

On Symbolic Scheduling Independent Tasks with Restricted Execution Times

Daniel Sawitzki*

University of Dortmund, Computer Science 2
D-44221 Dortmund, Germany
daniel.sawitzki@cs.uni-dortmund.de

Abstract. Ordered Binary Decision Diagrams (OBDDs) are a data structure for Boolean functions which supports many useful operations. It finds applications in CAD, model checking, and symbolic graph algorithms. We present an application of OBDDs to the problem of scheduling N independent tasks with k different execution times on m identical parallel machines while minimizing the over-all finishing time. In fact, we consider the decision problem if there is a schedule with makespan D . Leung's dynamic programming algorithm solves this problem in time $\mathcal{O}(\log m \cdot N^{2(k-1)})$. In this paper, a symbolic version of Leung's algorithm is presented which uses OBDDs to represent the dynamic programming table T . This heuristical approach solves the scheduling problem by executing $\mathcal{O}(k \log m \log(mD))$ operations on OBDDs and is expected to use less time and space than Leung's algorithm if T is large but well-structured. The only known upper bound of $\mathcal{O}((m \cdot D)^{3k+2})$ on its resource usage is trivial. Therefore, we report on experimental studies in which the symbolic method was applied to random scheduling problem instances.

1 Introduction

The problem of nonpreemptively scheduling N independent tasks with integral execution times on m identical and parallel machines while minimizing the over-all finishing time (the *makespan*) is one of the most fundamental and well-studied problems of deterministic scheduling theory. It is known to be NP-hard in the strong sense [5]. In this paper, we consider the restricted case that the tasks have only a constant number k of different execution times. Moreover, we are interested in the decision problem if there is a schedule with makespan not larger than D . This restricted problem is simply referred to as *scheduling problem* throughout this paper.

Definition 1 (Scheduling Problem). A scheduling problem \mathcal{P} consists of k execution times $t_1, \dots, t_k \in \mathbb{N}$, corresponding demands $N_1, \dots, N_k \in \mathbb{N}$, a

* Supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Research Cluster "Algorithms on Large and Complex Networks" (1126).

number m of machines, and a makespan bound D . The over-all demand of \mathcal{P} is defined by $N := \sum_{1 \leq i \leq k} N_i$.

A schedule $\mathcal{S}: \{1, \dots, k\} \times \{1, \dots, m\} \rightarrow \mathbb{N}$ for a scheduling problem \mathcal{P} is called valid if $\sum_{j=1}^m \mathcal{S}(i, j) \geq N_i$ for every $i \in \{1, \dots, k\}$ and $\sum_{i=1}^k t_i \cdot \mathcal{S}(i, j) \leq D$ for every $j \in \{1, \dots, m\}$.

A scheduling algorithm has to decide if there is a valid schedule \mathcal{S} for \mathcal{P} .

Leung [13] presents a scheduling algorithm with time $\mathcal{O}(\log m \cdot N^{2(k-1)})$ and space $\mathcal{O}(\log m \cdot N^{(k-1)})$. Following a dynamic programming approach, it computes a table T of $\mathcal{O}(\log m \cdot N^{k-1})$ partial solution values. The author considers the algorithm as polynomial for constant k because the input size of general scheduling problems is $\Omega(N)$. However, due to the restriction to k different execution times, the input can be represented by $2k+2$ numbers of length $\log(mD)$.

The idea behind the symbolic scheduling algorithm presented in this paper is to use *Ordered Binary Decision Diagrams (OBDDs)* [3, 4, 22] to represent the dynamic programming table T . OBDDs are a data structure for Boolean functions offering efficient functional operations, which is well-established in many areas like CAD, model checking [8, 14], and symbolic graph algorithms [9, 18, 17, 20, 23]. It is known to be a compact representation for structured and regular functions and allows to compute many table entries in parallel by few operations applied to the corresponding OBDDs. On the one hand, we expect this approach to require essentially less space than Leung's method; on the other hand, this implies also less runtime, because the efficiency of OBDD operations depends on the size of their operands.

In order to analyze the behavior of symbolic OBDD-based heuristics, we have to analyze the OBDD size of all Boolean functions occurring during their execution. This is known to be a difficult task in general and has been done only in a few pioneer works so far [18, 20, 21, 23]. So in most papers the usability of symbolic algorithms is just proved by experiments on benchmark inputs from special application areas [9, 10, 12, 15, 24]. In other works considering more general graph problems, mostly the number of OBDD operations (often referred to as "symbolic steps") is bounded as a hint on the actual runtime [2, 6, 7, 16].

To evaluate the usefulness of the presented scheduling method, it has been implemented and applied to random input instances for $k = 3$ due to three popular distributions of execution times. On these instances, the symbolic algorithm was observed to beat Leung's scheduling algorithm w. r. t. time and space if the product $P := \prod_{j=1}^{k-1} N_j$ of task quantities is sufficiently large.

The paper is organized as follows: Section 2 introduces OBDDs and the operations offered by them. Then, Sect. 3 gives some preliminaries on symbolic algorithms and their notation. After a brief description of Leung's algorithm in Sect. 4, we present the symbolic scheduling method in Sect. 5. The experiments' setting and results are documented in Sects. 6 and 7. Finally, Sect. 8 gives conclusions on the work.

2 Ordered Binary Decision Diagrams (OBDDs)

We denote the class of Boolean functions $f: \{0, 1\}^n \rightarrow \{0, 1\}$ by B_n . The i th character of a binary number $x \in \{0, 1\}^n$ is denoted by x_i and $|x| := \sum_{i=0}^{n-1} x_i 2^i$ identifies its value.

A Boolean function $f \in B_n$ defined on variables x_0, \dots, x_{n-1} can be represented by an *Ordered Binary Decision Diagram (OBDD)* [3, 4]. An OBDD \mathcal{G} is a directed acyclic graph consisting of *internal nodes* and *sink nodes*. Each internal node is labeled with a Boolean variable x_i , while each sink node is labeled with a Boolean constant. Each internal node is left by two edges one labeled by 0 and the other by 1. A *function pointer* p marks a special node that represents f . Moreover, a permutation $\pi \in \Sigma_n$ called *variable order* must be respected by the internal nodes' labels on every path from p to a sink. For a given variable assignment $a \in \{0, 1\}^n$, we compute the function value $f(a)$ by traversing \mathcal{G} from p to a sink labeled with $f(a)$ while leaving a node x_i via its a_i -edge.

An OBDD with variable order π is called π -OBDD. The minimal-size π -OBDD \mathcal{G} for a function $f \in B_n$ is known to be canonical. Its size $\text{size}(\mathcal{G})$ is measured by the number of its nodes and will be denoted by $\pi\mathcal{G}[f]$. We adopt the usual assumption that all OBDDs occurring in symbolic algorithms have minimal size, since all essential OBDD operations produce minimized diagrams. On the other hand, finding an optimal variable order leading to the minimum size OBDD for a given function is known to be NP-hard. There is an upper bound of $(2 + o(1))2^n/n$ for the OBDD size of every $f \in B_n$.

The satisfiability of f can be decided in time $\mathcal{O}(1)$. The negation \bar{f} as well as the replacement of a function variable x_i by a constant a_i (i. e., $f_{|x_i=a_i}$) is obtained in time $\mathcal{O}(\text{size}(\pi\mathcal{G}[f]))$ without enlarging the OBDD. Whether two functions f and g are equivalent (i. e., $f = g$) can be decided in time $\mathcal{O}(\text{size}(\pi\mathcal{G}[f]) + \text{size}(\pi\mathcal{G}[g]))$. These operations are called *cheap*. Further essential operations are the *binary synthesis* $f \otimes g$ for $f, g \in B_n$, $\otimes \in B_2$ (e. g., “ \wedge ” or “ \vee ”), and the *quantification* $(\mathcal{Q}x_i)f$ for a quantifier $\mathcal{Q} \in \{\exists, \forall\}$. In general, the result $\pi\mathcal{G}[f \otimes g]$ has size $\mathcal{O}(\text{size}(\pi\mathcal{G}[f]) \cdot \text{size}(\pi\mathcal{G}[g]))$, which is also the general runtime of this operation. The computation of $\pi\mathcal{G}[(\mathcal{Q}x_i)f]$ can be realized by two cheap operations and one binary synthesis in time and space $\mathcal{O}(\text{size}^2(\pi\mathcal{G}[f]))$.

The book of Wegener [22] gives a comprehensive survey on different types of binary decision diagrams.

3 Preliminaries on Symbolic Algorithms

The functions used for symbolic representations are typically defined on a number of m subsets of Boolean variables, each having a certain interpretation within the algorithm. We assume w. l. o. g. that all arguments consist of the same number of n Boolean variables. If there is no confusion, both a function $f \in B_{mn}$ defined on $x^{(1)}, \dots, x^{(m)} \in \{0, 1\}^n$ as well as its OBDD representation $\pi\mathcal{G}[f]$

will be denoted by f in this paper. Quantifications $(\mathcal{Q}x_0^{(i)}, \dots, x_{n-1}^{(i)})$ over all n variables of argument i will be denoted by $(\mathcal{Q}x^{(i)})$.

Argument reordering. Assume that each of the m function arguments $x^{(1)}, \dots, x^{(m)} \in \{0, 1\}^n$ has its own variable order $\tau_i \in \Sigma_n$. The global order $\pi \in \Sigma_{mn}$ is called m -interleaved if it respects each τ_i while reading variables $x_j^{(i)}$ with same bit index j en bloc, that is, $\pi := (x_{\tau_1(0)}^{(1)}, x_{\tau_2(0)}^{(2)}, \dots, x_{\tau_m(0)}^{(m)}, x_{\tau_1(1)}^{(1)}, \dots, x_{\tau_m(n-1)}^{(m)})$.

Let $\rho \in \Sigma_m$ and $f \in B_{mn}$ be defined on variables $x^{(1)}, \dots, x^{(m)} \in \{0, 1\}^n$. A function $g \in B_{mn}$ is called the *argument reordering* of f w.r.t. ρ if $g(x^{(1)}, \dots, x^{(m)}) = f(x^{(\rho(1))}, \dots, x^{(\rho(m))})$. Computing argument reorderings is an important operation of symbolic algorithms and is possible in linear time and space $\mathcal{O}(n)$ if an m -interleaved variable order is used and m is constant (see [19]).

Multivariate threshold and modulo functions. The symbolic scheduling algorithm contains comparisons of weighted sums with threshold values like $f(x, y, z) := (a \cdot |x| + b \cdot |y| \geq T)$, $a, b, T \in \mathbb{Z}$, which can be realized by *multivariate threshold functions*.

Definition 2 (Woelfel [23]). Let $f \in B_{mn}$ be defined on variables $x^{(1)}, \dots, x^{(m)} \in \{0, 1\}^n$. Then, f is called m -variate threshold function iff there are $W \in \mathbb{N}$, $T \in \mathbb{Z}$, and $w_1, \dots, w_m \in \{-W, \dots, W\}$ such that

$$f(x^{(1)}, \dots, x^{(m)}) = \left(\sum_{i=1}^m w_i \cdot |x^{(i)}| \geq T \right) .$$

Clearly, the relations $>$, \leq , $<$, and $=$ can be composed of multivariate threshold functions. For constant W and m , such comparisons have π -OBDDs of size $\mathcal{O}(n)$ for an m -interleaved variable order π with increasing bit significance (i.e., $\tau_i = \text{id}$) [23]. These OBDDs can be computed efficiently in linear time.

Moreover, the symbolic scheduling algorithm makes use of *multivariate modulo functions*.

Definition 3 (Woelfel [23]). Let $f \in B_{mn}$ be defined on variables $x^{(1)}, \dots, x^{(m)} \in \{0, 1\}^n$. Then, f is called m -variate modulo function iff there are $M \in \mathbb{N}$, $T \in \mathbb{Z}$, and $w_1, \dots, w_m \in \mathbb{Z}$ such that

$$f(x^{(1)}, \dots, x^{(m)}) = \left(\sum_{i=1}^m w_i \cdot |x^{(i)}| \bmod M = T \right) .$$

For constant M and m , m -variate modulo functions have π -OBDDs of size $\mathcal{O}(n)$ using an m -interleaved variable order π with increasing bit significance (i.e., $\tau_i = \text{id}$) [23]. These OBDDs can be computed efficiently in linear time.

We conclude that all essential functional operations are realized efficiently by OBDDs w.r.t. the corresponding OBDD size if an interleaved variable order is

used. Therefore, this property is assumed in the following. This is also crucial for threshold and modulo functions to have compact OBDDs, which will be the building blocks of all Boolean functions computed by the symbolic scheduling algorithm.

4 The Scheduling Algorithm of Leung

Let \mathcal{P} be a scheduling problem according to Def. 1. and assume that $\log_2(m) \in \mathbb{N}$. Leung’s algorithm [13] computes a k -dimensional table T with entries $T(\ell, i_1, \dots, i_{k-1})$ for $\ell = 0, \dots, \log_2 m$, $i_j = 0, \dots, N_j$, and $j = 1, \dots, k - 1$. Such an entry contains the maximum number I of tasks of type k that can be scheduled onto 2^ℓ machines together with i_j tasks of type j for all types $j = 1, \dots, k - 1$.

We define upper bounds $B_j := \min\{N_j, \lfloor D/t_j \rfloor\}$, $j = 1, \dots, k$, for the maximum number of tasks of type j that can be scheduled onto one machine. Let us consider the case $\ell = 0$: If $0 \leq i_j \leq B_j$ for $j = 1, \dots, k - 1$ and $D \geq \sum_{j=1}^{k-1} t_j \cdot i_j$, it is $I = \lfloor (D - \sum_{j=1}^{k-1} t_j \cdot i_j) / t_k \rfloor$; else, we define $I := -1$.

Having computed all $\prod_{j=1}^{k-1} (N_j + 1)$ entries $T(\ell, i_1, \dots, i_{k-1})$ for some machine count 2^ℓ , the entries for $2^{\ell+1}$ machines are obtained by

$$\begin{aligned}
 T(\ell + 1, i_1, \dots, i_{k-1}) := & \max \{ -1, T(\ell, i'_1, \dots, i'_{k-1}) + T(\ell, i''_1, \dots, i''_{k-1}) \\
 & | \forall j \in \{1, \dots, k - 1\}: i'_j, i''_j \in \{0, \dots, N_j\}, i_j = i'_j + i''_j, \\
 & T(\ell, i'_1, \dots, i'_{k-1}) \neq -1 \neq T(\ell, i''_1, \dots, i''_{k-1}) \} . \quad (1)
 \end{aligned}$$

This procedure can be easily modified to cope with values m that are not powers of 2. Finally, there is a valid schedule for \mathcal{P} if and only if $T(\log_2 m, N_1, \dots, N_{k-1}) \geq N_k$. Altogether, $\mathcal{O}(\log m \cdot \prod_{j=1}^{k-1} (N_j + 1)) = \mathcal{O}(\log m \cdot N^{k-1})$ table entries are computed, each one as maximum over $\mathcal{O}(\prod_{j=1}^{k-1} (N_j + 1)) = \mathcal{O}(N^{k-1})$ vectors (i'_1, \dots, i'_{k-1}) implying the runtime complexity $\mathcal{O}(\log m \cdot N^{2(k-1)})$.

The minimal makespan can be found by a binary search using $\mathcal{O}(\log \max \{t_1, \dots, t_k\})$ executions of Leung’s algorithm (see [13]). By storing the optimal partition vector (i'_1, \dots, i'_{k-1}) for each table entry, the algorithm can easily be extended to compute an optimal schedule if one exists.

5 The Symbolic Scheduling Algorithm

We again assume that $\log_2(m) \in \mathbb{N}$. Moreover, it is reasonable to require $t_i \leq D$ and $N_i \leq mD$ for $i = 1, \dots, k$. Then, $\lceil \log_2(mD + 1) \rceil =: n$ Boolean variables suffice to represent the number arguments of all Boolean functions occurring during the algorithm.

The symbolic scheduling algorithm works with *characteristic* Boolean functions $\chi_{T,\ell} \in B_{kn}$ of Leung’s dynamic programming table T defined by

$$\chi_{T,\ell}(x^{(1)}, \dots, x^{(k)}) = 1 \Leftrightarrow T(\ell, |x^{(1)}|, \dots, |x^{(k-1)}|) = |x^{(k)}|$$

for $\ell = 0, \dots, \log_2 m$ and vectors $x^{(j)} \in \{0, 1\}^n$, $j = 1, \dots, k$. The binary value of $x^{(j)}$ corresponds to the number i_j of tasks in Sect. 4. Because $|x^{(k)}|$ is non-negative, $\chi_{T,\ell}$ is false for table entries -1 .

In order to compute the initial function $\chi_{T,0}$, we express the conditions for $\ell = 0$ stated in Sect. 4 in terms of Boolean equations using multivariate threshold and modulo functions as building blocks. At first, we state a function g for the condition $|x^{(k)}| = \lfloor (D - \sum_{j=1}^{k-1} t_j \cdot |x^{(j)}|) / t_k \rfloor$, which is equivalent to

$$D - \sum_{j=1}^{k-1} t_j \cdot |x^{(j)}| = |x^{(k)}| \cdot t_k + \left(D - \sum_{j=1}^{k-1} t_j \cdot |x^{(j)}| \right) \bmod t_k .$$

This leads to the following symbolic formulation for g which can be computed by subsequent applications of OBDD operations starting with multivariate threshold and modulo functions. It uses two vectors $y, z \in \{0, 1\}^n$ of intermediate helping variables.

$$g(x^{(1)}, \dots, x^{(k)}) := (\exists y, z) \left[\left(|y| = D - \sum_{j=1}^{k-1} t_j \cdot |x^{(j)}| \right) \wedge (|z| < t_k) \wedge (|y| - |z| \bmod t_k = 0) \wedge (|y| = |x^{(k)}| \cdot t_k + |z|) \right]$$

Altogether, the initial function $\chi_{T,0}$ is obtained by

$$\chi_{T,0}(x^{(1)}, \dots, x^{(k)}) := \bigwedge_{j=1}^{k-1} (|x^{(j)}| \leq B_j) \wedge \left(D \geq \sum_{j=1}^{k-1} t_j \cdot |x^{(j)}| \right) \wedge g(x^{(1)}, \dots, x^{(k)}) .$$

At next, the iterative step (1) is realized in terms of OBDD operations using $2k + 1$ vectors $y, u^{(1)}, v^{(1)}, \dots, u^{(k)}, v^{(k)} \in \{0, 1\}^n$ of intermediate helping variables. Assume that $\chi_{T,\ell}$ has already been computed for some $\ell \in \{0, \dots, \log_2 m - 1\}$. We define $h_{\ell+1} \in B_{k,n}$ as

$$h_{\ell+1}(x^{(1)}, \dots, x^{(k)}) := \left(\exists u^{(1)}, v^{(1)}, \dots, u^{(k-1)}, v^{(k-1)} \right) \left[\bigwedge_{j=1}^{k-1} \left(|x^{(j)}| = |u^{(j)}| + |v^{(j)}| \right) \wedge \chi_{T,\ell}(u^{(1)}, \dots, u^{(k)}) \wedge \chi_{T,\ell}(v^{(1)}, \dots, v^{(k)}) \right].$$

That is, $h_{\ell+1}$ represents load vectors $(|x^{(1)}|, \dots, |x^{(k)}|)$ that can be partitioned into loads $(|u^{(1)}|, \dots, |u^{(k)}|)$ and $(|v^{(1)}|, \dots, |v^{(k)}|)$ each fitting onto 2^ℓ machines while respecting makespan bound D .

Finally, we have to guarantee the maximality of the number of type- k -tasks:

$$\chi_{T,\ell+1}(x^{(1)}, \dots, x^{(k)}) := h_{\ell+1}(x^{(1)}, \dots, x^{(k)}) \wedge \overline{(\exists y) [(|y| > |x^{(k)}|) \wedge h_{\ell+1}(x^{(1)}, \dots, x^{(k-1)}, y)]}.$$

That is, there is no number $|y|$ of type- k -tasks greater than $|x^{(k)}|$ that can also be distributed onto $2^{\ell+1}$ machines according to $h_{\ell+1}$.

Having computed $\chi_{T,\log_2 m}$ this way, we replace each variable vector $x^{(j)}$ by the binary representation of N_j for $j = 1, \dots, k-1$. Then, the unique satisfying assignment of the remaining variables $x^{(k)}$ correspond to $T(\log_2 m, N_1, \dots, N_{k-1})$ which is compared to N_k . The correctness follows from the correctness of Leung's algorithm. We have solved scheduling problem \mathcal{P} following Leung's approach by using a symbolic OBDD representation for the dynamic programming table T .

Similar to Leung's algorithm, the symbolic scheduling methods can be easily modified to handle arbitrary numbers m of machines as well as to compute concrete schedule \mathcal{S} .

Theorem 1. *The symbolic scheduling algorithm solves a scheduling problem \mathcal{P} with task execution times t_1, \dots, t_k , task demands N_1, \dots, N_k , machine count m , and makespan bound D by executing $\mathcal{O}(k \log m \log(mD))$ operations on OBDDs defined on $(3k+2)n$ Boolean variables with $n := \lceil \log_2(mD+1) \rceil$.*

Proof. We compute $\log_2 m + 1$ Boolean functions $\chi_{T,\ell}$ with $\ell = 0, \dots, \log_2 m$. Each function occurring during the algorithm is defined on no more than $(3k+2)n$ variables and computed by a constant number of binary syntheses and quantifications over variable vectors of length n . Altogether, each $\chi_{T,\ell}$ takes $\mathcal{O}(k \log(mD))$ OBDD operations. \square

The upper bound of $(2 + o(1))2^n/n$ for the OBDD size of every $f \in B_n$ implies a maximum OBDD size of $\mathcal{O}((mD)^{3k+2})$. Hence, each OBDD operation takes time $\mathcal{O}((mD)^{6k+4})$ in the worst case. After having computed $\chi_{T,\ell+1}$ we may discard $\chi_{T,\ell}$. Therefore, $\mathcal{O}((mD)^{3k+2})$ is also an upper bound on the overall space usage. Of course, these theoretical bounds cannot compete with the complexity of Leung's algorithm.

Nevertheless, we hope that heuristical methods like the symbolic scheduling algorithm perform much better than in the worst case when applied on practical

problem instances or in the average case. Then, they are expected to beat known algorithms with better worst-case behavior. Hence, we applied the presented algorithm to randomly generated instances hoping that structures and regularities of table T lead to compact OBDDs for the functions $\chi_{T,\ell}$ and, therefore, to an efficient over-all time and space usage.

6 Experimental Setting

The symbolic scheduling algorithm was implemented¹ in C++ using the gcc 2.95.3 and the OBDD package CUDD 2.3.1 by Fabio Somenzi.² Initially, an interleaved variable order with increasing bit significance is used for the Boolean variables of each function argument. After quantification operations, the actual variable order π is heuristically optimized by permuting three adjacent variables while keeping π interleaved. This is iterated until a local optimum is reached (see [11]).

The scheduling problem instances \mathcal{P} generated for the experiments have a load sum $L := \sum_{j=1}^k t_j \cdot N_j$ with mean $E[L] = M := (mD)/1.2$. That is, the effective capacity mD is 20% larger than the expected load. This is achieved by first drawing a uniformly distributed fraction F_j of M such that $\sum_{j=1}^k F_j = M$ for each task type $j = 1, \dots, k$. Then, the number N_j of tasks of each type j is drawn uniformly due to a mean parameter $E[N_j]$. Finally, the execution times t_j are drawn due to the uniform, exponential, or Erlang distribution (shape parameter 2) with mean $E[t_j] := F_j/N_j$, which are common distributions in modeling synthetic scheduling instances (see, e. g. [1]).

$$E[L] = \sum_{j=1}^k E[t_j \cdot N_j] = \sum_{j=1}^k E[t_j] \cdot N_j = \sum_{j=1}^k F_j = M$$

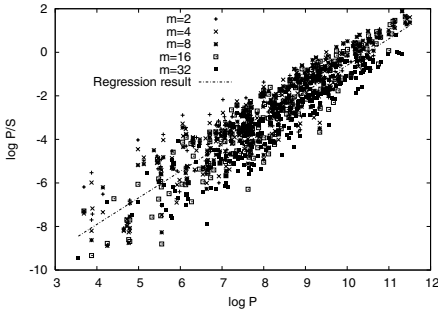
This random procedure has parameters m , D , and $E[N_j]$ for $j = 1, \dots, k$. The values t_j and N_j are rounded randomly to integers. Moreover, only those instances are accepted that fulfill the conditions $t_j \leq D$, $N_j \leq mD$, and $t_j \neq t_{j'}$ for $1 \leq j < j' \leq k$.

The experiments consist of three series with $mD = 800, 1600, 3200$ and $k = 3$. Within each series, m was chosen to be 2, 4, 8, 16, and 32. For each setting, 20 instances have been generated due to the three execution time distribution mentioned above. Moreover, 10 different values of $E[N_j]$ between $(mD)/6$ to $(mD)/3$ have been used per series ($E[N_1] = \dots = E[N_k]$).

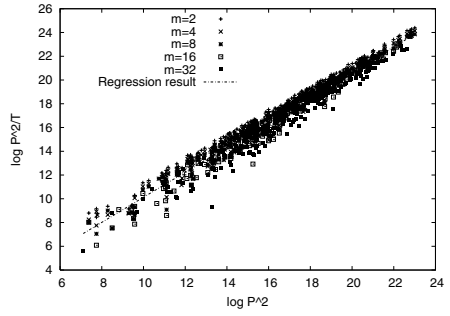
The experiments took place on a PC with Pentium 4 3GHz processor and 1 GB of main memory running Linux 2.4.21. The runtime has been measured by seconds of process time, while the space usage is given as the maximum number

¹ Implementation and experimental data are available at <http://thefigaro.sourceforge.net/>.

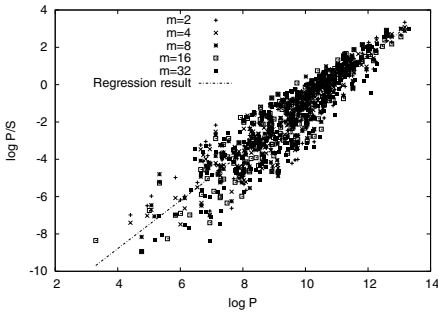
² CUDD is available at <http://vlsi.colorado.edu/>.



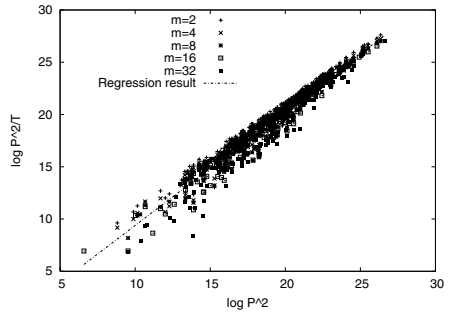
(a) Space comparison for $mD = 800$



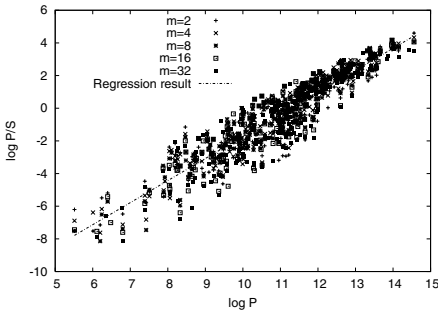
(b) Time comparison for $mD = 800$



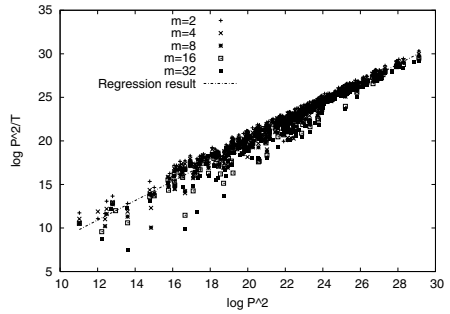
(c) Space comparison for $mD = 1600$



(d) Time comparison for $mD = 1600$



(e) Space comparison for $mD = 3200$



(f) Time comparison for $mD = 3200$

Fig. 1. Experimental results on random instances with exponentially distributed execution times. P denotes $\prod_{j=1}^{k-1} N_j$, S denotes the symbolic algorithm's space, T denotes the symbolic algorithm's time

of OBDD nodes present at any time during an algorithm execution. The latter is of same magnitude as the over-all space usage and independent of the used computer system.

7 Experimental Results

In order to compare Leung’s algorithm to the symbolic approach, we have to take a closer look at the resources used by the former one. Because we exclusively consider values m with $\log_2(m) \in \mathbb{N}$, each subtable $T(\ell, \dots)$ due to some machine count can be discarded after having computed the subtable for $\ell + 1$ machines. Moreover, we assume that $N_k = \min\{N_1, \dots, N_k\}$. Then, Leung’s algorithm needs space $\Omega(\prod_{j=1}^{k-1} N_j)$ and time $\Omega(\prod_{j=1}^{k-1} N_j^2)$.

We consider the experimental results for exponentially distributed execution times: Figures 1(a), 1(c), and 1(e) show plots of the ratios P/S for $P := \prod_{j=1}^{k-1} N_j$ and the symbolic algorithm’s space usage S for growing P using logarithmic scales. Analogue, Figs. 1(b), 1(d), and 1(f) show plots of the ratios P^2/T and the symbolic algorithm’s time usage T for growing P^2 . We observe the symbolic method to beat Leung’s algorithm w. r. t. both time and space for inputs with high task demand products P .

Concretely, the presented plots for exponentially distributed execution times as well as the omitted plots for the other two distributions hint to a linear dependence of $\log(P/S)$ of $\log P$ resp. $\log(P^2/T)$ of $\log P^2$. Therefore, a least squares method has been used to fit parameters a_1 and b_1 for $\log(P/S) = a_1 \cdot \log P + b_1$ resp. a_2 and b_2 for $\log(P^2/T) = a_2 \cdot \log P^2 + b_2$. Table 1 shows the fitting results for all three values of mD and the three considered distributions. The gradients’ a_1 and a_2 asymptotic standard errors never exceed 1.4%.

Table 1. Fits of a_1 and b_1 for P/S (Tab. 2(a)) resp. a_2 and b_2 for P^2/T (Tab. 2(b))

Distribution / mD	800	1600	3200
Uniform	1.28270 / -13.21930	1.32832 / -14.21380	1.36477 / -15.22760
Exponential	1.21732 / -12.77160	1.28701 / -13.93220	1.34987 / -15.22830
Erlang	1.26583 / -13.11690	1.37385 / -14.63660	1.44698 / -16.14440

(a) Fits for the symbolic algorithm’s space usage

Distribution / mD	800	1600	3200
Uniform	1.06093 / -0.47254	1.08898 / -1.35841	1.12353 / -2.51579
Exponential	1.05560 / -0.43123	1.09418 / -1.55419	1.11825 / -2.50760
Erlang	1.06234 / -0.52043	1.11002 / -1.82228	1.15972 / -3.31591

(b) Fits for the symbolic algorithms time usage

The Erlang distribution seems to result in slightly higher absolute values a and $|b|$.

The hypothesized linear dependencies imply $P/S = P^{a_1} \cdot 2^{b_1} \Leftrightarrow S = P^{1-a_1} \cdot 2^{-b_1}$ resp. $T = P^{2(1-a_2)} \cdot 2^{-b_2}$. Due to $1 < a_1, a_2 < 1.5$ (see Tab. 1), we conclude the symbolic scheduling algorithm to have space usage $2^{-b_1} / \sqrt[c_1]{P}$ for $c_1 = 1/(a_1 - 1) > 2$ resp. time usage $2^{-b_2} / \sqrt[c_2]{P}$ for $c_2 = 1/(2a_2 - 2) > 1$.

That is, the a - and b -parameters seem to depend only on mD and k while being independent of m . For fixed mD and k , the OBDD sizes shrink proportional to $\sqrt[c]{P}$ leading to essentially smaller time and space than Leung's algorithm. Although only experiments with $k = 3$ are documented, these results have been also observed for higher values k .

8 Conclusions

We presented a symbolic algorithm for the decision problem of scheduling independent tasks with restricted execution times. It solves the problem by performing $\mathcal{O}(k \log m \log(mD))$ OBDD operations, while its final runtime and space usage depends on the size of the OBDDs it generates. Therefore, it was applied to random scheduling instances whose execution times were generated due to the uniform, exponential, and Erlang distribution. On these instances, the symbolic algorithm was observed to beat Leung's scheduling algorithm w. r. t. time and space if the product $P := \prod_{j=1}^k N_j$ is sufficiently large. For fixed mD and k , the symbolic time and space usage is observed to grow as $\Theta\left(1/\sqrt[c]{P}\right)$ for some constant $c > 1$ depending on mD and the measured quantity.

Hence, we consider the application of OBDDs to Leung's scheduling method as a useful way to compress its dynamic programming table, which succeeds in savings of runtime and space on inputs with large demand N . Future research could address experiments on real world instances as well as several heuristics like different strategies for OBDD variable reordering.

Acknowledgments. Thanks to Detlef Sieling, Ingo Wegener, and Berthold Vöcking for fruitful discussions.

References

- [1] S. Albers and B. Schröder. An experimental study of online scheduling algorithms. *Journal of Experimental Algorithms*, 7:3, 2002.
- [2] R. Bloem, H.N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. In *Formal Methods in Computer-Aided Design 2000*, volume 1954 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2000.
- [3] R.E. Bryant. Symbolic manipulation of Boolean functions using a graphical representation. In *Design Automation Conference 1985*, pages 688–694. ACM Press, 1985.

- [4] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.
- [5] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
- [6] R. Gentilini, C. Piazza, and A. Policriti. Computing strongly connected components in a linear number of symbolic steps. In *Symposium on Discrete Algorithms 2003*, pages 573–582. ACM Press, 2003.
- [7] R. Gentilini and A. Policriti. Biconnectivity on symbolically represented graphs: A linear solution. In *International Symposium on Algorithms and Computation 2003*, volume 2906 of *Lecture Notes in Computer Science*, pages 554–564. Springer, 2003.
- [8] G.D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, Boston, 1996.
- [9] G.D. Hachtel and F. Somenzi. A symbolic algorithm for maximum flow in 0–1 networks. *Formal Methods in System Design*, 10:207–219, 1997.
- [10] R. Hojati, H. Touati, R.P. Kurshan, and R.K. Brayton. Efficient ω -regular language containment. In *Computer-Aided Verification 1993*, volume 663 of *Lecture Notes in Computer Science*, pages 396–409. Springer, 1993.
- [11] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchanges of variables. In *International Conference on Computer Aided Design 1991*, pages 472–475. IEEE Press, 1991.
- [12] H. Jin, A. Kuehlmann, and F. Somenzi. Fine-grain conjunction scheduling for symbolic reachability analysis. In *Tools and Algorithms for the Construction and Analysis of Systems 2002*, volume 2280 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 2002.
- [13] J. Y.-T. Leung. On scheduling independent tasks with restricted execution times. *Operations Research*, 30(1):163–171, 1982.
- [14] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, 1994.
- [15] I. Moon, J.H. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: The question in image computation. In *Design Automation Conference 2000*, pages 23–28. ACM Press, 2000.
- [16] K. Ravi, R. Bloem, and F. Somenzi. A comparative study of symbolic algorithms for the computation of fair cycles. In *Formal Methods in Computer-Aided Design 2000*, volume 1954 of *Lecture Notes in Computer Science*, pages 143–160. Springer, 2000.
- [17] D. Sawitzki. Experimental studies of symbolic shortest-path algorithms. In *Experimental and Efficient Algorithms 2004*, volume 3059 of *Lecture Notes in Computer Science*, pages 482–497. Springer, 2004.
- [18] D. Sawitzki. Implicit flow maximization by iterative squaring. In *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932 of *Lecture Notes in Computer Science*, pages 301–313. Springer, 2004.
- [19] D. Sawitzki. On graphs with characteristic bounded-width functions. Technical report, Universität Dortmund, 2004.
- [20] D. Sawitzki. A symbolic approach to the all-pairs shortest-paths problem. In *Graph-Theoretic Concepts in Computer Science 2004*, volume 3353 of *Lecture Notes in Computer Science*, pages 154–167. Springer, 2004.
- [21] D. Sawitzki. Lower bounds on the OBDD size of graphs of some popular functions. In *SOFSEM 2005: Theory and Practice of Computer Science*, volume 3381 of *Lecture Notes in Computer Science*, pages 298–309. Springer, 2005.

- [22] I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, Philadelphia, 2000.
- [23] P. Woelfel. Symbolic topological sorting with OBDDs. In *Mathematical Foundations of Computer Science 2003*, volume 2747 of *Lecture Notes in Computer Science*, pages 671–680. Springer, 2003.
- [24] A. Xie and P.A. Beerel. Implicit enumeration of strongly connected components. In *International Conference on Computer Aided Design 1999*, pages 37–40. ACM Press, 1999.