Online supplement to the paper "A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching"

Daniel Kowalczyk, Roel Leus
ORSTAT, Faculty of Economics and Business, KU Leuven, Naamsestraat 69, 3000 Leuven, Belgium
daniel.kowalczyk@kuleuven.be, roel.leus@kuleuven.be

1. Construction of ZDDs

A *recursive specification* S of $Z_{\mathcal{F}}$ is defined as a pair of functions $ROOT_S$ and $CHILD_S$ that return configurations: $ROOT_S$ is a function without arguments that returns the configuration of the root node of $Z_{\mathcal{F}}$, and $CHILD_S$ takes as input a configuration (j,t) of a node and $b \in \{0,1\}$ and outputs the configuration of the b-child of (j,t), where 0 and 1 refer to the low and high edge, respectively.

A generic approach to constructing the ZDD then follows Algorithm 1. This algorithm finds all the reachable configurations of recursive specification S from the root node to the terminal nodes. Using hash tables, one can establish a one-to-one correspondence between the nodes of the DD before the reduction and the configurations. If we assume the hash table operations and the recursive operations to run in constant time, then the runtime of Algorithm 1 is linear in the number of reachable configurations. This generic framework was presented in Iwashita and Minato (2013).

In order to construct the feasible set $\mathcal{S}$ of all feasible schedules, we need to define the functions $ROOT_S$ and $CHILD_S$, where S is the recursive specification of $Z_{\mathcal{S}}$. These functions are defined in Algorithm 2.

Algorithm 1: Generic top-down/breadth-first ZDD construction

Data: Recursive specification S for $\mathcal{F}$

Result: ZDD $Z_{\mathcal{F}}$

$(j_0, s_0) = ROOT_S()$;

Create a new node with configuration (j_0, s_0);

for $j = j_0$ **to** n

  for all nodes r with configuration (j, s) for some s

    for $b \in \{0, 1\}$

      $(j', s') = CHILD_S((j, s), b)$;

      if (j', s') corresponds to one of the terminal nodes **then**

        Set the terminal node to be the b-child of r;

      else

        Find or create a node r' with configuration (j', s');

        Set r' to be the b-child of r;

Apply the reduction algorithms for ZDDs to the constructed DD;

2. Constructing child nodes

We construct ZDDs for the child nodes of the B&B tree. For this we present a rather general recursive specification. We derive a recursive specification $S_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}$ for $\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}$. For every $j \in J$, define A_j as the set $\{j' \in J \mid \{j, j'\} \in E_{SAME}\}$ and B_j as the set $\{j' \in J \mid \{j, j'\} \in E_{DIFF}\}$. The configurations for recursive specification $S_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}$ are pairs $(j, (S_{ADD}, S_{REMOVE}))$ where $j \in J$, $S_{ADD} \subset J$ is a job set that contains all the jobs have that to be added based on the choices made higher in the ZDD, and $S_{REMOVE} \subset J$ contains all the jobs that cannot be added based on the previous choices. The configurations of the **1**-node and the **0**-node are defined as before. The configuration associated to the root node of $Z_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}$ is a pair for which the first component is 1 and the second component is $(\emptyset, \emptyset)$. The high child of a node with configuration $(j, (S_{ADD}, S_{REMOVE}))$ is given by $(j', (S_{ADD} \cup A_j, S_{REMOVE} \cup B_j))$ where $j' = \min(\{i > j | i \notin S_{REMOVE} \cup B_j\} \cup \{n+1\})$. The low child of a node with configuration $(j, (S_{ADD}, S_{REMOVE}))$ is given by $(j', (S_{ADD}, S_{REMOVE} \cup A_j))$ with $j' = \min(\{i > j | i \notin S_{REMOVE} \cup A_j\} \cup \{n+1\})$. For implementation details we refer to Algorithm 3.

Algorithm 2: Calculation of all reachable configurations of $\mathcal{S}$

Function $ROOT_S()$
  return $(1,0)$

Function $CHILD_S((j,t),b)$
  if $b=1$ **then**
    $t' \leftarrow t+p_j$;
  else
    $t' \leftarrow t$;
  $j' \leftarrow MINJOB(j,t')$;
  if $j' \leq n$ **then**
    if $t'+sum_{j'} < H_{\min}$ **then**
      return $(n+1,0)$;
    if $t' \in [H_{\min}, H_{\max}]$ *and* $t'+m_{j'} > H_{\max}$ **then**
      return $(n+1,1)$;
  else
    if $t' \in [H_{\min}, H_{\max}]$ **then**
      return $(n+1,1)$;
    return $(n+1,0)$
  return (j',t');

Function $MINJOB(j,t)$
  if $\min\{i>j | t \in [r_i, d_i-p_i]\}$ *exists* **then**
    return $\min\{i>j | t \in [r_i, d_i-p_i]\}$;
  return $n+1$;

The ZDD of the child nodes in the B&B algorithm can now be set up as follows. Suppose that we apply the generic branching scheme on the jobs j and j'. Let Z be the ZDD at the parent node and S its recursive configuration. The ZDD of the SAME child is the output of the top-down/breadth-first algorithm with recursive specification $S \cap S_{SAME}$, where $S_{SAME} = S_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}$ with $\mathcal{F}_{SAME} = \{s \subset J \mid a_{sj} = a_{sj'}\}$ and $\mathcal{F}_{DIFF} = 2^J$. The ZDD

of the DIFF child can be obtained similarly with recursive specification $S \cap S_{SAME}$, where $S_{SAME} = S_{\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}}$ with $\mathcal{F}_{DIFF} = \{s \subset J \mid a_{sj} + a_{sj'} \leq 1\}$ and $\mathcal{F}_{SAME} = 2^J$.

References

Iwashita, H., S.-I. Minato. 2013. Efficient top-down ZDD construction techniques using recursive specifications. Tech. Rep. TCS-TR-A-13-69, Hokkaido University, Graduate School of Information Science and Technology.

Algorithm 3: Calculation of all reachable configurations of $\mathcal{F}_{SAME} \cap \mathcal{F}_{DIFF}$

Function $CHILD_S((j,(S_{ADD},S_{REMOVE})),b)$

- **if** $A_j \cap B_j \neq \emptyset$ **then**
  - **return** $(n+1,0)$
- **if** $b=1$ **then**
  - **if** $j \in S_{REMOVE}$ **then**
    - **return** $(n+1,0)$;
  - $S'_{ADD} \leftarrow S_{ADD} \cup A_j$;
  - $S'_{REMOVE} \leftarrow S_{REMOVE} \cup B_j$;
- **else**
  - **if** $j \in S_{ADD}$ **then**
    - **return** $(n+1,0)$;
  - $S'_{REMOVE} \leftarrow S_{REMOVE} \cup A_j$;
- $j' \leftarrow \min(\{i > j \mid i \in S'_{REMOVE}\} \cup \{n+1\})$;
- **if** $j' = n+1$ *and* b **then**
  - **if** $j \notin S_{REMOVE}$ **then**
    - **return** $(n+1,1)$;
  - **return** $(n+1,0)$
- **else**
  - **if** $j \notin S_{ADD}$ **then**
    - **return** $(n+1,1)$;
  - **return** $(n+1,0)$
- **return** $(j',(S'_{ADD},S'_{REMOVE}))$