

# High-Level Synthesis Scheduling and Allocation using Genetic Algorithms based on Constructive Topological Scheduling Techniques\*

M.J.M. Heijligers

J.A.G. Jess

Eindhoven University of Technology  
Design Automation Section, Room EH 7.23  
P.O. Box 513  
5600 MB Eindhoven  
the Netherlands  
tel: -31 40 473238, fax: -31 40 464527  
e-mail: M.J.M.Heijligers@ele.tue.nl

## Abstract

*In this article constructive scheduling methods combined with genetic algorithms are used to search for a suitable order to schedule the operations. The method is extended with an encoding capable of allocating supplementary resources during scheduling. This makes it very suitable in high-level synthesis strategies based on lower bound estimations techniques. Experiments and comparisons show high quality results and fast run times that outperform results produced by other heuristic scheduling methods*

## 1 Introduction

During high-level synthesis a behavioral description of a chip is translated into a digital network structure [McFa90]. The behavioral description consists of calculations (like additions, multiplications, logical operations etc.) and control structures (like conditionals, loops and procedure calls) which are used to transform input data into output data. The digital network structure consists of functional modules (adders, multipliers, ALUs, logical gates), storage (like register, register files, RAM) and interconnect (point-to-point connections, buses), summarized by the term resources, which implements the behavior specified. During high-level synthesis several interdependent problems need to be solved:

- Selection. Which kind of resources do we need?
- Allocation. How many resources do we need?
- Scheduling. At which cycle steps will resources be occupied by operations, values and transfer of data?
- Assignment. To which specific resources will operations / values / value-transfers be assigned?

The interdependent character of these problems follows from the fact that a schedule induces a resource allocation (because particular operations are executing simultaneously) and a completion time (the cycle in which the last operation finishes its execution).

<sup>0</sup>This article will be presented at the International Conference on Evolutionary Computing, Perth, Western Australia, 1995

A common scheduling problem in high-level synthesis is the so called time-constrained scheduling problem. Given a time constraint which denotes the cycle step before which each operation must have finished its execution, and a behavioral description it tries to find the minimal resource allocation. Solving time constrained scheduling efficiently is a non-trivial matter because of the NP-complete nature of this problem.

To restrict the search space of time constrained scheduling, lower bound estimation techniques as reported in [Timm93] can be used. Given a time constraint and a data flow graph this technique gives an accurate lower bound on the number of functional units which are needed to schedule the graph within the time constraint specified. In some cases this technique finds a lower bound for which no feasible schedule exists, i.e. the completion time of all possible schedules exceed the time constrained. Schedulers which are used in such an environment should be able to cope with such a behavior. When exact solution methods like IP scheduling ([Hwan91], [Gebo90]) are used, the danger exists that they will perform an exhaustive search because they cannot detect that a combination of constraints is infeasible, and large run times might result. Heuristic methods like list scheduling [Thom90], critical path scheduling [Park86] and force directed (list) scheduling [Verh92] are faster, but may produce unsatisfactory results because of their greedy characteristics.

In this article a new time constrained scheduler will be presented which is based on constructive scheduling methods combined with genetic search techniques [Mich92]. During constructive scheduling operations are scheduled one after another. Genetic search techniques are used to find a suitable order in which the operations have to be scheduled to obtain good schedule results. First the article will present that reducing the search space for the genetic algorithm, by partially enforcing the order and the cycle steps in which operations are scheduled, greatly speeds up and improves the quality of the results. Secondly the article will present an extension with an encoding which provides allocation of supplementary resources during scheduling to cope with lower bound estimations which are estimated too low. Finally, some well known benchmarks will be presented to show the quality of results and fast execution times of this method.

## 2 Scheduling and Genetic Algorithms

During high-level synthesis the behavioral description of a chip is often represented by a data-flow graph [Eijn92]. In a data-flow graph nodes represent operations and control structures, and edges model flow of data. In figure 1 an example of a data flow graph of a fast discrete cosine transform filter [Mall90] can be found.

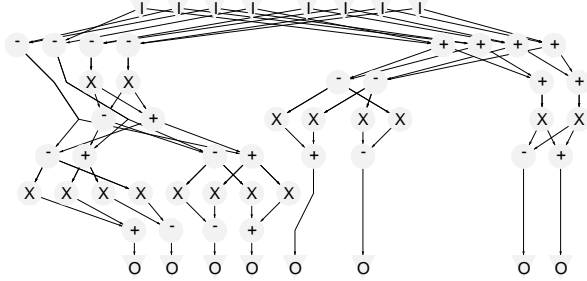


Figure 1: Fast Discrete Cosine Transform Filter

If there is flow of data from operation  $v_i$  to  $v_j$ , then  $v_i$  has to finish its execution (produce data) before  $v_j$  can start its execution (consume data), also called a dependency relation. When a data flow graph  $G = (V, E)$  and a time constraint  $T_{max}$  are given, a schedule range  $[asap(v), alap(v)]$  for each operation  $v \in V$  can be derived. If an operation is scheduled outside its schedule range it will either violate one or more dependency relations or the resulting schedule will exceed the time constraint. Let  $delay(v)$  denote the duration of the execution of an operation  $v$ , let  $pred(v)$  denote the set of predecessors of operation  $v$ , and let  $suc(v)$  denote the set of successors of operation  $v$ . The  $asap$  and  $alap$  values of  $v$  can be recursively defined as:

$$asap(v) = \max_{v_i \in pred(v)} (asap(v_i) + delay(v_i))$$

$$alap(v) = \min_{v_i \in suc(v)} (alap(v_i) - delay(v_i))$$

With  $asap(v) = 0$  if  $pred(v) = \emptyset$  and  $alap(v) = T_{max}$  if  $suc(v) = \emptyset$ .

One way to construct a schedule is by using a technique called constructive scheduling, in which operations are scheduled one by one within their feasible schedule range. Genetic techniques can be used to search for a suitable order of operations to obtain good quality schedules. Such a genetic approach generally works as follows. Initially a population with individuals is constructed, each containing a random permutation. Schedules are constructed by decoding permutations by applying a constructive scheduler. The genetic algorithm selects individuals for re-combination using stochastic sampling with replacement (also known as roulette

wheel selection). Using this strategy, fit individuals have higher probability to be selected than non-fit individuals. In [Star91] an overview of genetic sequencing operators is presented. Our own implementations show a clear advantage for uniform crossover, and an underlying theory is presented in [Mesm95]. The termination condition of the search usually consists of a specified number of runs, but when the resource allocation induced by the schedule found meets the lower bound estimation, an optimal solution is found, and the genetic search can be stopped (see subsection 3.3 for more details).

Most scheduling methods based on genetic algorithms don't regard feasible schedule ranges directly. Ignoring these constraints would lead to genetic search strategies that would evaluate infeasible schedules most of the time. In the next section several schedule constructors and their impact on the efficiency of Genetic Algorithms will be presented. This will finally lead to an efficient time constrained scheduler with fast execution times and good quality results.

## 3 A comparison of constructive scheduling methods

### 3.1 Absolute displacements

In [Wehn91] a scheduling method is presented based on genetic paradigms. This method does not use constraints, but searches for a trade-off between resource allocation and completion time by re-weighting a cost function. The method is based upon assigning a displacement  $d_a(v)$  to each operation  $v \in V$ . It then uses a modified asap schedule algorithm to construct a topological order schedule, in which the execution of each operation  $v$  selected for scheduling is deferred  $d_a(v)$  cycles. Resource constraints are used to defer operations even further when all resources are occupied in the current cycle considered for scheduling. The method uses simple crossover and mutation operators, and doesn't need 'complex' genetic sequencing operators.

Disadvantage of the method is that displacement of critical path operations has a large impact on the completion time of the schedule. In [Wehn91] special initialization routines are presented to construct a population containing schedules within their 'time constraint' by distributing displacements over critical paths. An improvement of the quality of the results is reported, however no attention has been paid to adapt re-combinators such that this property will be preserved during the run of the genetic algorithm. Our own experience using a time constrained scheduler derived from this method shows that only a few individuals are constructed that represent feasible schedules (i.e. within their time constraint, see table 1<sup>1</sup> for results). Results presented in subsection 3.4 show that the use of displacements complicates the search space for scheduling unnecessary.

### 3.2 Relative displacements

A new method has been developed that uses relative displacements  $d_r(v) \in [0, 1]$  for each operation  $v \in V$ . Operations are scheduled according to the order specified in a permutation

<sup>1</sup>x denotes that no feasible schedule could be found

II. A selected operation  $v \in V$  is scheduled in cycle step  $asap(v) + (alap(v) - asap(v) - delay(v)) * d_r(v)$ , and the feasible schedule ranges of other operations are updated if necessary. The cost of the resulting schedule can be determined by calculating the area of the resource allocation induced by the schedule. An individual contains both a permutation  $\Pi$  and a sequence of relative displacements. Uniform crossover, inversion and mutation are used to create new individuals.

The advantage of the method is that it always leads to feasible schedules with respect to time and precedence constraints. The method however produces disappointing results (see table 1). A possible explanation for the failure of the algorithms is the lack of problem specific knowledge inside the decoding of the permutation. For example, the method does not prevent 2 additions being scheduled simultaneously, even if both operations have large schedule ranges. Simple strategies to prevent from such a behavior are not known to us.

### 3.3 List scheduling

In [Heij95] a permutation  $\Pi$  of operations is used as a priority list for a list scheduler [Thom90]. A list scheduler is resource constrained, i.e. given a data flow graph and a resource constraint (e.g. number of functional units) it will try to find a schedule with the fastest completion time possible. To obtain a time constrained scheduler from a list scheduler, the method as depicted in figure 2 can be used.

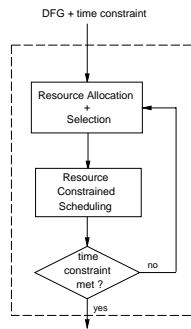


Figure 2: Resource Constrained towards Time Constrained

First a resource allocation must be performed to obtain a resource constraint. To be sure that not too many resources are allocated (which lead to non-optimal solutions), lower bound estimations as in [Timm93] can be used. After allocation, a list scheduler can be applied to construct a schedule. The completion time of this schedule can be used to calculate the fitness of the permutation and the resource allocation. A small penalty on the fitness is used to favor individuals representing schedules that are within the time constraint. If both the time constraint and resource constraint are satisfied (hence an optimal solution has been found) or the number of populations exceeds 100, the scheduler stops and returns the best schedule constructed so far.

In some cases lower bound estimation techniques find a lower bound for which no feasible schedule exists, i.e. the resource

allocation induces a completion time for which each resource constrained schedule exceeds the time constraint specified. Consequently the method must have the possibility to allocate extra hardware.



Figure 3: Encoding of supplementary resource allocation

In [Clui92] a genetic encoding of extra resource allocation is presented. The difference between the maximum and minimum number of resources needed is encoded in a string. A lower bound on the number of resources is obtained by using the technique of [Timm93], and an upper bound on the number of extra resources needed can be obtained by looking at the maximal parallelism for each resource type. This problem can be modelled as a min-flow max-cut problem on a comparability graph [Golu80], which can be solved in polynomial time. Each resource not part of the initial lower bound is accompanied by a binary variable which denotes whether a resource is available (1) or is not available (0) for scheduling (see figure 3). Standard cross and mutate re-combinators can be used to modify these classic bit-vector representations during the run of the genetic algorithm.

The advantage of using a genetic strategy for the allocation of extra resources is that no complex feedback paths are necessary for re-allocation (see figure 2), the results of which might heavily depend on the resource constrained scheduler used and the initial resource allocation taken, such as is the case in for instance [Kuma91].

Special attention has been paid to prevent constructing permutations which lead to identical schedules. In the example of figure 4 individual 0 and 1 lead to the same list schedule, hence exchanging operation  $v_1$  and  $v_3$  inside the permutation makes no sense.

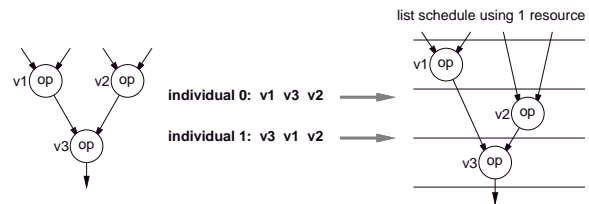


Figure 4: Permutations and lists schedules

Several observations assist in avoiding the exchange of operations leading to identical schedules:

- Only the exchange of operations whose schedule ranges have cycles in common will lead to priority lists with possibly different schedules.
- Only the exchange of nodes which don't have flow of data

in common (i.e. can be executed in parallel) can lead to priority lists with different schedules. This relation among nodes can be determined by taking the complement of the transitive closure of the data flow graph.

- Only the exchange of operations which have common resource types makes sense.

These observations have been incorporated inside the genetic recombinators as follows. Before scheduling starts for each node  $u$  a list of nodes  $L(u)$  is constructed that satisfies the observations mentioned before. During mutation first a node  $u$  from a permutation is selected randomly, then a node  $v \in L(u)$  is selected randomly, and after that  $u$  and  $v$  are exchanged inside the permutation. During inversion first a node  $u$  from a permutation is selected randomly, then the first node  $v \in L(u)$  in the permutation which comes after  $u$  inside the permutation is selected, and a position bigger or equal than the position of node  $v$  is selected. These new extensions greatly add to a more efficient investigation of the design space [Heij95] and results show that the method outperforms ordinary list scheduling (see table 1).

Disadvantage of the method is that it may exclude the optimal solution in some cases (see figure 5, in which list scheduling using one adder (1 cycle) and one multiplier (2 cycles) introduces one extra cycle independent of the priority list used).

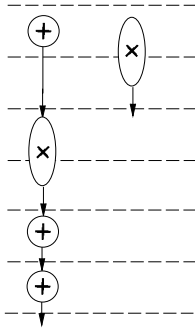


Figure 5: Greedy non-optimal list schedule

### 3.4 Topological sorted scheduling

Despite the good performance in most practical cases of the genetic schedule strategy using a list scheduler approach, it might never find an optimal solution in some cases as has been shown in figure 5. The method presented in subsection 3.2 using relative displacement does include the optimal solution in its search space, but results show a poor efficiency because the schedule constructor lacks problem specific knowledge.

A new scheduling approach has been constructed to overcome these disadvantages. Given a permutation  $\Pi$  it will build a schedule in a topological order to gradually construct schedules which prevent introduction of unnecessary parallelism compared to the method presented in subsection 3.2. The topological scheduling method is resource constrained in nature, and can be transformed

in a time constrained scheduler identical to the method presented in subsection 3.3.

Let  $\Pi$  be a permutation of operations. Let  $\Pi(i)$  denote the  $i^{th}$  operation from  $\Pi$ . Let  $\sigma(v)$  be the cycle an operation  $v \in V$  starts its execution. Topological sorted scheduling is performed by the following algorithm, called *construct\_schedule* :  $\Pi \mapsto \sigma$ :

```

for  $i := 0$  to  $|\Pi| - 1$  do
   $j := 0$ ;  $v := \Pi(j)$ ;
  while  $!(unscheduled(v) \wedge scheduledpreds(v))$  do
     $v := \Pi(j++)$ ;
   $\sigma(v) := asap(v)$ ;
   $\sigma(v) := satisfyResourceConstraint(v, \sigma(v))$ ;
  update schedule ranges;
  update resource usage;
   $cost := MAX(cost, \sigma(v) + delay(v))$ ;

```

The algorithm schedules each operation from permutation  $\Pi$  by repeatedly searching for the first operation  $v$  from  $\Pi$  which is un-scheduled and for which each predecessor is scheduled. The selected operation  $v$  is attempted to be scheduled in the as soon as possible cycle from its feasible range. When all resources are occupied at this cycle, the function *satisfyResourceConstraint*( $v, t$ ) defers operation  $v$  to the first succeeding cycle steps where a free resource is found. After an operation  $v$  is scheduled, the schedule ranges of all un-scheduled operations are updated, and the resource requirements due to scheduling  $v$  is administrated. The cost of the schedule is defined by the cycle step in which the last operation finishes its execution.

A proof will be given that there exists at least one permutation  $\Pi$  for which the topological sorted schedule constructor results in an optimal schedule.

Let  $\sigma^{-1}(t)$  denote the set of operations that are scheduled in cycle step  $t$ . Let *construct\_permutation* :  $\sigma \mapsto \Pi$  be given by the following algorithm:

```

 $i := 0$ ;
for  $t := t_{begin}$  to  $t_{end}$  do
  for  $v \in \sigma^{-1}(t)$  do
     $\Pi(i++) := v$ ;

```

**THEOREM:** There exists a permutation  $\Pi$  for which *construct\_schedule*( $\Pi$ ) returns an optimal schedule.

**PROOF:** Let  $\sigma_{opt}$  be an optimal (and hence feasible) schedule. Let  $\Pi = construct\_permutation(\sigma_{opt})$ . Then according to algorithm *construct\_permutation*,  $\Pi$  can be written as  $\Pi = \sigma_{opt}^{-1}(t_{begin}) \cdot \sigma_{opt}^{-1}(t_{begin}+1) \dots \sigma_{opt}^{-1}(t_{end}) = \Pi_1 \cdot \Pi_2 \dots \Pi_{t_{end}-t_{begin}+1}$ . For all  $v_i, v_j \in \Pi_i$  there are no precedence constraints or resource conflicts, because otherwise  $\sigma_{opt}$  would be an infeasible schedule.

Let  $\sigma = construct\_schedule(\Pi)$ . We first proof by induction that  $\forall t \in [t_{begin}, t_{end}] \forall v \in \sigma_{opt}^{-1}(t) \sigma(v) \leq t$ .

First, let  $\Pi = \sigma_{opt}^{-1}(t_{begin})$ . From  $\sigma_{opt}$  we know that there are no resource and precedence constraints between any tasks of  $\Pi$ , and hence *construct\_schedule* will schedule all tasks

in cycle step  $t_{begin}$ . Thus  $\forall v \in \sigma_{opt}^{-1}(t_{begin}) \sigma(v) = t_{begin} \Rightarrow \forall v \in \sigma_{opt}^{-1}(t_{begin}) \sigma(v) \leq t_{begin}$ .

Let the induction hypothesis be true for  $t \in [t_{begin}, t_{begin} + n]$ . Hence, if  $\Pi = \sigma^{-1}(t_{begin}) \cdot \sigma^{-1}(t_{begin+1}) \dots \sigma^{-1}(t_{begin} + n) = \Pi_1 \cdot \Pi_2 \dots \Pi_n$ , all tasks in  $\Pi$  can be scheduled by *construct\_schedule* in the range  $[t_{begin}, t_{begin} + n]$ , and  $\forall v \in \Pi \sigma(v) \leq t_{begin} + n$ .

Let  $\sigma^{-1}(t_{begin}) \cdot \sigma^{-1}(t_{begin+1}) \dots \sigma^{-1}(t_{begin} + n + 1) = \Pi \cdot \Pi_{n+1}$ . Because in the original schedule  $\sigma_{opt}$  all operations from  $\Pi_{n+1}$  could be scheduled without constraint violation in cycle step  $t_{begin} + n + 1$ , and from the induction hypothesis we know that no operations from  $\Pi$  are scheduled in cycle steps larger than  $t_{begin} + n$ , the operations from  $\Pi_{n+1}$  can be scheduled without constraint violation inside cycle step  $t_{begin} + n + 1$  or smaller. Hence  $\forall v \in \Pi \cdot \Pi_{n+1} \sigma(v) \leq t_{begin} + n + 1$ , which proves the induction hypothesis.

So if  $\Pi = \text{construct\_permutation}(\sigma_{opt})$ , and  $\sigma = \text{construct\_schedule}(\Pi)$ , then  $\text{cost}(\sigma) \leq \text{cost}(\sigma_{opt})$ . Because  $\sigma_{opt}$  is an optimal solution, we know by definition that  $\sigma$  is an optimal solution.

The main advantages of topological scheduling are (1) that it holds the optimal solution and (2) implies a significant design space reduction compared to the methods presented in subsections 3.1 and 3.2, because it doesn't have to deal with operation displacements. Instead, for each operation selected for scheduling it only has to consider the first cycle step in which the operation can be scheduled without constraint violation, without excluding the optimal solution from the search space. Observe that list scheduling is a subset of topological scheduling, in which the order of operations in permutation  $\Pi$  is restricted to non-decreasing asap values of the operations.

The topological scheduling method offers good quality results and fast run-times (see table 1). The method is very robust in the sense that it is hardly sensitive for changes in the genetic parameters like population size, distribution of crossing and mutation, the random seed, and the kind of data flow graphs supplied. The topological scheduling method has also been incorporated and tested inside a random search approach, in which optimal solutions are not found in almost any case tested. Topological sorted scheduling has also been incorporated and tested inside an exhaustive search using an efficient branch and bound, mainly based on the observations done in subsection 3.3 to avoid creation of permutations which would lead to identical schedules. Experiments showed that the exhaustive search needs excessive amounts of execution times, and had to be cancelled after weeks of running time.

## 4 Comparison

The schedulers proposed in this article have been implemented on a HP 9000/735 workstation using the NEAT synthesis tool-box [Heij94], and have been compared with well-known high-level synthesis time constrained scheduling methods, like improved force-directed scheduling [Verh92] and ordinary list scheduling [Thom90] using the schedule range length as priority function. The experiments show that genetic time constrained scheduler

based on list scheduling and topological scheduling offers better results and fast run times. One of the comparisons can be found in table 1<sup>2</sup>, which shows the resource allocations found for an often used high-level synthesis scheduling benchmark shown in figure 1. In the table the execution time of an adder is assumed to be 1 cycle, whereas the execution time of a multiplier is assumed to be 2 cycles. Typical run times of the methods can be found in the last row of table 1.

## 5 Conclusions

A genetic search strategy for scheduling which doesn't use schedule specific knowledge gives poor results. An efficient constructive scheduler based on topological sorting combined with genetic search techniques and incorporation of schedule specific knowledge like lower bound estimations, constraint satisfaction and the possibility of allocation of extra resources has resulted in a time constrained scheduler which offers high quality results and fast execution times. A prove has been given that the search space of the genetic scheduling algorithm contains the optimal schedule. Although the genetic search does not guarantee to find the optimal solution, the algorithm finds optimal results for all examples tested. Comparison with other high-level synthesis scheduling methods show that the new method offers better results and fast run times.

## References

- [Clui92] L.J.M. CLUITMANS. Using Genetic Algorithms for Scheduling Data Flow Graphs. Technical Report 92-E-266, Eindhoven University of Technology, 1992.
- [Eijn92] J.T.J. VAN EIJNDHOVEN AND L. STOK. A Data Flow Graph Exchange Standard. In *Proceedings of the European Conference on Design Automation*, pages 193–199, Brussels, March 1992. IEEE Computer Society Press.
- [Gebo90] C.H. GEBOTYS AND M.I. ELMASRY. A Global Optimization Approach for Architectural Synthesis. In *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pages 258–261, Santa Clara, November 1990. IEEE Computer Society Press.
- [Golu80] M.C. GOLUMBIC, editor. *Algorithmic graph theory and perfect graphs*. Academic Press, 1980.
- [Heij94] M.J.M. HEIJLIGERS, H.A. HILDERINK, A.H. TIMMER, AND J.A.G. JESS. NEAT: an Object Oriented High-Level Synthesis Interface. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1.233–1.236, London, June 1994. IEEE Computer Society Press.
- [Heij95] M.J.M. HEIJLIGERS, L.J.M. CLUITMANS, AND J.A.G. JESS. High-level Synthesis Scheduling and Allocation using Genetic Algorithms. In *submitted to Asia and South Pacific Design Automation Conference*, September 1995.
- [Hwan91] C.T. HWANG, Y.C. HSU, AND Y.L. LIN. A formal Approach to the Scheduling Problem in High Level Synthesis. *IEEE Transactions on Computer-Aided Design*, 10(4):464–475, April 1991.
- [Kuma91] A KUMAR, A. KUMAR, AND M. BALAKRISHNAN. A Novel Integrated Scheduling and Allocation Algorithm for Data Path Synthesis. *International Symposium on VLSI Design*, pages 212–218, January 1991.

<sup>2</sup>– means not applicable

Table 1: Results for fast discrete cosine transform

cycles	optimal		[Wehn91]		genetic relative		genetic list		genetic topsort		ordinary list		[Verh92]	
	mult	add	mult	add	mult	add	mult	add	mult	add	mult	add	mult	add
8	8	4	8	4	8	5	8	4	8	4	8	4	8	4
9	8	4	9	4	8	4	8	4	8	4	—	—	8	4
10	5	4	x	x	5	4	5	4	5	4	5	4	5	4
11	4	3	5	6	5	4	4	3	4	3	—	—	4	4
12	4	3	x	x	4	4	4	3	4	3	—	—	4	3
13	4	2	x	x	4	4	4	2	4	2	4	3	4	3
14	3	2	x	x	4	3	3	2	3	2	—	—	3	3
15	3	2	x	x	3	4	3	2	3	2	4	2	3	3
16	3	2	x	x	3	3	3	2	3	2	—	—	3	3
17	3	2	x	x	3	4	3	2	3	2	3	2	3	3
18	2	2	x	x	3	3	2	2	2	2	—	—	3	2
19	2	2	x	x	3	3	2	2	2	2	—	—	2	2
20	2	2	x	x	3	2	2	2	2	2	—	—	2	2
21	2	2	x	x	3	3	2	2	2	2	2	2	2	2
22	2	2	x	x	3	2	2	2	2	2	—	—	2	2
23	2	2	x	x	3	2	2	2	2	2	—	—	2	2
24	2	2	x	x	2	3	2	2	2	2	—	—	2	2
25	2	2	x	x	2	2	2	2	2	2	—	—	2	2
26	2	1	x	x	2	3	2	1	2	1	—	—	2	2
27	2	1	x	x	3	2	2	1	2	1	2	1	2	2
28	2	1	x	x	2	3	2	1	2	1	—	—	2	2
29	2	1	x	x	2	3	2	1	2	1	—	—	2	2
30	2	1	x	x	2	3	2	1	2	1	—	—	2	2
31	2	1	x	x	2	2	2	1	2	1	—	—	2	2
32	2	1	x	x	2	2	2	1	2	1	—	—	2	1
33	2	1	x	x	2	2	2	1	2	1	—	—	2	2
34	1	1	x	x	2	2	1	1	1	1	—	—	2	1
ex. time	—		20		30		1–20		1–20		0.7		5–300	

[Mall90] D.J. MALLON AND P.B. DENYER. A New Approach To Pipeline Optimisation. In *Proceedings of the European Conference on Design Automation*, pages 83–88, Glasgow, March 1990. IEEE Computer Society Press.

[McFa90] M.C. MCFARLAND, A.C. PARKER, AND R. CAMPOSANO. The High-Level Synthesis of Digital Systems. *Proceedings of the IEEE*, 78(2):301–318, February 1990.

[Mesm95] B. MESMAN. Genetic Algorithms for Scheduling Purposes. Master's thesis, Eindhoven University of Technology, May 1995.

[Mich92] Z. MICHALEWICZ. *Genetic Algorithms + Data Structures = Evolution Programs*. Artificial Intelligence. Springer Verlag, 1992.

[Park86] A.C. PARKER, J.T. PIZARRO, AND M. MLINAR. MAHA: A program for data path synthesis. In *Proceedings of the 23th ACM/IEEE Design Automation Conference*, pages 461–466, Las Vegas, June 1986. ACM and IEEE Computer Society.

[Star91] T. STARKWEATHER, S. MCDANIEL, K. MATHIAS, D. WHITLEY, AND C. WITHLEY. A Comparison of Genetic Sequencing Operators. In R.K. BELEW, L. BOOKER, AND J.D. SCHAFER, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 69–76, San Diego, July 1991. Morgan Kaufmann.

[Thom90] D.E. THOMAS, E.D. LAGNESE, R.A. WALKER, J.A. NESTOR, J.V. RAJAN, AND R.L. BLACKBURN. *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench*. Kluwer Academic Publisher, 1990.

[Timm93] A.H. TIMMER, M.J.M. HEIJLIGERS, L. STOK, AND J.A.G. JESS. Module Selection and Scheduling using Unrestricted Libraries. In *Proceedings of the European Conference on Design Automation with the European Event in ASIC Design*, pages 547–551, Paris, February 1993. IEEE Computer Society Press.

[Verh92] W.F.J. VERHAEGH, P.E.R. LIPPENS, E.H.L. AARTS, J.H.M. KORST, A. VAN DER WERF, AND J.L. VAN MEERBERGEN. Efficiency Improvements for Force-Directed Scheduling. In *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pages 286–291, Santa Clara, November 1992. IEEE Computer Society Press.

[Wehn91] N. WEHN, M. GLESNER, AND M. HELD. A Novel Scheduling and Allocation Approach for Datapath Synthesis based on Genetic Paradigms. In *IFIP Working Conference on Logic and Architecture Synthesis*, pages 47–56, Paris, 1991.