previous one except that the ties are broken by the descendants' weights.

- **Least Schedule Flexibility Longest Path** (LSFLP)

  The policy LSFLP selects candidates according to the rule of least schedule flexibility. If a tie occurs, the path length is used for its resolution.

- **Least Schedule Flexibility Most Descendants** (LSFMD)

  The policy LSFMD can be interpreted analogously except that the cardinality of the set of descendants is used to resolve ties.

- **Most Sons Longest Path Successors' Weight** (MSLPSW)

  The policy MSLPSW is an example of three-level priorities. As the name of the policy suggests, we first use the cardinality of successors to make a selection among the candidates, and use path length to break ties. If a tie still exists after such a two-level selection, we choose the task with the heaviest successors' weight.

- **Most Sons Longest Path Descendants' Weight** (MSLPDW)

  Th policy MSLPDW behaves like the previous one except that the descendants' weight is used as the last resort in tie breaking.

3 Simulated Annealing

The technique of Simulated Annealing (SA) is often used to find approximate solutions to combinatorial optimization problems [9]. This method is used here for solving the task scheduling problem.

The basic idea of the SA algorithms is as follows: one starts with an initial solution and an initial temperature (which is a control parameter) and iterates the improvement procedure. In each iteration, one decreases, when necessary, the temperature and generates a new solution by perturbing the configuration of the old one. If the cost of the new solution is lower than the old one, the new solution is accepted. Otherwise, this solution is accepted with a certain probability.

We distinguish two types of the SA algorithms according to the temperature decrease procedure:

- **homogeneous algorithm:** the temperature is decreased when some equilibrium state of the solutions is reached.
- **inhomogeneous algorithm:** the temperature is decreased at each iteration.

These two schemas are illustrated below, where T, S and C denote the temperature, the solution and the cost of the solution, respectively.

Homogeneous algorithm

T=initial temperature;
S=initial solution;
C=cost of the initial solution;
repeat
  repeat
    S'=PERTURB(S);
    C'=cost of S';
    if $C' < C$ **then** accept S'
    else accept S' with probability $\exp((C - C')/T)$;
    if accept **then** update S and C;
  until equilibrium is reached;
  update temperature T;
until stopping criterion is fulfilled.

Inhmogeneous algorithm

T=initial temperature;
S=initial solution;
C=cost of the initial solution;
repeat
  S'=PERTURB(S);
  C'=cost of S';
  if $C' < C$ **then** accept S'
  else accept S' with probability $\exp((C - C')/T)$;
  if accept **then** update S and C;
  update temperature T;
until stopping criterion is fulfilled.

In terms of a Markov chain, a solution is a state and an iteration represents a state transition. The homogeneous and the inhomogeneous algorithms correspond to the homogeneous and the inhomogeneous Markov chains, respectively.

We refer the reader to [9] for a detailed discussion on the convergence and implementa-

tion issues of the algorithms. We discuss below how these algorithms are used for solving our scheduling problem.

The **initial solution** is arbitrarily chosen.

The **initial temperature** should be determined in such a way that all the transitions at this temperature are accepted. However, in practice, we define an acceptance ratio χ_0, which equals the number of accepted solutions over the number of proposed solutions, and we determine the initial temperature in such a way that the acceptance rate at the initial temperature is not less than χ_0. One of the possible ways to do that is to generate several new solutions and compute the average increase of cost ΔC, and set $-\Delta C/\ln \chi_0$ as the initial temperature.

The **perturbation** of a solution is implemented by either moving a task from one processor to another or permute the execution order of two tasks on the same processor.

The **equilibrium** is reached when the probability distribution of the solution equals the Boltzman distribution given by : $P(S) = \exp(-C(S)/T)/Q(T)$, where S, T and $Q(T)$ are the solution, the temperature and the normalizing constant, respectively. In the implementation, we approach this equilibrium state by, according to the recommendations of Ramanujam et al. [14], setting the number of internal iterations (cf. homogeneous algorithm) as a multiple of the number of neighbors of a solution. In accordance with the above perturbation function, we define two measures of the number of neighbors in a problem:

$$\alpha = n(m-1) \qquad \text{and} \qquad \beta = \sum_{i=1}^{n} n_i/2,$$

where n is the number of tasks, m is the number of processors, and n_i is the number of tasks which have no precedence relation with task i in the task graph.

The **temperature update** function is geometric for homogeneous algorithms. As for inhomogeneous algorithms, it is shown (see [5,6,9]) that the convergence to a globally minimal solution is ensured when the temperature tends to zero not faster than $O([\log k]^{-1})$. In our experiments, it turns out that such a decreasing speed is too slow for most of the problems. Thus we have chosen decreasing functions of orders $O(q^k)$ $(q < 1)$ and $O(1/k)$.

Two **stopping criteria** are used in our implementations: The annealing process is stopped either when the temperature is lower than certain threshold or when consecutive solutions are identical for a certain number of times.

4 Comparison between Heuristics

The comparison between heuristics has been carried out for about 700 problems. We have randomly generated 100 task graphs by fixing the precedence ratio to be 1/8, and by varying the number of tasks from 20 to 100, and the execution times of the tasks from 1 to 100. For each task graph and each heuristic, experiments are performed for the number of processors being equal to $2, 3, \cdots$, until the makespan reaches the *theoretical minimum makespan*, i.e. the length of the critical path of the task graph.

4.1 Scheduling with a Given Assignment

Consider first the case where the assignment of the tasks is predefined. The heuristics are used to obtain the execution order on each of the processors. We have generated $J = 700$ problems and for each of them, the assignment is randomly defined.

The comparison results are presented in Table 1. The results are obtained by computing the solutions of all the heuristics for each of the problems. Denote by π an arbitrary scheduling policy. For each problem i, $i = 1, \cdots, J$, denote by $M_i(\pi)$ the makespan of problem i given by policy π, and denote by M_i the makespan of the best solution of the problem found by the policies under investigation, viz. $M_i = \min_\pi M_i(\pi)$. The data provided in the table indicate:

- column **A**: the rate at which a policy yields the best solution: $100 \times \sum_{i=1}^{J} 1(M_i(\pi) = M_i)/J$.
- column **B**: the average relative difference between the solution of a policy and the best solution: $\sum_{i=1}^{J}(M_i(\pi) - M_i)/M_i$.
- column **C**: the average of relative qualities of the solutions of a policy with respect to the best solutions: $\sum_{i=1}^{J}(M_i(\pi)/M_i)/J$.
- column **D**: the relative quality of the average of the solutions of a policy with respect to the best solutions: $\sum_{i=1}^{J} M_i(\pi)/\sum_{i=1}^{J} M_i$.

One might have observed that the **Least Schedule Flexibility** policy as well as the hybrid policies based on it are the best among this set of heuristics. Note also that the policies based on the notion **Most Descendants** are also very effective. According to these results, the **Largest Processing Time** policy is the poorest within this set of heuristics.

Table 1: Comparison of the heuristics with a given assignment

	A	B	C	D
Largest Processing Time	3.29	19.40	1.19	1.19
Smallest Processing Time	9.43	11.47	1.11	1.11
Highest Level	52.14	2.10	1.02	1.02
Most Sons	21.86	8.33	1.08	1.08
Most Descendants	50.86	2.06	1.02	1.02
Longest Path	40.29	2.75	1.03	1.03
Successors' Weight	10.43	12.53	1.13	1.13
Descendants' Weight	48.86	2.07	1.02	1.02
Earliest Enabled	12.71	9.59	1.10	1.09
Least Schedule Flexibility	60.00	1.45	1.01	1.01
Highest Level Essential Task	50.57	2.20	1.02	1.02
Highest Level Largest Processing Time	43.00	2.99	1.03	1.03
Highest Level Smallest Processing Time	52.71	1.98	1.02	1.02
Most Sons Longest Path	24.29	7.67	1.08	1.08
Most Descendants Longest Path	53.14	1.93	1.02	1.02
Most Descendants Least Schedule Flexibility	55.57	1.81	1.02	1.02
Longest Path Largest Processing Time	40.00	2.82	1.03	1.03
Longest Path Smallest Processing Time	40.00	2.76	1.03	1.03
Longest Path Most Sons	40.14	2.77	1.03	1.03
Longest Path Most Descendants	40.14	2.76	1.03	1.03
Longest Path Least Schedule Flexibility	40.00	2.76	1.03	1.03
Least Schedule Flexibility Highest Level	60.00	1.44	1.01	1.01
Least Schedule Flexibility Descendants' Weight	59.14	1.45	1.01	1.01
Least Schedule Flexibility Longest Path	56.86	1.53	1.02	1.01
Least Schedule Flexibility Most Descendants	59.57	1.44	1.01	1.01
Most Sons Longest Path Successors' Weight	24.43	7.66	1.08	1.08
Most Sons Longest Path Descendants' Weight	24.29	7.67	1.08	1.08

4.2 Scheduling without Assignment Constraints

Consider now the case where the task assignment has no constraints. The heuristics have to assign the tasks and schedule their executions. The comparison has been done on $J = 699$ problems and the results are provided in Table 2.

Note that in this case, the best heuristics are those based on the notion of **Longest Path**. The poorest one now becomes the **Smallest Processing Time**.

5 Comparison between Heuristics and Simulated Annealing

In this section, we compare the heuristics with the SA algorithms. As in the previous section, the problems are randomly generated. The parameters used for generating the problems are the same.

5.1 Scheduling with a Given Assignment

We first focus on the case where the task assignment is predefined. We use the same 700 problems as in Section 4.1. We have implemented a homogeneous SA algorithm. The initial solution is randomly generated. The initial temperature is computed in order to have an initial acceptance ratio not less than 0.95. The number of internal iterations equals β (cf. Section 3). The temperature updating function is geometric with ratio 0.95. The algorithm is stopped when the temperature is lower than 10^{-6}. The comparison results are summerized in Table 3, where the columns have the same meaning as in Section 4.1.

It turns out that in 55.43% of the cases, the SA algorithm strictly overperforms the heuristics, whereas in 5.71% of the cases, the heuristics strictly overperform the SA algorithm, and in 38.86% of the cases the best solution found by the heuristics has the same makespan as the SA algorithm. The average difference of makespan between heuristics and the simulated annealing is 17.20.

We thus conclude that when there is a given task assignment, the SA algorithm is effective. Moreover, it finishes within a reasonable time. In Table 4, we provide the average execution times of the heuristics and the SA algorithm applied to these 700 problems on a DEC-5400.

Table 2: Comparison of the heuristics without assignment constraints

	A	B	C	D
Largest Processing Time	27.18	5.68	1.06	1.05
Smallest Processing Time	15.88	7.16	1.07	1.07
Highest Level	44.49	1.42	1.01	1.01
Most Sons	25.89	4.88	1.05	1.05
Most Descendants	43.35	1.49	1.01	1.01
Longest Path	67.67	0.24	1.00	1.00
Successors' Weight	27.90	4.78	1.05	1.04
Descendants' Weight	56.08	0.56	1.01	1.00
Earliest Enabled	25.46	5.06	1.05	1.05
Least Schedule Flexibility	42.35	3.76	1.04	1.03
Highest Level Essential Task	45.35	1.42	1.01	1.01
Highest Level Largest Processing Time	57.51	0.57	1.01	1.00
Highest Level Smallest Processing Time	41.92	1.89	1.02	1.02
Most Sons Longest Path	33.76	3.20	1.03	1.03
Most Descendants Longest Path	54.08	0.67	1.01	1.01
Most Descendants Least Schedule Flexibility	42.35	1.53	1.02	1.01
Longest Path Largest Processing Time	67.67	0.25	1.00	1.00
Longest Path Smallest Processing Time	68.10	0.24	1.00	1.00
Longest Path Most Sons	67.53	0.24	1.00	1.00
Longest Path Most Descendants	67.38	0.24	1.00	1.00
Longest Path Least Schedule Flexibility	67.38	0.24	1.00	1.00
Least Schedule Flexibility Highest Level	44.21	3.16	1.03	1.03
Least Schedule Flexibility Descendants' Weight	46.21	2.99	1.03	1.02
Least Schedule Flexibility Longest Path	49.36	2.82	1.03	1.02
Least Schedule Flexibility Most Descendants	44.21	3.28	1.03	1.03
Most Sons Longest Path Successors' Weight	33.48	3.20	1.03	1.03
Most Sons Longest Path Descendants' Weight	33.91	3.20	1.03	1.03

Table 3: Comparison between the heuristics and the SA algorithm with a given assignment

	A	B	C	D
Largest Processing Time	2.00	22.73	1.23	1.22
Smallest Processing Time	6.14	14.60	1.15	1.14
Highest Level	32.00	5.00	1.05	1.04
Most Sons	16.00	11.40	1.11	1.11
Most Descendants	31.57	4.96	1.05	1.04
Longest Path	23.57	5.66	1.06	1.05
Successors' Weight	6.71	15.71	1.16	1.15
Descendants' Weight	28.57	4.97	1.05	1.04
Earliest Enabled	8.86	12.68	1.13	1.12
Least Schedule Flexibility	34.86	4.34	1.04	1.04
Highest Level Essential Task	31.00	5.10	1.05	1.05
Highest Level Largest Processing Time	26.57	5.92	1.06	1.05
Highest Level Smallest Processing Time	31.43	4.87	1.05	1.04
Most Sons Longest Path	16.43	10.73	1.11	1.11
Most Descendants Longest Path	32.00	4.83	1.05	1.04
Most Descendants Least Schedule Flexibility	33.86	4.70	1.05	1.04
Longest Path Largest Processing Time	23.57	5.74	1.06	1.05
Longest Path Smallest Processing Time	23.43	5.67	1.06	1.05
Longest Path Most Sons	23.57	5.69	1.06	1.05
Longest Path Most Descendants	23.57	5.68	1.06	1.05
Longest Path Least Schedule Flexibility	23.43	5.67	1.06	1.05
Least Schedule Flexibility Highest Level	34.86	4.32	1.04	1.04
Least Schedule Flexibility Descendants' Weight	34.43	4.34	1.04	1.04
Least Schedule Flexibility Longest Path	33.00	4.41	1.04	1.04
Least Schedule Flexibility Most Descendants	34.71	4.33	1.04	1.04
Most Sons Longest Path Successors' Weight	16.57	10.72	1.11	1.11
Most Sons Longest Path Descendants' Weight	16.43	10.73	1.11	1.11
Simulated Annealing	94.29	0.16	1.00	1.00

Table 4: Average execution times when the assignment is given

Largest Processing Time	0.02s
Smallest Processing Time	0.02s
Highest Level	0.02s
Most Sons	0.02s
Most Descendants	0.02s
Longest Path	0.02s
Successors' Weight	0.02s
Descendants' Weight	0.02s
Earliest Enabled	0.02s
Least Schedule Flexibility	0.13s
Highest Level Essential Task	0.02s
Highest Level Largest Processing Time	0.02s
Highest Level Smallest Processing Time	0.02s
Most Sons Longest Path	0.02s
Most Descendants Longest Path	0.02s
Most Descendants Least Schedule Flexibility	0.13s
Longest Path Largest Processing Time	0.02s
Longest Path Smallest Processing Time	0.02s
Longest Path Most Sons	0.02s
Longest Path Most Descendants	0.02s
Longest Path Least Schedule Flexibility	0.13s
Least Schedule Flexibility Highest Level	0.13s
Least Schedule Flexibility Descendants' Weight	0.13s
Least Schedule Flexibility Longest Path	0.13s
Least Schedule Flexibility Most Descendants	0.13s
Most Sons Longest Path Successors' Weight	0.02s
Most Sons Longest Path Descendants' Weight	0.02s
Simulated Annealing	1m8.19s

5.2 Scheduling without Assignment Constraints

When the task assignment has no constraints, the SA algorithms are much more time-consuming. Indeed, the scheduling process now consists of two steps: assigning the tasks and ordering the executions. For each step, either SA algorithms or heuristics can be applied. We thus implement two "mixed" SA algorithms with the annealing process for the task assignment step and the heuristics for the execution scheduling, three two-step SA algorithms with annealing processes for both steps. In addition, we also implement a single step SA algorithm, which, at each perturbation, chooses with a certain probability a perturbation of task assignment or a perturbation of execution order. In all these annealing processes, the initial solution is randomly generated and the initial temperature is determined by the acceptance ratio 0.95.

1. HSA (.95, Random): We first implement an algorithm with a homogeneous simulated annealing for the task assignment step and a random policy for the execution ordering step.
   - Homogeneous annealing for the task assignment with the number of internal iterations equals α (cf. Section 3). The temperature updating function is geometric with ratio 0.95. The annealing process is stopped when the temperature is lower than 0.01.
   - The scheduling of task executions is done randomly subject to the precedence constraints.
2. HSA (.95, LSF):
   - Homogeneous annealing for the task assignment with the number of internal iterations being α and a geometrical temperature decrease of ratio 0.95. The annealing process is stopped when the temperature is lower than 0.01.
   - The Least Schedule Flexibility policy for the scheduling of task executions.
3. HSA (.5, .5): Homogeneous annealings for both the task assignment and the task execution scheduling. For both annealing processes, the number of internal iterations equals α, the temperature updating function is geometric with ratio 0.5, and the processes are stopped when the temperature is lower than 0.01 or when the last 5α consecutive solutions are identical.
4. IHSA (.98, .98):
   - Inhomogeneous annealing for the task assignment with a geometrical temperature decrease of ratio 0.98. The annealing process is stopped when the temperature is lower than 0.01.

- Inhomogeneous annealing for the scheduling of task executions with a geometrical temperature decrease of ratio 0.98. The annealing process is stopped when the temperature is lower than 0.001.

5. IHSA (1/n, 1/n):

   - Inhomogeneous annealing for the task assignment with the temperature updating function given by T_0/n, where T_0 is the initial temperature and n is the index of the current iteration. The annealing process is stopped when the temperature is lower than 0.01 or when the last 5α consecutive solutions are identical.
   - Inhomogeneous annealing for the scheduling of task executions with the temperature updating function given by T_0/n. The annealing process is stopped when the temperature is lower than 0.001 or when the last 5β consecutive solutions are identical.

6. HSA (.98): This is a one-step Homogeneous Simulated Annealing algorithm. At each perturbation, with probability p, we modify the task assignment by moving a task from one processor to another, and with probability $1 - p$, we modify the scheduling of the tasks by interchanging the execution order of two tasks on a processor provided there is no precedence constraint between these two tasks. The probability p is determined in such a way that $p/(1-p) = \alpha/\beta$. The temperature updating function is geometric with ratio 0.98. The number of internal iterations equals $p\alpha + (1-p)\beta$. The annealing process is stopped when the temperature is lower than 0.01.

Note that in the two-step annealing processes, we have chosen the above parameters in order to make possible the comparison between heuristics and the SA algorithms. However, with these parameters, the annealing processes will not converge. Thus, in the implementations of IHSA(1/n, 1/n) and HSA(.5, .5), we take the best solution encountered during the execution of the algorithms as the final solution.

In Table 5, we provide the comparison results over a set of $J = 115$ problems, where the number of tasks vary from 20 to 30. The columns in the table have the same interpretation as in those of Section 4.

It can be seen from the table that on average the simulated annealing algorithms are effective. However, they are much more time-consuming. In Table 6, we provide the average execution times of the heuristics and the SA algorithms applied to these 115 problems on a DEC-5400. The symbol ϵ indicates that the execution time is smaller than 0.01s. Observe that the HSA(.98) is the most effective and also the most time-consuming.

Table 5: Comparisons of heuristics and SA algorithms without assignment constraints

	A	B	C	D
Largest Processing Time	20.00	8.89	1.09	1.09
Smallest Processing Time	11.30	11.41	1.11	1.11
Highest Level	53.04	2.28	1.02	1.02
Most Sons	39.13	3.82	1.04	1.04
Most Descendants	50.43	2.25	1.02	1.02
Longest Path	69.57	0.58	1.01	1.01
Successors' Weight	43.48	4.46	1.04	1.04
Descendants' Weight	62.61	0.75	1.01	1.01
Earliest Enabled	19.13	8.89	1.09	1.09
Least Schedule Flexibility	33.04	6.24	1.06	1.06
Highest Level Essential Task	53.91	1.93	1.02	1.02
Highest Level Largest Processing Time	66.96	0.64	1.01	1.01
Highest Level Smallest Processing Time	49.57	3.06	1.03	1.03
Most Sons Longest Path	54.78	1.43	1.01	1.01
Most Descendants Longest Path	58.26	0.94	1.01	1.01
Most Descendants Least Schedule Flexibility	51.30	2.33	1.02	1.02
Longest Path Largest Processing Time	70.43	0.56	1.01	1.01
Longest Path Smallest Processing Time	68.70	0.57	1.01	1.01
Longest Path Most Sons	68.70	0.54	1.01	1.01
Longest Path Most Descendants	68.70	0.56	1.01	1.01
Longest Path Least Schedule Flexibility	71.30	0.56	1.01	1.01
Least Schedule Flexibility Highest Level	44.35	6.27	1.06	1.06
Least Schedule Flexibility Descendants' Weight	45.22	6.35	1.06	1.06
Least Schedule Flexibility Longest Path	45.22	6.32	1.06	1.06
Least Schedule Flexibility Most Descendants	45.22	6.35	1.06	1.06
Most Sons Longest Path Successors' Weight	55.65	1.41	1.01	1.01
Most Sons Longest Path Descendants' Weight	54.78	1.43	1.01	1.01
Homogeneous SA (.95, Random)	66.96	0.98	1.01	1.01
Homogeneous SA (.95, LSF)	80.87	0.23	1.00	1.00
Homogeneous SA (.5 .5)	67.83	0.51	1.01	1.00
Inhomogeneous SA (.98 .98)	69.57	0.85	1.01	1.01
Inhomogeneous SA (1/n, 1/n)	78.26	0.39	1.00	1.00
Homogeneous SA (.98)	93.91	0.08	1.00	1.00

Table 6: Average execution times without assignment constraints

Largest Processing Time	ϵ
Smallest Processing Time	ϵ
Highest Level	ϵ
Most Sons	ϵ
Most Descendants	ϵ
Longest Path	ϵ
Successors' Weight	ϵ
Descendants' Weight	ϵ
Earliest Enabled	ϵ
Least Schedule Flexibility	0.02s
Highest Level Essential Task	ϵ
Highest Level Largest Processing Time	ϵ
Highest Level Smallest Processing Time	ϵ
Most Sons Longest Path	ϵ
Most Descendants Longest Path	ϵ
Most Descendants Least Schedule Flexibility	0.02s
Longest Path Largest Processing Time	ϵ
Longest Path Smallest Processing Time	ϵ
Longest Path Most Sons	ϵ
Longest Path Most Descendants	ϵ
Longest Path Least Schedule Flexibility	0.02s
Least Schedule Flexibility Highest Level	0.02s
Least Schedule Flexibility Descendants' Weight	0.02s
Least Schedule Flexibility Longest Path	0.02s
Least Schedule Flexibility Most Descendants	0.02s
Most Sons Longest Path Successors' Weight	ϵ
Most Sons Longest Path Descendants' Weight	ϵ
Homogeneous SA (.95, Random)	1m12.04s
Homogeneous SA (.95, LSF)	5m50.27s
Homogeneous SA (.5 .5)	10m37.81s
Inhomogeneous SA (.98 .98)	3m45.37s
Inhomogeneous SA (1/n, 1/n)	10m19.59s
Homogeneous SA (.98)	25m19.02s

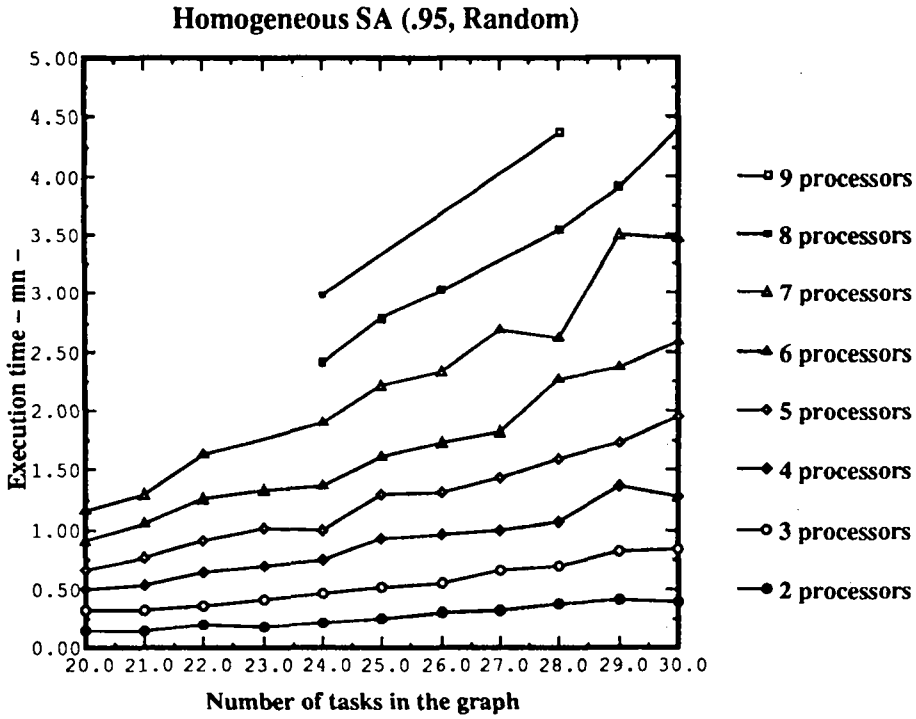

Figure 1: The average execution times of HSA(.95, Random).

In Figures 1 to 6, we illustrates the average execution times of the above 6 SA algorithms as a function of the number of tasks and processors. Due to the fact that there are only 115 problems with randomly generated task graphs, a point in the figures represents the average execution time of one, two or three problems. Therefore, the average execution times provided by the figures are not precise. However, one can observe that in general the execution times of the implemented SA algorithms are monotonically increasing in the number of tasks, and they are monotonically increasing, except for the HSA(.98) algorithm, in the number of processors. Concerning the HSA(.98) algorithm, Figure 6 shows that it is monotonically decreasing in the number of processors! One of the reasons for this is that the number of internal iterations equals $(\alpha^2+\beta^2)/(\alpha+\beta)$ which is not monotonic in α. Another is that the movement of a task from one processor to another may generate a deadlock on the destination processor due to the precedence constraints. According to our experience, such phenomena occur more often when the number of processors is small.

It is worthwhile noticing that if these heuristics are used together for each of the problems, i.e. for each problem we try all the heuristics and take the best solution, they have quite a similar effectiveness as "best" SA algorithms. Such a fact is illustrated in Table 7, where the experimental results are obtained for the above 115 problems. One observes that the total execution time of the heuristics is over 100 times smaller than those SA algorithms which have

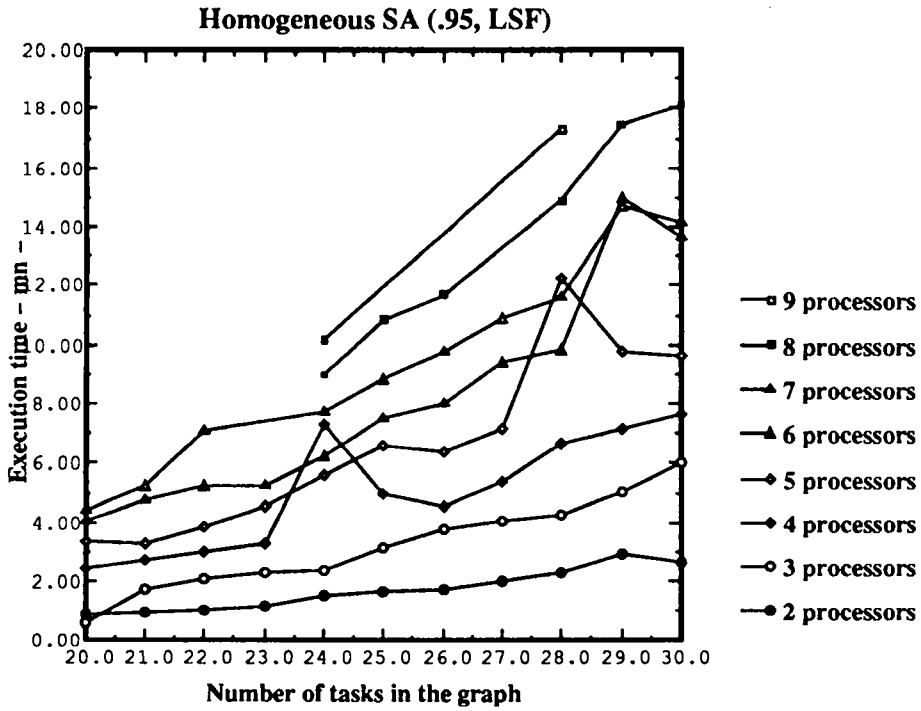

Figure 2: The average execution times of HSA(.95, LSF).

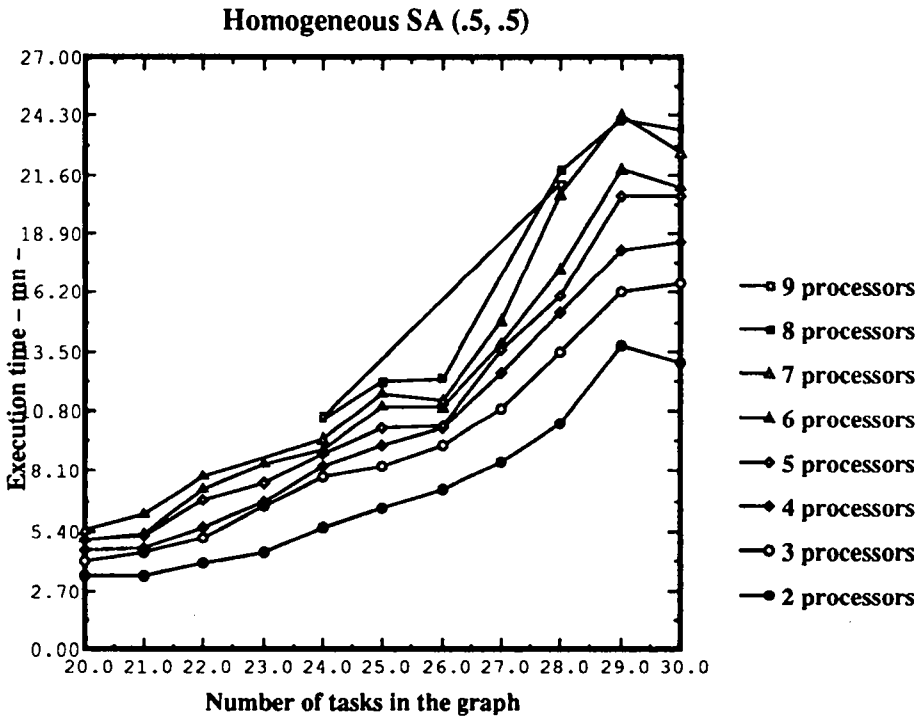

Figure 3: The average execution times of HSA(.5, .5).

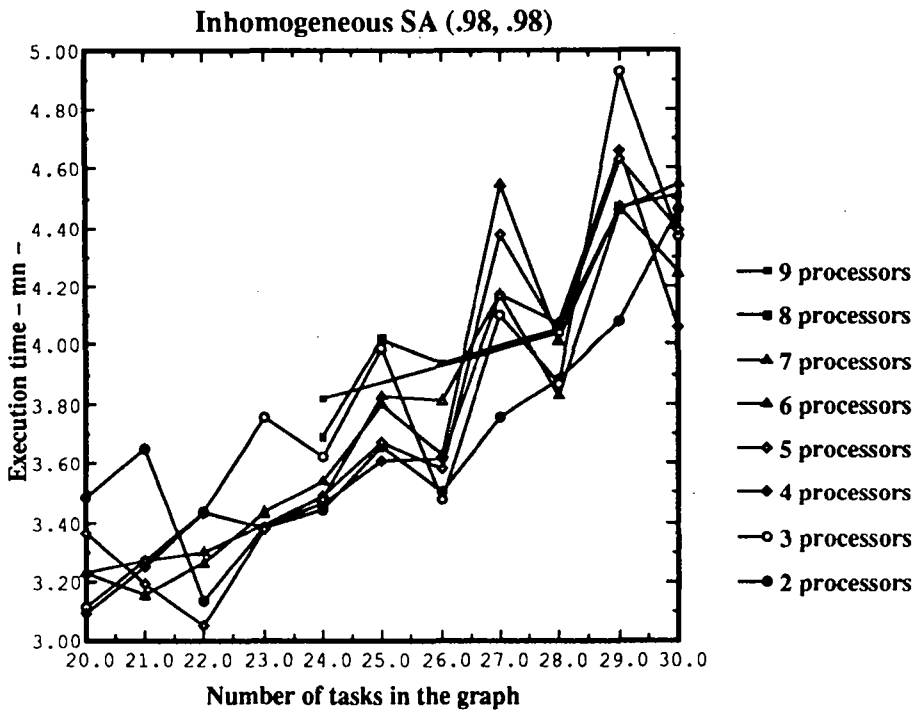

Figure 4: The average execution times of IHSA(.98, .98).

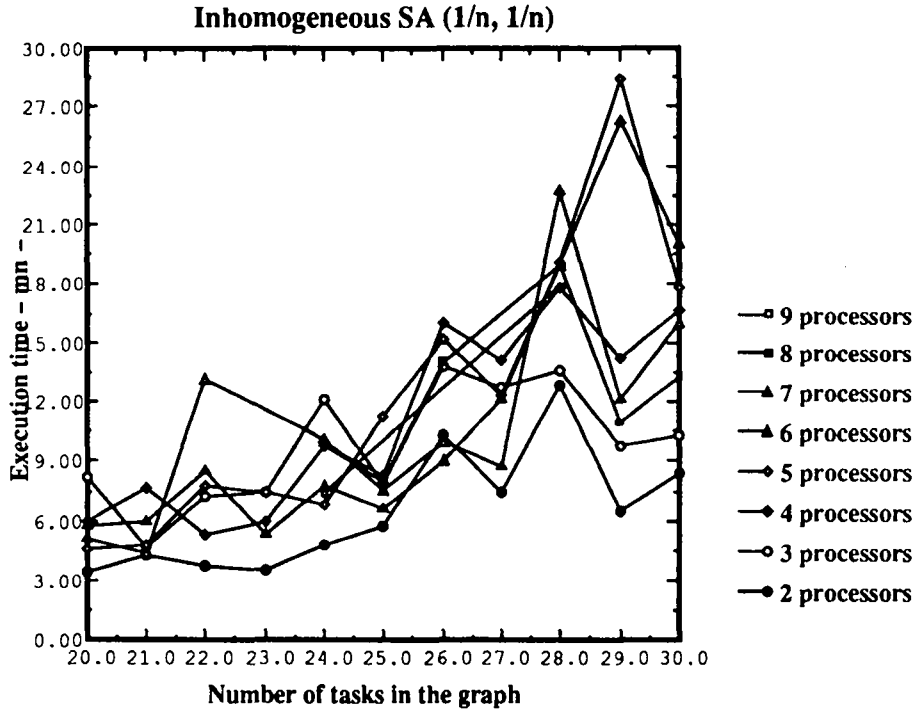

Figure 5: The average execution times of IHSA(1/n, 1/n).

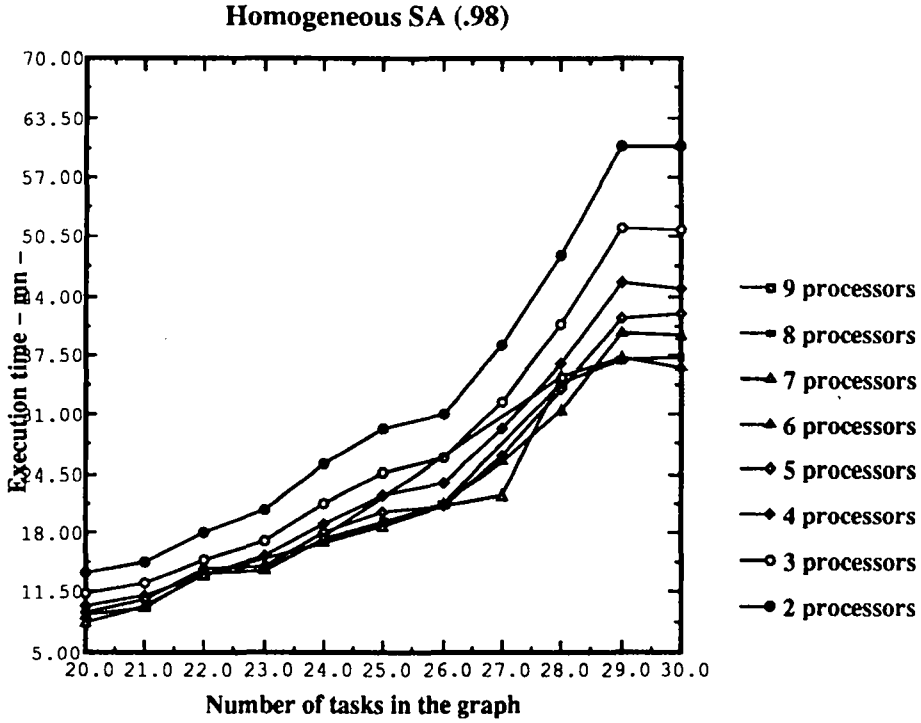

Figure 6: The average execution times of HSA(.98).

a similar effectiveness. As a consequence, the heuristics, provided they are used together, are not only efficient but also very effective.

6 Conclusions

In this paper, we have evaluated, by means of computational experiment, the effectiveness of 27 heuristics and 7 SA schemas used for the minimization of makespan. Our experiment has shown that efficient algorithms exist for finding optimal or near optimal solutions.

In case the task assignment is predefined, the Least Schedule Flexibility policy is the most effective among the 27 heuristics on average. The homogeneous SA algorithm finishes within a reasonable time and yields quite good results for most of the problems. When the heuristics are used together for problem resolution, the homogeneous SA algorithm still overperforms the heuristics on average. However, for about 40% of the cases, the heuristics, provided they are used together, have the same solution as the SA algorithm. Moreover, the heuristics are much faster.

In case the task assignment has no constraints, the Longest Path policy is the most effective among the 27 heuristics on average. We have implemented two "mixed", three two-step and an

Table 7: Heuristics vs. SA algorithms

	average difference of makespan between heuristics and SA	overperformance rate of heuristics	overperformance rate of SA	equal performance rate
HSA (.95, Random)	-0.73	27.83	8.70	63.48
HSA (.95, LSF)	0.01	10.43	15.65	73.91
HSA (.5 .5)	-0.27	22.61	12.17	65.22
HHSA (.98 .98)	-0.60	25.22	7.83	66.96
HHSA (1/n, 1/n)	-0.15	13.04	15.65	71.30
HSA (.98)	0.17	5.22	20.87	73.91

one-step SA algorithms. We have compared these SA algorithms with the 27 heuristics. We have shown that the one-step homogeneous SA algorithm is the most effective among the six annealing processes and that all the annealing algorithms overperform the heuristics. Nevertheless, when the heuristics are used together, they provide the same solutions as the SA algorithms for about 70% of the cases. Moreover, the total execution time of the heuristics is more than 100 times smaller than those SA algorithms which have a similar effectiveness.

In the implementation of the two-step annealing processes, in order for the empirical comparison be feasible, we have chosen the parameters of the algorithms which do not guarantee the convergence of the algorithms. This may explain the fact that these algorithms appear to be less effective than the one-step SA algorithm. However, if the parameters are appropriately chosen so that the convergence is ensured, one may expect that, at the expense of the execution time, they provide optimal or near-optimal solutions.

References

[1] T. L. Adam, K. M. Chandy, J. R. Dickson, "A Comparison of List Schedules for Parallel Processing Systems", *Comm. ACM*, Vol. 17, pp. 685-690, 1974.

[2] E. G. Coffman Jr., R. L. Graham, "Optimal Scheduling for Two-Processor Systems", Acta Informatica Vol. 1, pp. 200-213, 1972.

[3] E. G. Coffman Jr., Z. Liu, "On the Optimal Stochastic Scheduling of Out-Forests", Rapport INRIA No. 1156, 1990, also to appear in the *Oper. Res.*.

[4] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies", *SIAM J. Appl. Math.*, Vol. 17, No. 2, pp. 416-429, 1969.

[5] B. Hajek, "Cooling Schedules for Optimal Annealing", *Mathematics of Operations Research*, Vol. 13, No. 2, may 1988.

[6] B. Hajek, G. Sasaki, "Simulated Annealing - to cool or not", *Systems and Control Letters*, Vol. 12, pp. 443-447, 1989.

[7] Y. C. Hu, "Parallel Sequencing and Assembly Line Problems", *Operations Research*, Vol. 9, 841-848, 1961.

[8] S. Kirkpatrick, C. D. Gellat Jr., M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, Vol. 220, pp. 671-680, 1983.

[9] P. J. M. van Laarhoven, E. H. L. Aarts, "Simulated Annealing: Theory and Applications", *D. Reidel Publishing Company*, 1987.

[10] J. K. Lenstra, A. H. G. Rinnooy Kan, "Complexity of Scheduling under Precedence Constraints", *Operations Research*, Vol. 26, No. 1, pp. 22-35, 1978.

[11] Z. Liu, J. Labetoulle, "A Heuristic Method for Loading and Scheduling Flexible Manufacturing Systems", *Proc.of the Intern. Conf. Control 88*, London, IEE Conference Publication, No. 285, pp. 195-200, 1988.

[12] V. M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems", *Proc. of the Int. Conf. on Distributed Computing Systems*, San Francisco, California, pp. 30-39, May 1984.

[13] C. V. Ramamoorthy, K. M. Chandy, M. J. Gonzalez Jr., "Optimal Scheduling Strategies in a Multiprocessor System", *IEEE Transactions on Computers*, Vol. C-21, No. 2, pp. 137-146, 1972.

[14] J. Ramanujam, F. Ercal, P. Sadayappan, "Task Allocation by Simulated Annealing", *Proceedings of the Third Intern. Conf. on Supercomputing*, Vol 3, 471-480, 1988.

[15] F. B. Talbot, J. H. Patterson, W. V. Gehrlein, "A Comparative Evaluation of Heuristic Line Balancing Techniques", *Management Science*, Vol. 32, No. 4, pp. 430-454, 1986.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249 - 6399