Static Multiprocessor Task Graph Scheduling in the Genetic Paradigm: A Comparison of Genotype Representations

Pascal Rebreyend, F. E. Sandnes, G. M. Megson

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité de recherche associée au CNRS n° 1398

Static Multiprocessor Task Graph Scheduling in the Genetic Paradigm: A Comparison of Genotype Representations [2]

P. Rebreyend
F. E. Sandnes[1]
G.M. Megson[1]

F. E. Sandnes is supported by a studentship provided by The Research Council of Norway

[1] Parallel, Emergent and Distributed Architecture Laboratory (PEDAL)
Department of Computer Science, The University of Reading
Reading, RG6 2AY, United Kingdom

[2]This work was partially supported by the HCM project
SCOOP - Solving Combinatorial Optimisation Problems in Parallel -
of the European Union.

May 1998

Research Report N° 98-25

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.00
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

Static Multiprocessor Task Graph Scheduling in the Genetic Paradigm: A Comparison of Genotype Representations [2]

P. Rebreyend
F. E. Sandnes[1]
G.M. Megson[1]

F. E. Sandnes is supported by a studentship provided by The Research Council of Norway
[1] Parallel, Emergent and Distributed Architecture Laboratory (PEDAL)
Department of Computer Science, The University of Reading
Reading, RG6 2AY, United Kingdom
[2]This work was partially supported by the HCM project
SCOOP - Solving Combinatorial Optimisation Problems in Parallel -
of the European Union.

May 1998

Abstract

In the NP-hard *multiprocessor scheduling problem* a set of precedence constrained tasks are allocated onto processors in a processing order in order to minimise the makespan. Many heuristic methods for finding solutions exist, but they are all sub-optimal on general task graphs. To improve these solutions, genetic algorithms have successfully been applied to the problem and the results reported have been superior to the list-scheduling approaches. However, the application of genetic algorithms to the multiprocessor scheduling problem have predominantly followed two main paths of developments, namely the use of direct and indirect representations. In the direct chromosome representation the schedule is represented and manipulated directly by the genetic operators, and the genotype is identical to the phenotype. In the indirect representation only the decisions on how to build the schedule is encoded in the chromosome. The genetic operators affect the schedules implicitly, and the genotype is different to the phenotype. In this paper these two main approaches to genetic scheduling are compared by evaluating their respective quality of results and time of convergence.

Keywords: Multiprocessor Scheduling, Genetic Algorithms, direct and indirect representation

Résumé

Dans le problème NP-dur de l'ordonnancement sur machine parallèle, un ensemble de tâches avec des contraintes de précédences sont ordonnancées sur un ensemble de processeurs afin de minimiser la durée totale d'exécution. Plusieurs méthodes heuristiques existent pour trouver des solutions sous-optimales. Pour améliorer ces solutions, les algorithmes génétiques ont été utilisés avec succès et les résultats sont meilleurs que ceux obtenus avec des algorithmes de listes. Cependant, cette utilisation d'algorithmes génétiques pour le problème de l'ordonnancement sur machine parallèle s'est fait suivant deux voies principales: la représentation directe et celle indirecte. Dans le cas de la représentation directe, l'ordonnancement est représenté directement par le chromosome et manipulé aussi de manière directe par les opérateurs génétiques. Le génotype est donc identique au phénotype. Par contre, dans le cas de la représentation indirecte, le chromosome code seulement des décisions sur comment construire la solution. Les opérateurs génétiques affectent donc la solution de manière implicite, indirecte et donc le génotype est différent du phénotype. Dans ce rapport, les deux méthodes sont comparées.

Mots-clés: ordonnancement de programme parallèle sur machines multiprocesseurs, algorithmes génétiques, représentation directe et indirecte.

1 Introduction

Static multiprocessor task graph scheduling belong to one of the many combinatorial optimisation problems and numerous scheduling algorithms have been published [4, 8, 6, 17, 19]. The static multiprocessor scheduling problem involves allocating processing order and processor elements to a set of precedence constrained tasks.

The quality of a schedule is evaluated quantitatively using the *makespan* or response time, which is defined as the difference between the start-time of the earliest task and the finishing-time of the latest task. Clearly, the makespan is sensitive to variations in processing order and processor allocations. The *multiprocessor scheduling problem* therefore consist of finding the configuration that minimises the makespan. The most widely referenced constructive method is the Critical Path - Most Immediate Successor First CP/MISF [13]. The problems with this and the other traditional constructive (list- scheduling) methods are the poor quality of the sub-optimal solutions (i.e., solutions whose optimality cannot be guaranteed) produced. Because of the inadequacy of these algorithms genetic algorithms have successfully been employed in the search for the sub-optimal solution [1, 2, 3, 5, 7, 10, 11, 12, 14, 20, 23, 21, 22, 24]. A genetic algorithm is a guided random parallel search strategy where elements (called *individuals*) in a given set of solutions (called *population*) are randomly combined and modified by genetic operators such as *crossover* and *mutation*, respectively until some termination condition is satisfied. The population evolves iteratively through series of *generations* in order to improve the *fitness* of its individuals. A chromosome structure is also called a genotype, and a solution structure is often called a phenotype.

The genetic scheduling attempts can be classified into two major categories, direct and indirect representations. In the direct representation the genotype is identical to the phenotype and the genetic operators are manipulating the schedules directly. In the indirect representation the phenotype is different from the genotype, and the chromosome contains decisions on how to build the schedule. As a result the genetic operators only affect the phenotype implicitly. The direct representation was introduced by Hou et. al. [10] and later modified by the likes of Greenwood et. al. [12], Baccouche [2] and Rebreyend [5]. Early indirect representations were proposed by Benten and Sait [3] and Kidwell [14] and later modifications are introduced by the likes of Ahmad and Dhodhi [7, 1] Ravikumar and Gupta [20] and Sandnes and Megson [21, 22].

In this paper we compare these two major strategies. There are two main criteria for comparing genetic algorithms, namely the quality of the results and the effort of computation. These criteria can be materialised quantitatively using the response-time as a qualitative measure and the time of convergence as a measure of effort. The main emphasis is on the state-of-the-art, namely Rebreyend et. al.'s enhanced direct representation strategy and Sandnes and Megson's modified indirect representation, but the results obtained with Hou et. al. and Ahmad and Dhodhi are included for completeness.

This paper is organised as follows. First the scheduling problem is defined formally. Thereafter, the different approaches are surveyed. This is followed by experimental evidence and a discussion. The paper is closed by a set of concluding remarks.

2 Multiprocessor scheduling

A *(homogeneous) multiprocessor system* is composed of a set $\mathcal{P} = \{p_1, \cdots, p_m\}$ of m identical processors. These processors are fully or partially inter-connected in a network, where all links are identical. Each processor can execute at most one task at a time and tasks can not be pre-empted. A processor can communicate asynchronously through one or several of its links simultaneously. Processors are fully connected.

The parallel program 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, i.e. the precedence relation limits the processing order of the tasks . An arc $(t_{i_1}, t_{i_2}) \in A$ represents the fact that at the end of its execution, t_{i_1} sends a message whose contents are required by t_{i_2} to start execution. In this case, t_{i_1} is said to be an *immediate predecessor* of t_{i_2}, and t_{i_2} itself is an *immediate successor* of t_{i_1}.

A *path* is a sequence of nodes $< t_{i_1}, \cdots, t_{i_k} >$, $1 < k \leq n$ such that t_{i_l} is an immediate predecessor of $t_{i_{l+1}}$, $1 \leq l < k$. A task t_{i_1} is a *predecessor* of another task t_{i_k} if there is a path $< t_{i_1}, \cdots, t_{i_k} >$ in $\mathcal{D}$. To every task t_i, there is an associated weight representing its duration, known before the execution of the program. In addition, all the communications are also known at compile-time. Thus, to every arc $(t_{i_1}, t_{i_2}) \in A$ there is an associated weight representing the transfer time of the message sent by t_{i_1} to t_{i_2}.

Tasks without predecessors are called entry nodes and are to be executed first. Tasks without successors are called exit tasks and are executed last.

A *schedule* is a vector $s = \{s_1, \cdots, s_m\}$, 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 the *makespan*. A *list heuristic* is used 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 [5].

2.1 Genetic algorithms

A GA is a stochastic optimisation method [9, 18]. It maintains a set of *chromosomes* representing problem solutions. These chromosomes are modified using simulated evolution by employing genetic operators. The two main operators are crossover and mutation. The crossover attempts to adopt and combine characteristics from two parents in its offspring. The mutation introduces perturbation into the search. These modifications are applied through successive generations. *Selection* is applied to the individuals of one generation to form the population of the next generation. This allows the algorithm to take biased decisions favouring good individuals. For this, some of the more-fit individuals are replicated, while some of the less-fit individuals are filtered. As a consequence, after the selection, the population is likely to be "dominated" by good individuals.

3 Direct representations

In the Direct representation scheme, a chromosome represents a solution. A chromosome can be easily build from a solution and vice-versa. Hou, Ansari and Ren [10] used this scheme in the HAR algorithm. In the next paragraph we explain the HAR chromosome representation. This representation is adopted by Rebreyend, Correa and Ferreira [5] and their version called Lyon is described thereafter.

3.1 Representation

We have a solution for the scheduling problem when we know the processor and time allocated to each task. This means that the same solution always yields the same makespan.

A chromosome, like DNA, is often a string or a set of strings and operators use some properties of these strings. A string of elements defines an order on these elements. We can use it to express the order between tasks [5]. But, a string represents the execution order on one processor but not the starting time of each task. An execution order can represent several feasible solutions. Such solutions are made by delaying some task. But, as we focus on solutions with the lowest makespan, we only represent a subset of all the solutions. Given a chromosome, the solution is built using a list scheduler which schedules each task at the earliest possible time-slot. It is also possible to build a chromosome from a solution. Consequently, the search is reduced but optimal solutions can still be represented with this genotype as show in [5].

The initial version of HAR assumes zero-communication times. To allow for non-zero communication times the fitness function is extended.

3.2 HAR

The HAR algorithm employs the above representation. It is desirable to only represent feasible solutions, as this allows the genetic algorithm to converge onto quality solutions with less computational effort. If a task t_i is before t_j on one string and there is a dependency from t_j to t_i the chromosome represents an infeasible solution. In HAR and the Lyon algorithm, chromosomes only represent feasible solutions. This is achieved by using problem-specific genetic operators. These genetic operators can be slower, especially for larger problems. For example, if a mutation operator swaps two tasks, to conserve feasibility, these two tasks cannot be picked totally at random. The two tasks must be selected such that the offspring is feasible. HAR uses the levels of each task to achieve this. The level of a task is the longest path (i.e the number of tasks) from a task without predecessor to the given task. Task levels are used to rapidly build feasible schedules. It is trivial to show that the schedule on each processor is feasible if the order of the tasks is chosen according to their levels. The proof is simple: If a schedule is infeasible, it is because a task is scheduled before its predecessors. However, this is not possible since the level of the predecessors is strictly less than the level of the task.

Building the initial population is done as follows: For each level, tasks are assigned to a processor and this is done from level 0 to the highest level. When a task is assigned to a processor, it is added at the end of the current string

which represents the schedule at this processor. A similar approach applies to the crossover operator. The procedure is to cut the set of tasks into two parts, one which consists of tasks executing at the beginning and the other of tasks executed at the end. Thereafter, two new schedules can be built by swapping the last two parts between the two schedules. To guarantee feasibility, the members of the first partition represents tasks with a level smaller than a chosen level and the other partition contains the remaining tasks. For mutation, HAR only swaps two tasks at the same level.

3.3 Lyon

Rebreyend et. al. identified two shortfalls of HAR. First, HAR is sometimes unable to reach the optimal solution as shown in [5] and tasks are not uniformly distributed among processors in the initial population. Instead of using levels to ensure the feasibility of a schedule, Lyon uses the task graph. This is computationally more costly but at least one optimal solution can be represented. They prove that a schedule s is feasible if and only if $D(s)$ is acyclic, where $D(s)$ is the graph D of precedence constraints augmented by the set of precedence constraint given by the schedule (When a task is scheduled before another on the same processor).

The initial population is randomly chosen by using a list algorithm. The Lyon algorithm uses knowledge augmented operators. Traditional genetic algorithms are based on a random search and can often be slow in practice. To improve the speed of the search, additional knowledge can be used by incorporating it into the operators.

The crossover operator first cuts all strings in two parts as done in HAR. But, instead of using levels, the cut is chosen by using the digraph D: At each step, a free task (not already assigned to a part) is assigned to a part and all predecessors (or successors), both from D or from a string are assigned to the same part. This is repeated until all tasks are assigned to a part. The left part remains unchanged and the right part is built using the right end-part of the other parent. To build the right part, a greedy algorithm is used and to maintain characteristics of the other parent, a precedence constraint is added to D between two tasks if and only if these two tasks are scheduled adjacently on the same processor. The first step is to build all feasible task-processor couples. In the second step we determine the subset of couples with the earliest start-time. The third step selects couples with the longest critical path. Finally, the couples with the highest number of descendants remain. From this final set one couple is picked at random. The mutation operator constructs the schedule in a similar manner to the right part of the crossover. However, only the two first steps are performed.

3.4 Other uses of the direct representation

Both Greenwood et. al. [12] and Baccouche and Muntean [2] are inspired by the work of Hou et. al. and thus adopt their problem encoding. However, their objectives are slightly different as they are scheduling real-time tasks. Besides the precedence constraints each task has a periodic deadline, thus the genetic algorithm has to satisfy a set of timing constraints. Moreover, in a fault-tolerant real-time systems, a subset of the tasks must be replicated onto different

processors and child nodes of the replicated nodes are responsible for voting and thus detect and correct failures. The genetic algorithm has to adopt to a set of replication constraints such that the schedules are made fault-tolerant. Because of these additional constraints both approaches introduce invalid schedules, thus schedules that contain tasks that do not meet their deadlines, and tasks that are not replicated according to the replication constraints. These difficulties are overcome by penalising such solutions, such that they eventually are removed from the population by the genetic engine.

4 Indirect representations

In the indirect chromosome representation the phenotype is constructed using the genotype. The genotype is a set of symbols that are used by some decoding algorithm to build the schedule.

4.1 Previous work

Benten and Sait [3] only encode the processor allocations in the genetic algorithm. The time allocation is performed by building a list schedule, where all the processor allocations are determined by the chromosome string. This encoding will always produce valid schedules when using standard one-point order crossover, however just a limited set of the search space is considered, constrained by the list scheduler. Kidwell [14] also encode the processor allocations in the genetic algorithm in a similar manner to Benton and Sait.

Ravikumar and Gupta [20] fix the number of processors but try to find the optimal inter-processor connection topology. Their chromosome consists of three parts, each with its own genetic operators. The first part of the chromosome is a $p \times p$ connectivity matrix. The second part of the chromosome is an integer array of size n that comprises the processor allocations, where element i denote the processor allocated to task i. The third part of the chromosome consists of the scheduling information in an integer array of size n. This is encoded as a list of priorities, where element i contain the priority of task i. Schedules are built using the processor allocations encoded in the chromosome, and using a priority list scheduler, where the priorities encoded in the chromosome are used (Increasing priority order). The crossover operator for the link assignments copies the $p \times p$ matrices from the parents to the offsprings, then a random matrix row is exchanged between the two offsprings and rotated to maintain symmetry. No mutation operator is applied to the link assignments. The crossover operator for the processor assignments uses a one point order crossover. Mutation is achieved by random exchange. The priority list crossover operator is constructed by selecting a random time t. Two lists are created, such that each contains the list of nodes in the parents that could have been fired at time t. These two lists are replaced for the two offsprings. Mutation is achieved by selecting one of the n tasks at random, say task t. The task t' among the predecessors of t that have the latest schedule time is determined, similarly, the task t'', which has the earliest schedule time among the successors of t is found. Clearly, the position of task t in the ready list can be varied in the range $t' + 1$ to $t'' - 1$ without violating any constraints. The mutation operator assigns a position for t randomly in the above range. This may necessitate the modification of

the priorities of the other tasks in the range in a domino like fashion. The authors claim that their approach is suitable for heterogeneous systems, as each processor is assigned a class number, and each task is assigned a class set that indicates which processor the task can run on.

Dhodhi and Ahmad et al. [7] attempt to use genetic algorithms to determine the optimal number of processors for a given task graph in a heterogeneous system. The first part of the chromosome indicates the availability of processors. This is implemented as a boolean array that indicates the presence or absence of a particular processor. Note that each processor can be different, hence the heterogenicity. The second part of the chromosome consists of a priority list, where each allele represents the priority for the task with that index. When decoding the chromosome, this priority is used to build the schedule using priority list scheduling. A separate single point crossover and mutation approach is applied to each of the two parts of the chromosome.

Ahmad and Dhodhi [1] also used a chromosome representation where only task priorities are encoded. The schedules are built by scheduling all tasks in the freelist in bulk in their order of priority, before the freelist is updated. Processor assignments are made by assigning a task the processor with the earliest possible start time. They report to have better results than Hou et al. but do not consider communication overheads.

4.2 Reading approach

Sandnes and Megson's [23] improved scheduling approach is based on the implicit representation proposed by Ahmad and Dhodhi due to its high quality results (See the experiments). It was extended to include non-zero communication overheads and partially connected processor networks through an extended chromosome representation.

Ahmad and Dohdi proposed to allocate processors implicitly using an earliest start-time heuristic [1, 7]. Their original representation consists only of a priority list of size N. A schedule is built by scheduling all tasks in the freelist. Each task is allocated to the processor that offers the earliest start-time taking the precedence constraints into consideration.

4.2.1 Non-zero communication costs

This processor allocation heuristic is extended to incorporate communication overheads and partially connected processor networks. The proposed extended processor allocation heuristic is to allocate a given task to the processor that offer the earliest start-time. The earliest start-time is found by computing the earliest start-time on each processor and choosing the smallest. The earliest start-time on a processor is computed by finding the maximum of the finishing-times of the parent tasks plus the product of the communication time and the number of relays the message has to make.

The genetic algorithm can adapt schedules for different network structures, and the network topology must be incorporated into the scheduling model. A network connectivity matrix can be used to accomplish this, where the elements denote the cost of transmitting unit data between two processors. A zero element means that there is no link connecting the two processing elements, and messages must travel via intermediate nodes. Most networks of interests are

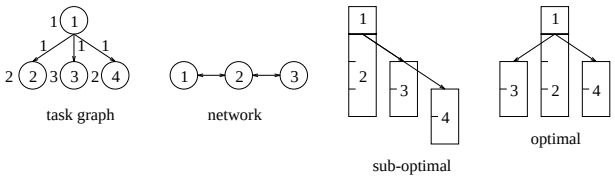

Figure 1: Shortfall of implicit processor allocation.

partially-connected; therefore, some non-diagonal elements in the matrix are zero. In a partially connected network a message is relayed through the nodes of a path connecting the two communicating processors.

4.2.2 Expanding the search space

Situations occur where the implicit processor allocation strategy is sub-optimal. Imagine a scenario where the earliest possible start time is the same on several processors. For instance, this occurs when scheduling the first task as all the start-times are zero. The implicit processor allocation strategy then selects a processor from the set of feasible processors using some fixed strategy. For example, the strategy might be to select the available processor with the smallest index. Such processor assignments limit the search space.

For example, a three-processor star topology, and a task graph with four nodes as depicted in Figure 1. When scheduling task 1 with the implicit processor allocation strategy one processor is chosen according to some fixed policy, such as the processor with the smallest index, here processor 1. However, in this example task 1 must be allocated processor 2 in order to find the optimal solution with makespan 4. If task 1 is allocated processor 1 or 3 the best solution will have at best a sub-optimal makespan of 5. Hence, the static processor allocation policy that selects from a set of processors with equal smallest start-times can lead to sub-optimal solutions, especially on asymmetric network topologies.

This shortfall is amended by modifying the implicit processor assignment representation. A second chromosome part is added, namely a processor choice part. This part is an integer string of size N where the genes are in the range $[1..p]$. This part of the chromosome is used when an ambiguous situation occurs, i.e. when there is a tie between two or more processors. If e is the set of processors with the earliest start time for task i, then processor $e_{\chi mod |e|}$ is chosen, where χ is the gene value. In other words, the decision concerning which processors to chose in ambiguous situations is encoded in the chromosome, instead of choosing one statically or randomly. This safeguard reduces the chances of converging at sub-optimal solutions.

The advantage of this simple chromosome representation is that a standard genetic search engine can be applied to the problem with simple computationally efficient operators. Sandnes and Megson used Goldberg's PMX crossover and random swap mutation for the priority part of the chromosome and uniform crossover and random-increment-decrement for the processor allocation part of the chromosome.

5 Experimental evidence

5.1 The test-suite

Three test-suites where used: First a set of general task-graphs generated by a tool called ANDES-Synth [15, 16], a set of state-space controllers from Sandnes and Megson [21, 22] and the standard Stanford arm and elbow manipulator task graphs of Kasahira and Narita [13] with added communication overheads. ANDES-Synth is a tool that generates synthetic task digraphs whose shapes represent known parallel programs, such as divide and conquer, prolog solving and Gauss elimination. The parameters of the ANDES-Synth test-suite represents the characteristics on an IBM SP-1 parallel computer.

1	*Bellford*
2	*Diamond1*
3	*Diamond2*
4	*Diamond3*
5	*Diamond4*
6	*Divconq*
7	*FFT*
8	*Gauss*
9	*Iterative*
10	*MS-Gauss*
11	*Prolog*
12	*QCD*
13	*elbow manipulator*
14	*Stanford arm*
15	*SSC*

Five algorithms are applied to the test-suite and compared: 1) CP/MISF - the heuristic is run iteratively where ambiguous choices are resolved randomly, i.e. guided random search, 2) HAR - Hou et. al.'s GA with non-zero communication costs. 3) Lyon GA, 4) Ahmad's GA with non-zero communication costs and 5) Reading GA. Each trial is run with exclusive access to an Intel Pro 200 for two hours.

6 Discussion

The table 3 summarises the results. It shows the relative deviation in percentage from the best for all methods applied to all the problems. The best result for each problem is highlighted in boldface.

Clearly, HAR produces the worst results. These results are even worse than the results obtained with guided random search. The reasons for these poor results are explained in detail in [5]. They can be summarised as being due to a limited search space caused by the genetic operators and a non-uniform distribution of tasks to processors in the initial population.

The results are relatively consistent, where the Lyon, Ahmad and Reading approaches produce results of similar quality although with slight variations.

Graphs		size	NON GA	GA			
			CP/MISF	Direct		Indirect	
				HAR	Lyon	Ahmad	Reading
1	-m	365	**71,936,050**	269,190,150	76,124,350	76,512,200	78,994,250
	-l	992	**194,182,200**	716,522,200	206,050,150	228,000,650	223,578,850
2	-m	258	**131,940,750**	225,332,550	**131,940,750**	134,422,800	134,422,800
	-l	1026	286,453,550	937,634,850	300,803,550	**281,255,850**	289,169,200
3	-m	486	**128,769,500**	326,383,350	162,952,700	231,852,500	238,133,400
	-l	1227	**254,504,850**	787,740,800	273,341,400	424,524,050	422,459,250
4	-m	731	180,576,400	556,004,100	**179,567,350**	194,562,400	212,661,500
	-l	1002	229,754,350	720,136,400	**228,796,900**	248,498,500	282,471,300
5	-m	731	**132,875,550**	336,299,850	145,427,450	153,293,850	159,215,400
	-l	1003	**164,264,000**	435,479,000	173,383,850	185,391,100	200,830,400
6	-m	384	108,044,840	275,053,700	**99,098,490**	107,072,860	107,114,140
	-l	766	290,972,340	514,613,720	**171,301,530**	195,686,370	213,520,410
7	-m	194	**29,888,250**	66,772,300	30,650,550	**29,888,250**	30,560,950
	-l	1026	108,813,250	421,553,650	**102,736,900**	118,611,500	126,212,450
8	-m	782	**237,709,950**	615,097,400	239,739,000	240,809,300	271,997,050
	-l	1227	**325,456,750**	976,564,500	326,068,350	353,580,250	367,395,100
9	-m	262	22,824,200	51,381,250	22,099,550	**16,793,350**	**16,793,350**
	-l	938	51,524,200	179,685,100	52,661,200	**47,936,700**	48,363,350
10	-m	768	4,633,100,500	9,920,502,750	4,628,742,350	**4,598,559,550**	**4,598,559,550**
	-l	1482	2,936,022,450	9,234,372,700	2,915,146,000	**2,570,505,150**	f2,795,218,300
11	-m	214	80,436,450	186,779,350	**63,292,900**	63,445,800	63,522,250
	-l	1313	717,576,450	1,039,016,450	**270,162,250**	299,015,150	311,565,450
12	-m	326	1,176,047,050	3,566,221,900	1,176,047,050	**1,173,100,600**	**1,173,100,600**
	-l	1027	3,928,300,600	11,127,636,700	3,913,568,350	**2,038,576,050**	**2,038,576,050**
13		103	**6630**	6864	**6630**	**6630**	**6630**
14		90	668	844	647	**627**	639
15	5	200	110	151	**108**	109	110
	6	288	126	177	127	**125**	127
	7	392	**139**	253	140	144	142
	8	512	**164**	289	169	173	172
	9	648	**173**	313	178	179	180

Table 1: Results in 2 hours

The Lyon approach produce the best results on average, followed by Ahmad and Reading. The Ahmad approach is the most reliable as it has the lowest maximum deviation from the best solution found, thus it is more likely to produce good results in most situations. The Reading approach produces slightly worse results than Ahmad. This is because Reading is an extension of Ahmad specially designed for a network of partially connected processors. In these tests the problems were evaluated on a 16-processor fully connected network. However, Sandnes and Megson shows that Reading outperforms Ahmad on a partially connected processor networks.

An interesting observation is that the CP/MISF heuristic with random search scores the highest in number of best solutions as about half of the best solutions are found by this method, although it ranks as second worst on average quality and maximum deviation from the best. This is an indication that a heuristic method with random search is a competitive strategy, although not as

reliable as the genetic algorithms. With random search one is likely to occasionally hit a desirable location in the search space, likewise there is a probability that such a region of the search space will never be traversed. The genetic algorithm however, takes the best from two worlds, it starts with a random search in the initial population, and then apply its stochastic operators to refine these solutions. Consequently, the genetic algorithm almost always converges at a high quality solution.

An exception to the general results can be seen for the graph *12*. For this graph, the two indirect methods give superior results to the other approaches. Repeated runs of the various algorithms reveal this predominant pattern, and we are unable to explain this phenomena.

Table 3 shows that the indirect representation is efficient. It is relatively simple to implement. One other hand, the direct representation is non-trivial to implement since one relies on non-standard computationally-costly genetic operators with built-in heuristics. However, operating directly on the phenotype is flexible and allows for certain extensions such as genetic operators with different heuristics to be made.

Figures 2 and 3 shows the converging characteristics of the different approaches, where fitness is plotted against a log-scale of time in seconds. Clearly, the Lyon, Ahmad and Reading GA's converge in a similar manner and HAR appears to converge prematurely. Note that Lyon in Figure 3 obtain the first solution after 10 seconds. This delay is due to setup and initialisation overheads.

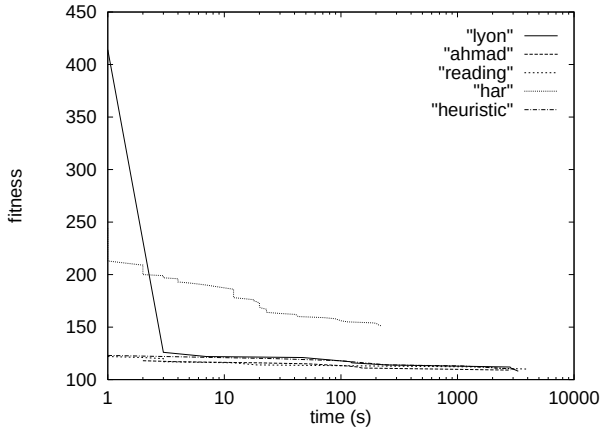

Figure 2: Convergence of methods on the *15* graph.

7 Conclusion

This paper investigates the two main directions of research into genetic algorithms applied to static multiprocessor scheduling, namely the direct representation proposed by Hou et. al and the indirect representations. This paper verifies that genetic algorithms are effective in the search for high quality schedules. The experiences obtained with both approaches indicate that genetic algorithms can be applied with little knowledge about a problem and still produce acceptable solutions, given adequate computational effort. However, both camps of re-

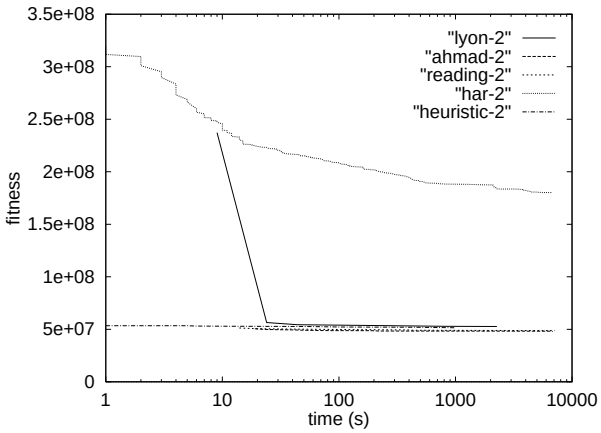

Figure 3: Convergence of methods for the *7* graph

search has revealed the fact that combining the genetic algorithm with domain specific knowledge greatly improves the quality of the schedules and reduces the time of convergence. In multiprocessor scheduling this imply the assignment of processing order and processor elements to the tasks.

The first attempts at applying genetic algorithms to multiprocessor scheduling varied greatly in quality. However, as more domain specific information is incorporated into the search both approaches tends towards the same lower bounds with respect to quality.

The experiments also show that the time of convergence are similar for the two main approaches where the indirect method is slightly faster than the direct method. This is most likely due to the added complexity of the non-trivial augmented operators used with the direct method.

In general the indirect method is simple and straightforward in structure. The direct method require more implementation effort, however it is more flexible.

References

[1] Imtiaz Ahmad and Muhammed K. Dhodhi. *Multiprocessor Scheduling in a Genetic Paradigm.* Parallel Computing, vol 22, pp395-406, 1996.

[2] L. Baccouche. *Efficient Static Allocation of Real-Time Tasks Using Genetic Algoritms.* Internal Report, Imag Institute, Laboratoire de génie informatique, 1995.

[3] Muhammad S. T. Benten and Sadiq M. Sait. Genetic scheduling of task graphs. *International Journal of electronics*, 77(4):pp401–415, 1994.

[4] Thomas L. Casavant and Jon G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):pp141–154, 1988.

[5] R. C. Correa, A. Ferreira, and P. Rebreyend. *Integrating List Heuristics into Genetic Algorithms for Multiprocessor Scheduling.* IEEE Symposium on Parallel and Distributed Processing 1996, pp462-469, 1996.

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

[7] Muhammad K. Dhodhi, Imtiaz Ahmad, and Robert Storer. Shemus: synthesis of heterogeneous multiprocessor systems. *Microprocessors and Mircosystems*, 19(6):pp 311–319, 1995.

[8] E. G. Coffman et al. eds. *Computer and Job-Shop Scheduling Theory.* Wiley, 1976.

[9] David E. Goldberg. *Genetic Algorithms in Search, Optimisation, and Machine Learning.* Addison-Wesley Publishing Company INC, 1989.

[10] Edvin S. H. Hou, Nirwan Ansari, and Hong Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):pp113–120, 1994.

[11] S. Hurley. Taskgraph mapping using a genetic algorithm: A comparison of fitness functions. *Parallel Computing*, 19:pp1313–1317, 1993.

[12] S. Hurley. *An Evolutionary Strategy for Scheduling Periodic Tasks in Real-Time Systems.* In *Applied Decision Technologies*, pages 171–188, 1995.

[13] Hironori Kasahara and Seinosuke Narita. Parallel processing of robot-arm control computation on a multiprocessor system. *IEEE Journal of Robotics and Automation*, RA-1(2):pp104–113, 1985.

[14] Michelle D. Kidwell. *Using Genetic Algorithms to Schedule Distributed Tasks on a Bus-Based System.* 1st International Conference on Genetic Algorithms, pp368-374, 1993.

[15] J. Kitajima and B. Plateau. *Modelling parallel program behaviour in ALPES.* Information and Software Technology, 36(7), pp457-464, 1994.

[16] J. Kitajima, C. Tron, and B. Plateau. *ALPES: A tool for the performance evalutation of parallel programs.* In Environments and Tools for Parallel Scientific Computing, J. J. Dongarra and B. Tourancheau, Eds., Advances in Parallel Computing, vol 6, pp213-228, 1993.

[17] Brian A. Malloy, Errol E. Lloyd, and Mary Lou Soffa. Scheduling dag's for asynchronous multiprocessor execution. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):pp498–508, 1994.

[18] Zbigniew Michalewicz. *Genetic Algorithms+Data Structures=Evolution Programs,3rd edition.* Springer, 1996.

[19] M. Norman and P. Thanisch. *Models of machines and computations for mapping in multicomputers.* ACM Computer Surveys, 25(9), pp263-302, 1993.

[20] C. P. Ravikumar and A. K. Gupta. Genetic algorithm for mapping tasks onto a reconfigurable parallel processor. *IEE Proc-Comput. Digit. Tech*, 142(2):pp81–96, 1995.

[21] F. E. Sandnes and G. M. Megson. *Genetic Algorithms Applied to Automatic Parallel Controller Code Generation*. in Proceedings of the 2nd ICSC International Synopsium on Soft Computing, B114-B120, 1996.

[22] F. E. Sandnes and G. M. Megson. *A Hybrid Genetic Algorithm Applied to Automatic Parallel Controller Code Generation*. IEEE Proceedings of the 8'th Euromicro Workshop on Real-Time Systems, pp70-75, 1996.

[23] F.E. Sandnes and G.M. Megson. *Improved Static Multiprocessor Scheduling Using Cyclic Task Graphs - A Genetic Approach*. IPPS'97 Workshop on Randomised Parallel Computing, 1997.

[24] Pai-Chou Wang and Willard Korfhage. *Process Scheduling Using Genetic Algorithms*. IEEE Symposium on Parallel and Distributed Processing, pp638-641, 1995.

Graphs		NON GA	GA			
		CP/MISF	Direct		Indirect	
			HAR	Lyon	Ahmad	Reading
1	-m	2362	6577	6144	7079	5261
	-l	821	6928	4668	1575	7193
2	-m	838	6988	7037	1877	330
	-l	1072	6306	2209	2451	4923
3	-m	815	7043	1838	7052	2201
	-l	4022	6601	2950	6105	5511
4	-m	854	3651	2619	4767	1201
	-l	2851	5539	4329	6515	1196
5	-m	1818	7015	5211	6385	6534
	-l	6840	6953	878	2101	7045
6	-m	1416	5426	3771	658	2762
	-l	127	6322	1424	6906	5045
7	-m	1237	908	692	250	4833
	-l	6176	65	1843	5005	3656
8	-m	1998	6415	2723	3087	7053
	-l	171	6452	6395	7105	6469
9	-m	406	6751	807	34	59
	-l	980	6602	2266	6826	7082
10	-m	6522	7178	2183	6	6
	-l	5516	7088	1242	6022	4098
11	-m	188	1082	4285	1804	2802
	-l	6894	7015	3800	5776	6447
12	-m	25	1516	31	4	4
	-l	786	4688	5392	20	20
13		1	2519	1	230	5
14		6960	3136	1534	2056	3676
15	5	3071	224	3324	2849	4066
	6	5815	6480	2802	5430	3951
	7	909	1474	4176	482	3781
	8	460	6125	2482	988	699
	9	4939	7141	2851	5525	3808

Table 2: Time of the results

Graphs		NON GA	GA			
		CP/MISF	Direct		Indirect	
			HAR	Lyon	Ahmad	Reading
1	-m	**0.00**	274.21	5.82	6.36	9.81
	-l	**0.00**	268.99	6.11	17.42	15.14
2	-m	**0.00**	70.78	**0.00**	1.88	1.88
	-l	1.85	233.37	6.95	**0.00**	2.81
3	-m	**0.00**	153.46	26.55	80.05	84.93
	-l	**0.00**	209.52	7.40	66.80	65.99
4	-m	0.56	209.64	**0.00**	8.35	18.43
	-l	0.42	214.75	**0.00**	8.61	23.46
5	-m	**0.00**	153.09	9.45	15.37	19.82
	-l	**0.00**	165.11	5.55	12.86	22.26
6	-m	9.03	177.56	**0.00**	8.05	8.09
	-l	69.86	200.41	**0.00**	14.24	24.65
7	-m	**0.00**	123.41	2.55	**0.00**	2.25
	-l	5.91	310.32	**0.00**	15.45	22.85
8	-m	**0.00**	158.76	0.85	1.30	14.42
	-l	**0.00**	200.06	0.19	8.64	12.89
9	-m	35.91	205.96	31.60	**0.00**	**0.00**
	-l	7.48	274.84	9.86	**0.00**	0.89
10	-m	0.75	115.73	0.66	**0.00**	**0.00**
	-l	14.22	259.24	13.41	**0.00**	8.74
11	-m	27.09	195.10	**0.00**	0.24	0.36
	-l	165.61	284.59	**0.00**	10.68	15.33
12	-m	0.25	204.00	0.25	**0.00**	**0.00**
	-l	92.70	445.85	91.98	**0.00**	**0.00**
13		**0.00**	3.53	**0.00**	**0.00**	**0.00**
14		6.54	34.61	3.19	**0.00**	1.91
15	5	1.85	39.81	**0.00**	0.93	1.85
	6	0.80	41.60	1.60	**0.00**	1.60
	7	**0.00**	82.01	0.72	3.60	2.16
	8	**0.00**	76.22	3.05	5.49	4.88
	9	**0.00**	80.92	2.89	3.47	4.05
average		13.0803	169.7094	6.8876	8.9918	11.8427
maximum		165.6100	445.8500	91.9800	80.0500	84.9300

Table 3: distance from the best