AllRight: A BDD Based Approach to the Routing and Wavelength Assignment Problem

Gustav S. Bruhns, Martin P. Hansen, Rasmus Hebsgaard, and Frederik M. W. Hyldgaard

Aalborg University, Institute of Computer Science

Abstract. The application of Wavelength Division Multiplexers (WDM) in optical fiber networks is a well-known approach to increase bandwidth. A problem that arises for WDM optical fiber networks is the Routing and Wavelength Assignment problem (RWA), as the demands of the network must each be assigned a wavelength without causing wave interference. State-of-the-art tools that solve the RWA problem find only a single solution which provides little flexibility for the network engineers. We present a novel approach to solving the RWA problem by using Binary Decision Diagrams (BDD) to find all possible solutions to the RWA problem. We implement our approach as the tool AllRight and provide different techniques to increase the efficiency of AllRight. We then evaluate AllRight by comparing it to the state-of-the-art method Mixed Integer Programming (MIP) that only finds a single solution. From the results, we find that AllRight is not as fast as MIP, but AllRight finds all possible solutions to the RWA problem as opposed to finding a single solution.

Keywords: binary decision diagrams, routing and wavelength assignment

1 Introduction

Optical fiber networks use Wavelength Division Multiplexers (WDM) [1] to transmit data demands through links in the network, where each link has a number of possible wavelength channels. A key advantage of WDMs is that they increase the bandwidth of optical fiber networks [1, 2]. However, a problem that arises when using WDMs in optical fiber networks is that demands transmitted on the same wavelength cannot be routed through the same links due to wave interference. This problem is referred to as the Routing and Wavelength Assignment (RWA) problem [2]. Specifically, in the RWA problem, each demand must be assigned a wavelength and must be routed from an ingress router to an egress router such that no demands on the same wavelength are routed through the same links. Generally, the RWA problem can be split into three variants: the static, incremental, and dynamic RWA problem [2]. The static RWA problem assumes that all demands are known in advance, and it does not consider any future changes in the network traffic. The incremental RWA problem considers one demand at a time for a fixed total number of demands. Like the static RWA problem, it assumes no future changes. Lastly, the dynamic RWA problem routes and assigns wavelengths to demands as they change in the network. Because the RWA problem and its variants have been studied for many years, a wide range of techniques have been developed to solve it [2, 3, 4, 5]. However, to the best of our knowledge, these techniques find only a single solution to the RWA problem. A network engineer will thereby not be able to choose from a number of solutions, which can prove to be disadvantageous from a pragmatic viewpoint. For example, the provided solution may route the demands through links that the network engineer knows are unstable from experience.

Our Contributions. We present the software tool AllRight which computes all possible solutions to a static RWA problem. AllRight represents the solutions compactly in a Binary Decision Diagram (BDD) [6], from which a specific solution can be picked. We prove that the BDD represents all possible solutions to the RWA problem. To increase the speed of AllRight, we describe techniques to reduce the number of equivalent solutions, to divide the RWA problem into subproblems that

can be solved in parallel, and to restrict the solutions found by AllRight to only the optimal ones. Additionally, we also provide an alternative encoding of the RWA problem that scales the number of variables in the BDD logarithmically on the number of demands.
Because no other tool finds all solutions, we compare our tool to the method from [3] which uses Mixed Integer Programming (MIP) to find a single optimal solution to the RWA problem. Despite the aforementioned techniques, we find that AllRight has a longer run time than the method from [3] when running experiments on network topologies from the Topology Zoo benchmark [7]. However, AllRight does find all solutions as opposed to only one solution, which can be useful for network engineers. The source code of AllRight and the tools, data and results used for the experiments can be found in [8].

Organization of the Paper. The rest of the paper is organized as follows. In Section 2, we provide an overview of related works. We then present our formalization of the RWA problem in Section 3, which we encode using BDDs in Section 4. In Section 5 we evaluate the implementation of the BDD encoding. In Section 6, we explore different approaches to improve the time to build the BDD for the RWA problem. Lastly, we conclude in Section 7 and propose some possible future areas of research in Section 8.

2 Related Work

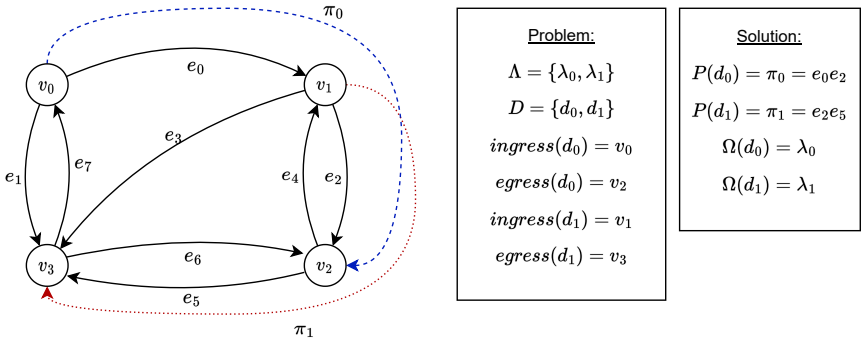
The Routing and Wavelength assignment problem for optical networks is an NP-complete problem [9] and a well-studied topic, where much of the seminal work dates back to the 90s [10, 11, 2]. In these works, the RWA problem is defined as an Integer Linear Programming (ILP) problem. As it is unlikely that an efficient polynomial solution exists for the RWA problem, many works study a simpler version of the problem, where routing and wavelength assignment are decomposed into separate stages [2]. Such an approach has no guarantee that the combined solutions for the two stages result in an optimal solution to the complete RWA problem, which we will show in Section 3. A possible decomposition involves solving the routing problem using Mixed Integer Programming (MIP) to find paths that minimize the number of demands going over any link, and solving the wavelength assignment problem using a variety of graph coloring methods such as MIP and SAT [3].

More recent works leverage genetic programming [5] and machine learning [12] to find near-optimal solutions. In [4], Cicco et al. implement a deep reinforcement approach to solve the RWA problem. The solution is compared against several greedy heuristics, genetic programming, as well as ILP to serve as the benchmark. The results suggest DRL is a middleground between the fast computational speed of the greedy heuristics and the solution quality of the genetic algorithm and ILP. Both the machine learning and genetic algorithms do not guarantee soundness in regards to their solution being an optimal solution to the RWA problem. Our method differs from these approaches as it guarantees that all optimal solutions are found. Additionally, to the extent of our knowledge, all of the existing approaches to the RWA problem focus on finding a singular solution, and not all optimal solutions. However, from the pragmatic viewpoint of a network operator, a wider selection of solutions offers more agency to be able to configure networks based on the current situation, and thus this is what we focus with the tool presented in this paper.

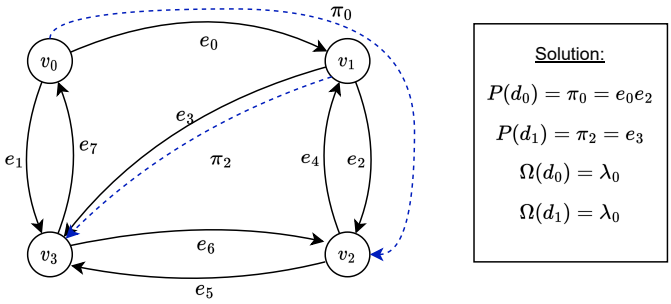
3 A Model for Routing and Wavelength Assignment

In this section, we present our formalisation of the Routing and Wavelength Assignment (RWA) problem. First, we represent a given network topology for an optical network as a directed multigraph, where nodes correspond to routers, and edges correspond to links.

Definition 1 (Network Topology). *A* network topology *is a directed multigraph* $G = (V, E, src, tgt)$ *where* V *is a finite set of nodes (routers),* E *is a finite set of edges (links) and*



(a) Non-optimal solution.



(b) Optimal solution.

Fig. 1: Example of RWA problem with two demands and two wavelengths, including two possible paths and wavelength assignments to the demands.

$src, tgt : E \to V$ represent the source and target of the edge respectively. We require for all edges that $src(e) \neq tgt(e)$.

In the network topology, data flows from some ingress router to some egress router. We represent each such flow as a demand.

Definition 2 (Demands). *A set of* demands *is a finite set D, and $ingress, egress : D \to V$ represent the source and target node of the demand, respectively.*

The route of a demand is a path through the network, starting in the ingress router and ending in the egress router. We represent a path in the network topology as a sequence of edges.

Definition 3 (Path). *A* path *π from node s to node t is a finite sequence of edges $e_0e_1e_2...e_n$ such that $tgt(e_i) = src(e_{i+1})$ for $0 \leq i < n$, and $src(e_0) = s$ and $tgt(e_n) = t$. A* simple path *is a path which has no repeating vertices. Let **Path** be the set of all simple paths.*

In this paper, we focus on simple paths for demands. A demand can have different simple paths through the network. Hence, a simple path has to be chosen for each demand, and this corresponds to the routing part of the RWA problem. We represent the assignments of simple paths to demands as a mapping between demands and simple paths.

Definition 4 (Path Assignment). *A* path assignment *is a total function $P : D \to$ **Path** such that for every demand d if $P(d) = e_0e_1e_2...e_n$, then $src(e_0) = ingress(d)$ and $tgt(e_n) = egress(d)$.*

An example network topology with two demands d_0 and d_1 is shown in Figure 1a. The network consists of four nodes with eight edges in total. The demand d_0 has node v_0 as its source and node v_2 as its target. There are multiple paths from the source to target, but in this example, d_0 is assigned the simple path $\pi_0 = e_0e_2$.

In addition to routing, each demand must also be assigned a wavelength $\lambda \in \Lambda$ where Λ is a finite set of wavelengths. Similarly to the simple path assignment to demands, we define the wavelength assignment as a mapping between demands and wavelengths.

Definition 5 (Wavelength Assignment). *A* wavelength assignment *for a given set of demands D and a finite set of possible wavelengths Λ is a total function $\Omega : D \to \Lambda$.*

In Figure 1a, demand d_0 is assigned a wavelength λ_0, which is represented by the dashed blue line. Thus, $\Omega(d_0) = \lambda_0$. However, not all wavelength assignments are valid, as wavelength clashing between demands must be avoided in the RWA problem. That is, demands that are assigned the same wavelength cannot be routed over the same edge.

Definition 6 (Wavelength Clashing between Demands). *Let $\pi \cap \pi'$ denote the set of edges that two paths share. Then, a pair of demands $d, d' \in D$ is said to* wavelength clash *if $d \neq d'$ and $P(d) \cap P(d') \neq \varnothing$ and $\Omega(d) = \Omega(d')$.*

As seen in Figure 1a, the two demands d_0 and d_1 share the same edge $P(d_0) \cap P(d_1) = \{e_2\}$. Hence, with the current paths, if the demands are assigned the same wavelength λ, there will be a wavelength clash. In the example, the problem is alleviated by assigning the demands different wavelengths such that $\Omega(d_0) \neq \Omega(d_1)$. Alternatively, demand d_1 can be routed by $P(d_1) = e_3$, allowing any wavelength assignment.

To solve the RWA problem, we must find a simple path assignment and wavelength assignment for all demands such that no two demands wavelength clash.

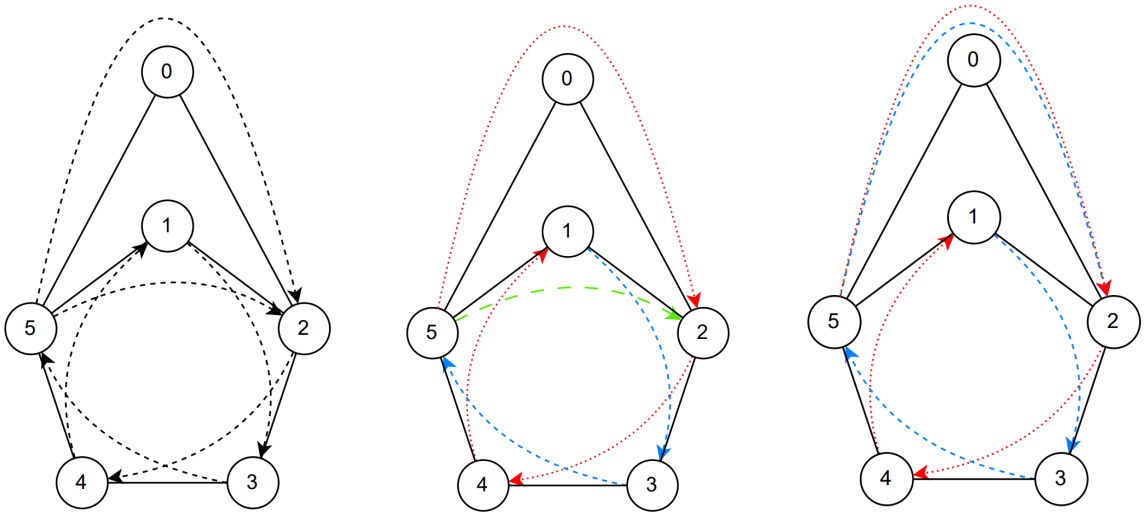
Definition 7 (Routing and Wavelength Assignment Problem). *Given a network topology G, a set of demands D, and a set of wavelengths Λ, the* Routing and Wavelength assignment problem *is to find a path assignment P and a wavelength assignment Ω such that no pair of distinct demands $d, d' \in D$ wavelength clash.* A solution to the routing and wavelength assignment problem *is then a pair (P, Ω) that solves the RWA problem.*

Figure 1a shows a simple example of a solution to a RWA problem. There are two demands d_0 and d_1, and both demands are assigned a simple path from their source to target node. The simple paths for the demands overlap on edge e_2, but they do not wavelength clash, since $\Omega(d_0) \neq \Omega(d_1)$. However, other solutions exist, where the two demands are assigned the same wavelength while still not wavelength clashing. One such solution is shown in Figure 1b, where $\Omega(d_0) = \Omega(d_1)$ and $P(d_0) \cap P(d_1) = \varnothing$. We consider solutions like this to be optimal, since the minimal possible number of different wavelengths are assigned to the demands.

Definition 8 (Network Optimal Routing and Wavelength Assignment). *A* network optimal solution *is a solution to the RWA problem (P, Ω) where $|\Omega(D)| \leq |\Omega'(D)|$ for any other solution to the RWA problem (P', Ω').*

3.1 Ensuring Optimality

As mentioned in Section 2, a common approach to solving the RWA problem is decomposing the problem into two independent phases, routing and wavelength assignment. However, this approach does not have a guarantee of finding network optimal solutions. To show this, we provide a counter



(a) Routing of demands minimizing the number of demands over each edge.

(b) Wavelength assignment using 3 wavelength.

(c) Optimal wavelength assignment using 2 wavelengths.

Fig. 2: Network where a solid line corresponds to a undirected connection, and a dashed line with an arrow corresponds to a demand.

example in Figure 2. Figure 2a shows a possible optimal solution to the first phase of the decomposition approach, the routing assignment of the demands, as it routes at most 2 demands over each edge. Given this routing assignment, it can be trivially shown that the wavelength assignment requires 3 wavelengths, as shown in Figure 2b. However, as shown in Figure 2c, using another optimal solution for the routing assignment yields a wavelength assignment that uses 2 wavelengths. With our tool, AllRight, we verify that 2 wavelengths is the optimal solution for this problem.

In the following section, we describe our approach to solving the RWA problem using BDDs, and show that this approach is sound.

4 BDD Encoding of RWA

A Binary Decision Diagram (BDD) is a directed, acyclic graph structure used to compactly represent Boolean functions [6]. Within a BDD, non-leaf nodes are labeled with Boolean variables, while leaf nodes are labelled with truth values 0 ($False$) or 1 ($True$). Each non-leaf node u has two outgoing edges denoted as $low(u)$ and $high(u)$ corresponding to its label variable being $False$ or $True$ respectively. These edges are commonly visualized by using a solid line for $high(u)$ and a dotted line for $low(u)$ [6]. A BDD is said to be an ordered BDD (OBDD) if the variables in the BDD come in the same order $x_1 < x_2 < ... < x_n$ on all paths of the BDD [6]. An OBDD can be reduced by merging nodes with identical subgraphs, and by deleting nodes where the subgraphs for $low(u)$ and $high(u)$ are equivalent. Such a BDD is called a reduced OBDD (ROBDD) [6]. In the following sections, we use the short form BDD in place of ROBDD. Lastly, we note that we use BDDs that support first-order quantifiers, and that these BDDs are closed under both Boolean operations and quantifiers [13].

Our goal is to represent the RWA problem as a Boolean function using BDDs. We use BDDs to compute and store all valid solutions, such that a valid solution is a truth assignment which

makes the Boolean function for the RWA problem evaluate to $True$. We do this by constructing a set of simple BDDs to describe the RWA problem. Using these simple BDDs as building blocks, we construct increasingly complex BDDs until we have a single BDD from which all valid combinations of path and wavelength assignments can be inferred. To construct these BDDs, we must be able to represent sets of elements as Boolean expressions.

4.1 Representation of Sets as Boolean Expressions

A finite set of elements S can be represented as a vector of Boolean variables containing a variable for each element in the set. Any subset $S' \subseteq S$ can then be represented by a truth assignment to the vector, where a variable being assigned to $True$ means that the corresponding element is in S', and a variable assigned to $False$ means that the corresponding element is not in S'. Due to the one-to-one correspondence between elements of the the set and the Boolean variables we call this an unary set encoding.

Definition 9 (Unary set encoding). *Given a finite set $S = \{s_0, s_1, ..., s_{|S|-1}\}$, let $\overline{\mathbf{x}} = [\boldsymbol{x}_{s_0}, \boldsymbol{x}_{s_1}, ..., \boldsymbol{x}_{s_{|S|-1}}]$ be a vector of Boolean variables. Then, $\overline{\mathbf{x}}$* encodes a subset *$S' \subseteq S$ s.t. $\boldsymbol{x}_{s_i}$ is true iff $s_i \in S'$.*

Elements of a finite set can also be represented using a standard binary encoding, like seen in [13]. If we have an enumeration $s_0, s_1, ..., s_{|S|-1}$ of the finite set S, we can uniquely identify each element using $k = \lceil log_2(|S|) \rceil$ Boolean variables $\overline{\mathbf{x}} = [\mathbf{x_{k-1}}, ..., \mathbf{x_0}]$, since there are $2^{\lceil log_2(|S|) \rceil} \geq |S|$ different variable assignments for $\lceil log_2(|S|) \rceil$ variables.

Definition 10 (Binary set encoding). *Given a finite set $S = \{s_0, s_1, ..., s_{|S|-1}\}$, let $\overline{\mathbf{x}} = [\mathbf{x_{k-1}}, ..., \mathbf{x_0}]$ be a vector of Boolean variables where $k = \lceil log_2(|S|) \rceil$. Then, any truth assignment α to $\overline{\mathbf{x}}$ can be interpreted as a natural number $n(\alpha) \in \mathbb{N}$ written in binary notation. Thus $\overline{\mathbf{x}}$ encodes the $n(\alpha)$'th element of S. Let $\overline{\mathbf{x}}(s)$ denote the Boolean expression over $\overline{\mathbf{x}}$ with just the single truth assignment corresponding to $\{s\}$. Lastly, given some Boolean expression $b(\overline{\mathbf{x}})$, let $[\![b(\overline{\mathbf{x}})]\!]$ denote the encoded subset $\{s_{n(\alpha)} | \alpha$ satisfies $b(\overline{\mathbf{x}})\} \subseteq S$, such that $[\![b(\overline{\mathbf{x}})]\!]$ is a set consisting of the elements, that are encoded by the Boolean assignments which satisfy the Boolean expression b.*

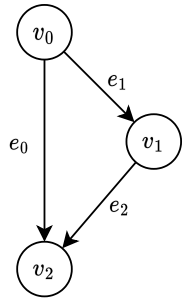
Example: Given a set of nodes $V = \{v_0, v_1, v_2, v_3\}$, we can represent any node in the set using two binary variables $\overline{\mathbf{x}} = [\mathbf{x_1}, \mathbf{x_0}]$. As an example, the encoding of $\{v_2\}$ is $\overline{\mathbf{x}}(v_2) = \mathbf{x_1} \wedge \neg \mathbf{x_0}$. If we have the Boolean expression $b = \mathbf{x_0}$, then $[\![b(\overline{\mathbf{x}})]\!] = \{v_1, v_3\}$, since the truth assignments satisfying b are $[\mathbf{x_1} \mapsto 0, \mathbf{x_0} \mapsto 1]$ and $[\mathbf{x_1} \mapsto 1, \mathbf{x_0} \mapsto 1]$, which correspond to nodes v_1 and v_3.

4.2 Construction of BDDs

From the definitions in Section 3, the RWA problem is defined using the four sets V, E, D, and Λ. We use the binary encoding method from Definition 10 to encode the elements of these sets as Boolean expressions. Furthermore, we use an unary encoding of the set E to encode the paths introduced in Definition 3. An overview of the encoding notation is given in Table 1.

Variables	Description
$\overline{\mathbf{v}} = [\mathbf{v}_n, \mathbf{v}_{n-1}, ..., \mathbf{v}_0], \overline{\mathbf{s}} = [\mathbf{s}_n, \mathbf{s}_{n-1}, ...\mathbf{s}_0], \overline{\mathbf{t}} = [\mathbf{t}_n, \mathbf{t}_{n-1}, ...\mathbf{t}_0]$ where $n = \lceil \log_2(\lvert V \rvert) \rceil$	Binary encodings for nodes in V
$\overline{\mathbf{e}} = [\mathbf{e}_n, \mathbf{e}_{n-1}, ..., \mathbf{e}_0]$ where $n = \lceil \log_2(\lvert E \rvert) \rceil$	Binary encoding for edges in E
$\overline{\mathbf{d}} = [\mathbf{d}_n, \mathbf{d}_{n-1}, ..., \mathbf{d}_0]$ where $n = \lceil \log_2(\lvert D \rvert) \rceil$	Binary encoding for demands in D
$\overline{\boldsymbol{\lambda}} = [\boldsymbol{\lambda}_n, \boldsymbol{\lambda}_{n-1}, ..., \boldsymbol{\lambda}_0]$ where $n = \lceil \log_2(\lvert \Lambda \rvert) \rceil$	Binary encoding for wavelengths in Λ
$\overline{\mathbf{p}} = [\mathbf{p}_{e_0}, \mathbf{p}_{e_1}, ..., \mathbf{p}_{e_{\lvert E \rvert - 1}}]$	Unary encoding for a subset of edges $E' \subseteq E$

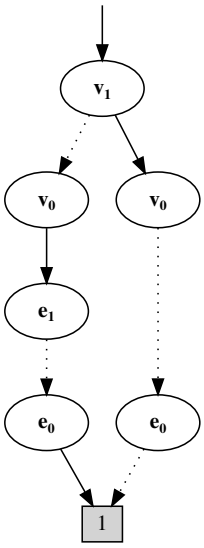
Table 1: Variables used in the BDD encodings.



(a) Small network consisting of 3 nodes and edges.

$$(\neg \mathbf{v_1} \wedge \mathbf{v_0}) \wedge (\neg \mathbf{e_1} \wedge \mathbf{e_0}) \quad \vee$$
$$(\mathbf{v_1} \wedge \neg \mathbf{v_0}) \wedge (\neg \mathbf{e_1} \wedge \neg \mathbf{e_0}) \vee$$
$$(\mathbf{v_1} \wedge \neg \mathbf{v_0}) \wedge (\mathbf{e_1} \wedge \neg \mathbf{e_0})$$

(b) The Boolean expression represented by $\boldsymbol{in}$.



(c) The BDD $\boldsymbol{in}(\overline{\mathbf{v}}, \overline{\mathbf{e}}) = \boldsymbol{in}([\mathbf{v}_1, \mathbf{v}_0], [\mathbf{e}_1, \mathbf{e}_0])$.

Fig. 3: Example of how to encode the BDD $\boldsymbol{in}$ for a small network.

We now encode the RWA problem described in Section 3 by constructing BDDs, using the encodings in Table 1. First, we need to encode a given network topology $G = (V, E, src, tgt)$. We use the binary variable encodings $\overline{\mathbf{v}}$ and $\overline{\mathbf{e}}$ to encode any $v \in V$ and $e \in E$ respectively. To encode the mappings src and tgt, we define the BDDs $\boldsymbol{out}$ and $\boldsymbol{in}$ as follows

$$\boldsymbol{out}(\overline{\mathbf{v}}, \overline{\mathbf{e}}) = \bigvee_{v \in V} \bigvee_{e \in E, src(e) = v} (\overline{\mathbf{v}}(v) \wedge \overline{\mathbf{e}}(e)) \tag{1}$$

$$\boldsymbol{in}(\overline{\mathbf{v}}, \overline{\mathbf{e}}) = \bigvee_{v \in V} \bigvee_{e \in E, tgt(e) = v} (\overline{\mathbf{v}}(v) \wedge \overline{\mathbf{e}}(e)) \tag{2}$$

and clearly $(v, e) \in [\![\boldsymbol{out}(\overline{\mathbf{v}}, \overline{\mathbf{e}})]\!]$ iff $src(e) = v$ and $(v, e) \in [\![\boldsymbol{in}(\overline{\mathbf{v}}, \overline{\mathbf{e}})]\!]$ iff $tgt(e) = v$.

Example: Figure 3c shows an example of $\boldsymbol{in}$, encoding the network topology depicted in Figure 3a. The BDD $\boldsymbol{in}$ contains all assignments to the Boolean variables that satisfy the Boolean expression shown in Figure 3b. An example of such an assignment is $[\mathbf{v_1} \mapsto 0, \mathbf{v_0} \mapsto 1, \mathbf{e_1} \mapsto 0, \mathbf{e_0} \mapsto 1]$ which is found by following the left path in Figure 3c. From this assignment, we know that $\alpha(\overline{\mathbf{v}}) = [0, 1]$ encodes node v_1 and $\alpha(\overline{\mathbf{e}}) = [0, 1]$ encodes edge e_1. As such, $(v_1, e_1) \in [\![\boldsymbol{in}(\overline{\mathbf{v}}, \overline{\mathbf{e}})]\!]$, which follows from $tgt(e_1) = v_1$ as seen in Figure 3a.

We encode the demands in the network using the binary variable encoding $\overline{\mathbf{d}}$. To encode the mappings *ingress* and *egress* from Definition 2, we define the BDDs `source` and `target` as follows

$$\texttt{source}\,(\overline{\mathbf{v}}, \overline{\mathbf{d}}) = \bigvee_{d \in D} \overline{\mathbf{v}}(ingress(d)) \wedge \overline{\mathbf{d}}(d) \tag{3}$$

$$\texttt{target}\,(\overline{\mathbf{v}}, \overline{\mathbf{d}}) = \bigvee_{d \in D} \overline{\mathbf{v}}(egress(d)) \wedge \overline{\mathbf{d}}(d) \tag{4}$$

and clearly $(v, d) \in [\![\texttt{source}\,(\overline{\mathbf{v}}, \overline{\mathbf{d}})]\!]$ iff $ingress(d) = v$ and $(v, d) \in [\![\texttt{target}\,(\overline{\mathbf{v}}, \overline{\mathbf{d}})]\!]$ iff $egress(d) = v$.

As part of solving the RWA problem, each demand is assigned a simple path through the given network topology. Hence, we need a way to encode a simple path. We do this using a vector of Boolean variables $\overline{\mathbf{p}} = [\mathbf{p}_{e_0}, \mathbf{p}_{e_1}, ..., \mathbf{p}_{e_{|E|-1}}]$ which encodes the subset $E' \subseteq E$ using unary set encoding as described in Definition 9. Now, the goal is to ensure that the subset E' encoded by the vector of variables $\overline{\mathbf{p}}$ represents a simple path.

Definition 11 (Set representation of simple paths). *The subset $E' \subseteq E$* represents a simple path *if the edges in E' can be permuted into a sequence of edges that satisfy the definition of a simple path.*

To keep track of what edges $e \in E$ are part of a path, we define the BDD `passes` as

$$\texttt{passes}\,(\overline{\mathbf{e}}, \overline{\mathbf{p}}) = \bigvee_{e \in E} \overline{\mathbf{e}}(e) \wedge \mathbf{p}_e \tag{5}$$

and clearly $(e, E') \in [\![\texttt{passes}\,(\overline{\mathbf{e}}, \overline{\mathbf{p}})]\!]$ iff $\overline{\mathbf{p}}$ encodes $E' \subseteq E$ and $e \in E'$.

An important property of a simple path is that each vertex can have at most one outgoing edge on that path. To encode this property, we define the BDD `singleout` as follows

$$\texttt{singleOut}\,(\overline{\mathbf{v}}, \overline{\mathbf{p}}) = \forall \overline{\mathbf{e}}, \overline{\mathbf{e'}}.\texttt{out}\,(\overline{\mathbf{v}}, \overline{\mathbf{e}}) \wedge \texttt{out}\,(\overline{\mathbf{v}}, \overline{\mathbf{e'}}) \wedge \texttt{passes}\,(\overline{\mathbf{e}}, \overline{\mathbf{p}}) \wedge \texttt{passes}\,(\overline{\mathbf{e'}}, \overline{\mathbf{p}}) \implies \overline{\mathbf{e}} = \overline{\mathbf{e'}} \tag{6}$$

and clearly $(v, E') \in [\![\texttt{singleOut}\,(\overline{\mathbf{v}}, \overline{\mathbf{p}})]\!]$ iff $\overline{\mathbf{p}}$ encodes $E' \subseteq E$ and $|E' \cap \{e | e \in E \wedge src(e) = v\}| \leq 1$.

To enforce that the vector $\overline{\mathbf{p}}$ encodes a simple path, we define the BDD `path` which encodes all possible simple paths between any source node s and target node t. We use the binary variable encoding $\overline{\mathbf{s}}$ and $\overline{\mathbf{t}}$ to encode any $s, t \in V$.

Definition 12 (Path). *The BDDs $\texttt{path}^k, k \geq 0$, are defined as:*

$$\texttt{path}^0(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}}) = (\overline{\mathbf{s}} = \overline{\mathbf{t}}) \wedge \bigwedge_{e \in E} \neg \mathbf{p}_e \tag{7}$$

and for $k > 0$ we have

$$
\begin{aligned}
\mathit{path}^k(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}}) \;=\; & && (8)\\
& \mathit{path}^{k-1}(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}}) \vee && (9)\\
& \Big(\exists \overline{\mathbf{v}}, \overline{\mathbf{e}}, \overline{\mathbf{p'}}. \mathit{out}(\overline{\mathbf{s}}, \overline{\mathbf{e}}) \wedge \mathit{in}(\overline{\mathbf{v}}, \overline{\mathbf{e}}) \wedge && (10)\\
& \exists e \in E \,.\, \Big(\overline{\mathbf{e}}(e) \wedge \mathbf{p}_e \wedge \neg \mathbf{p}'_e \wedge \big(\bigwedge_{e' \in E \setminus \{e\}} \mathbf{p}_{e'} = \mathbf{p}'_{e'} \big) \Big) \wedge && (11)\\
& \mathit{path}^{k-1}(\overline{\mathbf{v}}, \overline{\mathbf{t}}, \overline{\mathbf{p'}}) \wedge && (12)\\
& (\overline{\mathbf{s}} \neq \overline{\mathbf{t}}) \wedge \mathit{singleOut}(\overline{\mathbf{s}}, \overline{\mathbf{p}}) \Big). && (13)
\end{aligned}
$$

Observe that $[\![\mathit{path}^{k-1}]\!] \subseteq [\![\mathit{path}^k]\!]$ by Condition 9. Since these BDDs consist of a finite number of Boolean variables, eventually $\mathit{path}^k = \mathit{path}^{k+1}$, and we refer to this BDD path^k as path.

The BDD path encodes all possible simple paths from some source node s to some target node t in the network. By Condition 7, it includes all simple paths from a node to itself, i.e. $s = t$. For each following step, the BDD contains all previous simple paths of lengths $k-1$ by Condition 9 and the simple paths of length k. Simple paths of length k are found by extending the paths of length $k-1$ with an edge going out of a node s into a node v (Condition 10) which is known to have a path of length $k-1$ to the target node due to Condition 12. The addition of a new edge to the path is handled by Condition 11. To ensure that the paths are still simple paths, Condition 13 enforces that the new source node is different from the target node and that it has at most one outgoing edge.

Lemma 1. *Let $E' \subseteq E$ and let Boolean variables $\overline{\mathbf{p}}$ encode the set E'. We then have $(s, t, E') \in [\![\mathit{path}(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})]\!]$ iff E' is a representation of a simple path from s to t.*

Proof. We prove the left to right implication by induction on iteration k of path. The induction hypothesis is

$$IH(k)\text{: If } (s, t, E') \in [\![\mathit{path}^k(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})]\!] \text{ then } E' \text{ represents a simple path from } s \text{ to } t.$$

For the base case $k = 0$, let $(s, t, E') \in [\![\mathit{path}^0(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})]\!]$. From Condition 7, we have that $s = t$ and, since all variables in $\overline{\mathbf{p}}$ are set to false, $E' = \emptyset$. Thus, E' represents the simple path from s to s. For the inductive step $k > 0$, let $(s, t, E') \in [\![\mathit{path}^k(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})]\!]$. We want to show that E' represents a simple path from s to t.

If s, t and E' satisfy Condition 9, then we have $(s, t, E') \in [\![\mathit{path}^{k-1}(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})]\!]$. Thus, we can apply the induction hypothesis $IH(k-1)$ to obtain that E' represents a simple path from s to t.
Otherwise, s, t and E' satisfy Conditions 10-13. By Condition 10, there must exist an edge e going from node s to some node v. Condition 11 ensures that all variables in $\overline{\mathbf{p}}$ and $\overline{\mathbf{p'}}$ must hold the same truth assignment, except for the variable for edge e which must be *False* in $\overline{\mathbf{p}}'$, and *True* in $\overline{\mathbf{p}}$. Hence, $\overline{\mathbf{p'}}$ must encode the set $E'' = E' \setminus \{e\}$.

By Condition 12, we have that $(v, t, E'') \in \mathit{path}^{k-1}(\overline{\mathbf{v}}, \overline{\mathbf{t}}, \overline{\mathbf{p'}})$. By the induction hypothesis, we know that E'' represents a simple path from v to t. We know that the edge e goes from node s to node v, where v is the source node in the simple path represented by E''. Thus, E' is a simple path if the node s is not already present in the simple path represented by E''. By Condition 13, node s cannot be node t, nor be any other node in the simple path represented by E'', as the node otherwise would have two outgoing edges. Hence E' must represent a simple path from s to t.

We now prove the right to left implication by induction on $|E'|$. The induction hypothesis is

$IH(k)$: If $E' \subseteq E$ represents a simple path from s to t where $|E'| \leq k$, and $\overline{\mathbf{p}}$ encodes the set E', then $(s, t, E') \in [\![\mathit{path}^k(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})]\!]$.

For the base case, $k = 0$, let $|E'| = 0$ and $\overline{\mathbf{p}}$ encode the set E'. Then we have that $E' = \emptyset$ represents a simple path from any node s to itself. Clearly, $(s, s, E') \in [\![\mathit{path}^0(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})]\!]$ by Condition 7. For the inductive step $k > 0$, let E' represent a simple path from s to t where $|E'| \leq k$ and $\overline{\mathbf{p}}$ encodes E'. If $|E'| < k$, then we can by induction hypothesis obtain that $(s, t, E') \in [\![\mathit{path}^{k-1}(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})]\!]$, thus satisfying Condition 9, meaning that $(s, t, E') \in [\![\mathit{path}^k(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})]\!]$.

If $|E'| = k$, then we argue that the encodings of s, t and E' satisfy Conditions 10-13. Since E' represents a simple path from s, there must be an edge e going from node s to some node v, thereby satisfying Condition 10. Additionally, since $\overline{\mathbf{p}}$ encodes E', the $\overline{\mathbf{p'}}$ constructed in Condition 11 must encode the set $E'' = E' \setminus \{e\}$. Since we only remove the edge e from E', E'' must represent a simple path from v to t. Hence, we can apply our induction hypothesis to obtain that $(v, t, E'') \in [\![\mathit{path}^{k-1}(\overline{\mathbf{v}}, \overline{\mathbf{t}}, \overline{\mathbf{p'}})]\!]$, satisfying Condition 12. Finally, Condition 13 is satisfied by E' by the definition of a simple path. Thus, we can conclude that $(s, t, E') \in [\![\mathit{path}^k(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}})]\!]$. □

Every demand in the network must be assigned a simple path from their source node to their target node. We encode all simple paths for a given demand by defining the BDD $\mathit{demandPath}$ as

$$\mathit{demandPath}(\overline{\mathbf{d}}, \overline{\mathbf{p}}) = \exists \overline{\mathbf{s}}, \overline{\mathbf{t}} \,.\, \mathit{path}(\overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{p}}) \wedge \mathit{source}(\overline{\mathbf{s}}, \overline{\mathbf{d}}) \wedge \mathit{target}(\overline{\mathbf{t}}, \overline{\mathbf{d}}) \tag{14}$$

such that $(d, E') \in [\![\mathit{demandPath}(\overline{\mathbf{d}}, \overline{\mathbf{p}})]\!]$ iff $\overline{\mathbf{p}}$ encodes $E' \subseteq E$ and E' is a representation of a simple path from node $s \in V$ to node $t \in V$ where $\mathit{ingress}(d) = s$ and $\mathit{egress}(d) = t$.

To avoid demands wavelength clashing as defined in Definition 6, we define the BDD $\mathit{noClash}$ as follows

$$\mathit{noClash}(\overline{\mathbf{d}}, \overline{\mathbf{d'}}, \overline{\mathbf{p}^d}, \overline{\mathbf{p}^{d'}}, \overline{\boldsymbol{\lambda}^d}, \overline{\boldsymbol{\lambda}^{d'}}) = \exists \overline{\mathbf{e}} \big(\mathit{passes}(\overline{\mathbf{e}}, \overline{\mathbf{p}^d}) \wedge \mathit{passes}(\overline{\mathbf{e}}, \overline{\mathbf{p}^{d'}}) \big) \implies (\overline{\boldsymbol{\lambda}^d} \neq \overline{\boldsymbol{\lambda}^{d'}}) \vee (\overline{\mathbf{d}} = \overline{\mathbf{d'}}) \tag{15}$$

such that $(d, d', \pi, \pi', \lambda, \lambda') \in [\![\mathit{noClash}(\overline{\mathbf{d}}, \overline{\mathbf{d'}}, \overline{\mathbf{p}^d}, \overline{\mathbf{p}^{d'}}, \overline{\boldsymbol{\lambda}^d}, \overline{\boldsymbol{\lambda}^{d'}})]\!]$ iff the demands $d, d' \in D$ follow the paths π and π' with wavelengths λ and λ' respectively, and the demands do not wavelength clash.

Lastly, we encode solutions (P, Ω) to the RWA problem described in Definition 7 by defining the BDD rwa, where we encode P with the vector of vectors $\overline{\mathbf{p}^D} = [\overline{\mathbf{p}^{d_0}}, \overline{\mathbf{p}^{d_1}}, ..., \overline{\mathbf{p}^{d_{|D|-1}}}]$ and Ω with the vector of vectors $\overline{\boldsymbol{\lambda}^D} = [\overline{\boldsymbol{\lambda}^{d_0}}, \overline{\boldsymbol{\lambda}^{d_1}}, ..., \overline{\boldsymbol{\lambda}^{d_{|D|-1}}}]$.

Definition 13 (Routing and wavelength assignments). *The BDD* rwa *is defined as*

$$\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D}) = \tag{16}$$

$$\bigwedge_{d \in D} \exists \overline{\mathbf{d}} \,.\, \Big(\overline{\mathbf{d}}(d) \wedge \mathit{demandPath}(\overline{\mathbf{d}}, \overline{\mathbf{p}^d}) \wedge \bigvee_{\lambda \in \Lambda} \overline{\boldsymbol{\lambda}^d}(\lambda) \Big) \wedge \tag{17}$$

$$\bigwedge_{d, d' \in D} \exists \overline{\mathbf{d}}, \overline{\mathbf{d'}} \,.\, \overline{\mathbf{d}}(d) \wedge \overline{\mathbf{d'}}(d') \wedge \mathit{noClash}(\overline{\mathbf{d}}, \overline{\mathbf{d'}}, \overline{\mathbf{p}^d}, \overline{\mathbf{p}^{d'}}, \overline{\boldsymbol{\lambda}^d}, \overline{\boldsymbol{\lambda}^{d'}}). \tag{18}$$

Note that Condition 17 enforces that each demand must be assigned a simple path, corresponding to a path assignment P, and a single wavelength, corresponding to a wavelength assignment Ω, while Condition 18 ensures that demands do not wavelength clash. The BDD rwa thus contains all valid solutions to the RWA problem, and we will refer to this BDD as the *baseline*.

Theorem 1. *There exists a solution* (P, Ω) *to the RWA problem iff* $(P, \Omega) \in [\![\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$.

Proof. For the left to right implication, assume (P, Ω) is a solution to the RWA problem. By Definition 7 of the RWA problem, the encodings $\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D}$ of the solution (P, Ω) satisfy Condition 17 and Condition 18. Hence, $(P, \Omega) \in [\![\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$.

For the right to left implication, let $(P, \Omega) \in [\![\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$. By Condition 17 and Lemma 1, we know for each demand d where $ingress(d) = s$ and $egress(d) = t$ that the mapping $P(d)$ defines a simple path from s to t. Additionally, we also know that each demand is assigned one wavelength by Ω. Lastly, by Condition 18, for each pair of distinct demands d, d', we know that Ω assigns wavelengths such that if the paths $P(d)$ and $P(d')$ share an edge, the demands are assigned different wavelengths. Hence, (P, Ω) is a solution to the RWA problem. □

It follows immediately from Theorem 1 that rwa also encodes all network optimal solutions as defined in Definition 8. The network optimal solutions in a given rwa can be found using Algorithm 1. The algorithm takes a BDD rwa and wavelengths Λ as input, and outputs a BDD $\mathit{rwaOptimal}$ that encodes exactly all network optimal solutions encoded by rwa.

Algorithm 1 Algorithm to find optimal solutions

Input: BDD rwa, wavelengths Λ.
Output: BDD $\mathit{rwaOptimal}$ encoding only the network optimal solutions encoded by rwa.

```
1: procedure GetOptimalSolutions(rwa, Λ)
2:     Let rwaOptimal be a BDD.
3:     rwaOptimal ← false
4:     ℓ ← 1
5:     while [[rwaOptimal]] = ∅ and ℓ ≤ |Λ| do
6:         rwaOptimal ← rwa ∧ ⋀_{d∈D} (⋁_{0≤i<ℓ} λ^d(λ_i))   ▷ Allow at most ℓ wavelengths to be used in rwa.
7:         ℓ ← ℓ + 1                                        ▷ Allow one more wavelength for next iteration.
8:     end while
9:     return rwaOptimal
10: end procedure
```

On lines 2-4, the BDD $\mathit{rwaOptimal}$ is initialized to have no valid assignments, and the variable ℓ to 1. Then, on line 5, we iterate lines 6-7 until $\mathit{rwaOptimal}$ contains at least one solution, or ℓ exceeds the maximum number of available wavelengths. On line 6, a $\mathit{rwaOptimal}$ is constructed for each iteration by conjugating rwa with the constraint that at most ℓ wavelengths can be assigned to the demands. The number of wavelengths allowed in a solution is then incremented by one on line 7 such that one more wavelength is allowed in the next iteration. Lastly, once the loop terminates, the BDD $\mathit{rwaOptimal}$ is returned on line 9.

Theorem 2. *Let the BDD* $\mathit{rwaOptimal}$ *be the output of Algorithm 1 given the input* rwa *and* Λ. *Then,* $(P, \Omega) \in [\![\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$ *is a network optimal solution to the RWA problem iff* $(P, \Omega) \in [\![\mathit{rwaOptimal}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$.

Proof. Obvious from analysis of Algorithm 1. □

5 Implementation and Evaluation of Baseline

We implement the tool AllRight to solve the RWA problem based on the BDDs defined in Section 4. AllRight is implemented in Python and leverages the CUDD package [14] to perform BDD computations. As input, the tool takes a set of demands, a number of wavelengths, and a network topology and outputs a BDD that contains all the solutions to the given problem. The time taken to compute this BDD is called the run time, and the number of nodes that constitute the BDD is called the size.

To evaluate our tool, we perform experiments on the baseline to find an effective configuration, and compare it to the state of the art. The experiments are run on a Ubuntu 18.04.5 cluster with 2.3 GHz AMD Opteron 6376 processors, with a memory limit of 50GB. To compare the solutions on a wide range of problems, we use the standard benchmark from Topology Zoo, consisting of 261 real world networks [7]. Based on the network topology and a desired number of demands, a set of concrete demands is randomly generated for each network; the tool ensures that all demands have at least one path between their ingress and egress nodes.

5.1 Experiments on Baseline

In the following, we analyze the baseline version of AllRight based on rwa as defined in Definition 13 to evaluate:

- how variable ordering affects run time and size of BDDs.
- how the run time scales with the number of wavelengths for a fixed number of demands.
- how the run time scales with the number of demands for a fixed number of wavelengths.

Effect of Variable Ordering It is important to identify a good variable order to use when constructing a BDD, as the specific variable order can heavily affect the number of nodes needed to encode a Boolean function in a BDD, and thus the time it takes to construct the BDD [15]. However, the problem of finding the optimal variable order, which minimizes the size of the BDD measured by the number of nodes, is NP-complete [16].

We now find a near-optimal variable order to use for the construction of the BDDs described in Section 4. As described in Table 1, we use 7 different groups of variables throughout the construction. In addition to these groups, for variable encodings $\overline{\mathbf{p}}$ and $\overline{\boldsymbol{\lambda}}$, we have both the general variables i.e. $\mathbf{p}_e$ and $\boldsymbol{\lambda}$ and the demand-specific versions $\mathbf{p}_e^d$ and $\boldsymbol{\lambda}^d$. For each permutation of these variable group orderings, we measure the run time for constructing the rwa BDD and its size. Furthermore, for each permutation, we examine the effect of placing the generic variables before the demand-specific variables, and whether it has an impact that the $\mathbf{p}_e^d$ variables are grouped by their corresponding edge or demand. In total this makes $7! \cdot 2 \cdot 2 = 20160$ different variable orderings to be examined. Lastly, we compare the best orderings with the default heuristic ordering offered by CUDD.

The baseline BDD is constructed for ten different networks with 5 demands and 5 wavelengths using each of the 20160 variable orderings. Each of the runs have a timeout of 60 seconds. In Figure 4, we plot the mean run time and mean size of the BDDs as a function of the variable orderings. As shown, there is an order of magnitude difference between the run times of the best and the worst variable orderings. Likewise, for the BDD sizes, there is a clear dependency on the variable orderings. It can also be seen that only 8400 of the variable orderings were able to finish before the time limit for all ten networks.

In Table 2, more details are shown for the five best orderings, the five worst orderings, and the five worst orderings that did not time out for all ten networks. The term *General First* means that the general variables $\mathbf{p}_e$ and $\boldsymbol{\lambda}$ precede their respective demand-specific variables, and the term *Edge Groupings* means that the $\mathbf{p}_e^d$ variables are grouped by their corresponding edge. As shown, the best variable ordering is $\overline{\mathbf{e}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{v}}, \overline{\mathbf{d}}, \overline{\mathbf{t}}, \overline{\mathbf{p}}, \overline{\mathbf{s}}$ using *Edge Groupings* and not *General First*. The build time

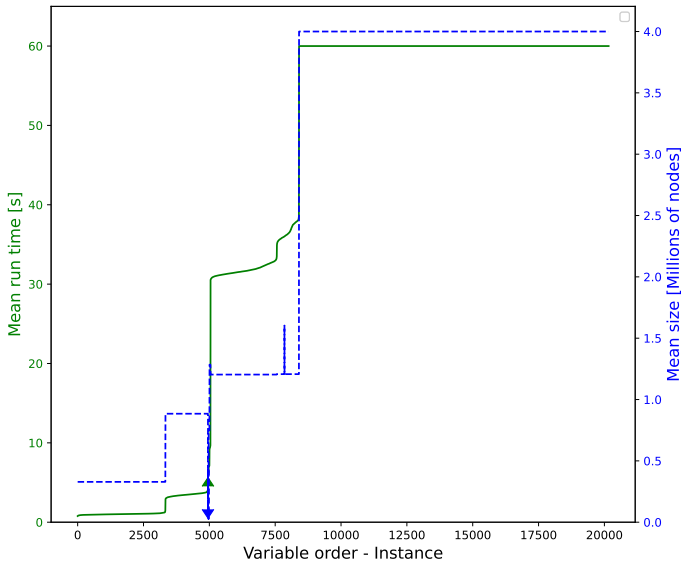


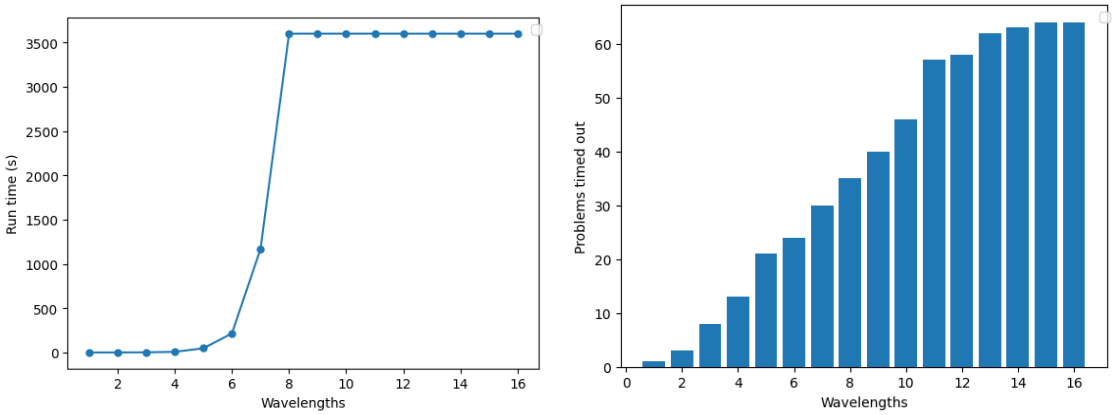
Fig. 4: Comparison of variable orderings. Variable orderings are ordered first by the run time then by the final size. The two triangles mark the values for the default heuristics used by CUDD.

#	Order	Edge Groupings	General First	Mean Time [s]	Mean Size
1	$\overline{\mathbf{e}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{v}}, \overline{\mathbf{d}}, \overline{\mathbf{t}}, \overline{\mathbf{p}}, \overline{\mathbf{s}}$	True	False	0.772	328820.8
2	$\overline{\mathbf{e}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{v}}, \overline{\mathbf{p}}, \overline{\mathbf{t}}, \overline{\mathbf{d}}, \overline{\mathbf{s}}$	True	False	0.776	328820.8
3	$\overline{\mathbf{e}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{v}}, \overline{\mathbf{d}}, \overline{\mathbf{s}}, \overline{\mathbf{p}}, \overline{\mathbf{t}}$	True	False	0.784	328820.8
4	$\overline{\mathbf{e}}, \overline{\mathbf{d}}, \overline{\mathbf{t}}, \overline{\mathbf{v}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{s}}, \overline{\mathbf{p}}$	True	True	0.805	328820.8
5	$\overline{\mathbf{e}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{p}}, \overline{\mathbf{s}}, \overline{\mathbf{d}}, \overline{\mathbf{t}}, \overline{\mathbf{v}}$	True	True	0.810	328820.8
...					
8396	$\overline{\mathbf{e}}, \overline{\mathbf{s}}, \overline{\mathbf{p}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{v}}, \overline{\mathbf{d}}, \overline{\mathbf{t}}$	True	False	38.403	1207037.7
8397	$\overline{\mathbf{e}}, \overline{\mathbf{d}}, \overline{\mathbf{p}}, \overline{\mathbf{s}}, \overline{\mathbf{v}}, \overline{\mathbf{t}}, \overline{\boldsymbol{\lambda}}$	True	False	38.427	1207037.7
8398	$\overline{\mathbf{e}}, \overline{\mathbf{d}}, \overline{\mathbf{p}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{s}}, \overline{\mathbf{v}}, \overline{\mathbf{t}}$	True	False	38.439	1207037.7
8399	$\overline{\mathbf{e}}, \overline{\mathbf{s}}, \overline{\mathbf{v}}, \overline{\mathbf{p}}, \overline{\mathbf{t}}, \overline{\mathbf{d}}, \overline{\boldsymbol{\lambda}}$	True	False	38.454	1207037.7
8400	$\overline{\boldsymbol{\lambda}}, \overline{\mathbf{d}}, \overline{\mathbf{e}}, \overline{\mathbf{s}}, \overline{\mathbf{t}}, \overline{\mathbf{v}}, \overline{\mathbf{p}}$	False	False	38.532	1207037.7
...					
20156	$\overline{\mathbf{v}}, \overline{\mathbf{t}}, \overline{\mathbf{s}}, \overline{\mathbf{p}}, \overline{\mathbf{d}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{e}}$	True	True	60.000	—
20157	$\overline{\mathbf{v}}, \overline{\mathbf{t}}, \overline{\mathbf{s}}, \overline{\mathbf{p}}, \overline{\mathbf{e}}, \overline{\mathbf{d}}, \overline{\boldsymbol{\lambda}}$	True	True	60.000	—
21058	$\overline{\mathbf{v}}, \overline{\mathbf{t}}, \overline{\mathbf{s}}, \overline{\mathbf{p}}, \overline{\mathbf{e}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{d}}$	True	True	60.000	—
21059	$\overline{\mathbf{v}}, \overline{\mathbf{t}}, \overline{\mathbf{s}}, \overline{\mathbf{p}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{d}}, \overline{\mathbf{e}}$	True	True	60.000	—
21060	$\overline{\mathbf{v}}, \overline{\mathbf{t}}, \overline{\mathbf{s}}, \overline{\mathbf{p}}, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{e}}, \overline{\mathbf{d}}$	True	True	60.000	—

Table 2: More details of specific variable orderings.

and final size for the default dynamic reordering heuristics in CUDD, initialized with the fastest variable ordering, is shown in Figure 4 using two triangles. Here, it can be seen that the final size of the BDD is very small compared with the other variable orderings. However, we obtain an order of magnitude speed up by selecting our best ordering, hence, we use the order from row 1 in Table 2 for the subsequent experiments.

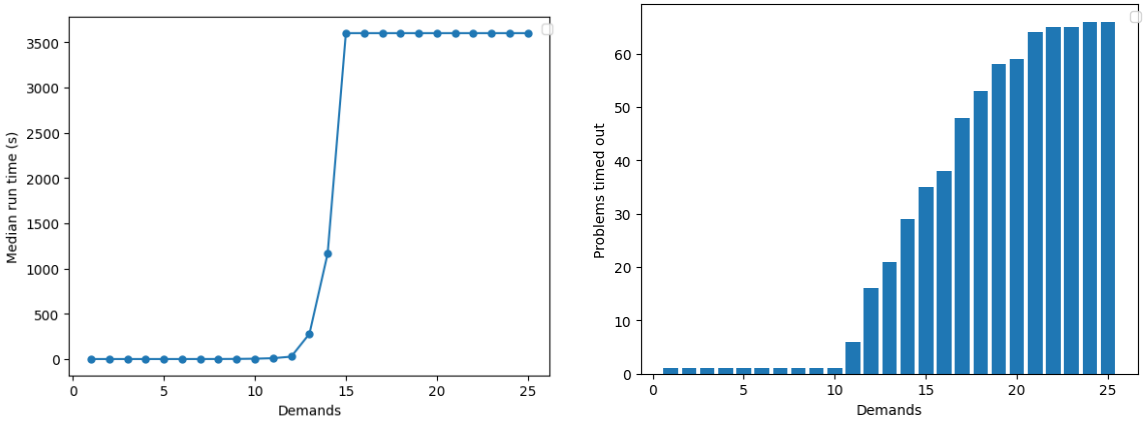
Scaling of Baseline on Number of Wavelengths For these experiments, the BDD construction run time is measured on a selection of 68 networks from topology zoo. These networks are selected based on preliminary testing, where they are able to be solved with 10 demands and 10 wavelengths. Using these networks, the number of demands is fixed to 15 with a range of 1-16 wavelengths. In



(a) Median run time as a function of number of wavelengths for 15 demands. Timeout is 3600 seconds.

(b) Number of problems timed out for each number of wavelengths.

Fig. 5: Effect of number of wavelengths on run time.



(a) Median run time as a function of demands using 8 wavelengths. For problems that timeout, the run time is set to 3600 seconds.

(b) Number of problems timed out for each number of demands.

Fig. 6: Effect of number of demands on run time.

Figure 5a the median run time is plotted as a function of the wavelengths. As seen, the scaling is exponential, and beyond 7 wavelengths, more than half of the networks time out. In Figure 5b, the number of timed out networks is shown for each wavelength. The results show that the number of timed out networks increases steadily with the number of wavelengths. These trends are expected, as the number of solutions increases with the number of wavelengths.

Scaling of Baseline on Number of Demands Based on the previous results, we now fix the number of wavelengths to 8, as a compromise between run time and agency to avoid wavelength clashing. The results are presented in Figure 6 over a range of 1-25 demands. In Figure 6a, the median run time scales exponentially, and after 16 demands, more than half the networks exceed the timeout of 3600 seconds. Figure 6b shows the number of networks that timed out for each demand. The result shows that most timeouts occur only when the problem includes more than ten demands.

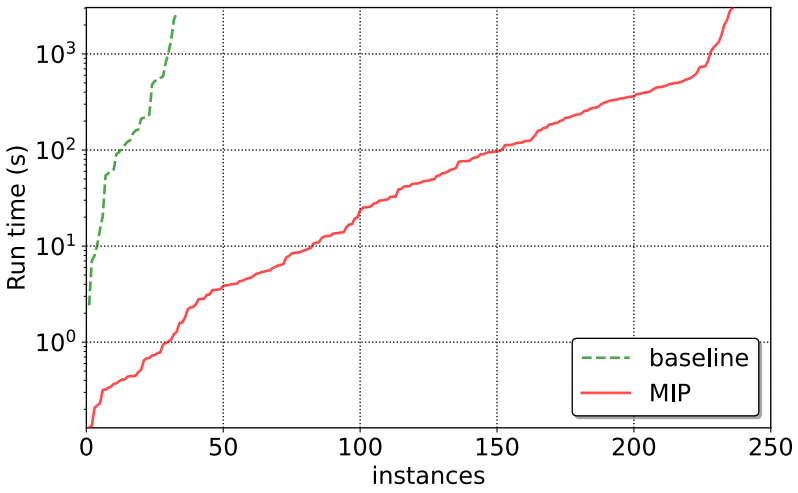


Fig. 7: Cactus plot of baseline compared to MIP for 15 demands, 8 wavelengths, and timeout of 3600 seconds.

5.2 Comparison to State of the Art

We compare the baseline to an implementation of MIP that uses the package Pulp [17] for python based on [3], which is also guaranteed to find optimal solutions. This method, however, only finds a singular solution in contrast to AllRight, which finds all solutions. Our implementation of MIP can also be found with our source code in [8]. To evaluate their effectiveness, we make *cactus-plots* [13, 18] to get an overview of the scalability of each method over a range of network instances from topology zoo. To make the plots, for each method, we sort the order of the instances on the x-axis by the run time from fastest to slowest, and the y-axis is plotted on a logarithmic scale. If the run time for any remaining instances is not shown, the method cannot solve the RWA problem within the allotted timeout. Both experiments are run on the 261 topologies from topology zoo with 15 demands, 8 wavelengths, and a timeout of 3600 seconds.

The results are shown in Figure 7. It can be seen, that MIP is able to solve significantly more of the problems and is three orders of magnitude faster for the most difficult problems solved by the baseline. This, clearly, leaves room for improvement.

6 Improvements to the Baseline

Since a comparison between the baseline and the state of the art shows a clear room for improvement, we implement and evaluate different approaches through which we expect the run time can be improved. The approaches are as follows:

- *Edge encoding*: An alternative encoding of the RWA problem, which encodes the paths differently in order to scale better on the number of demands.
- *Incremental*: Finding only the optimal solutions by incrementing wavelengths until at least one solution is found.
- *Pruning equivalent wavelength assignments*: Pruning equivalent wavelength assignments by limiting the set of possible assignable wavelengths for each demand.
- *Divide and Conquer*: Dividing the RWA problem into individually solved sub problems, solving these and then combining the solutions.

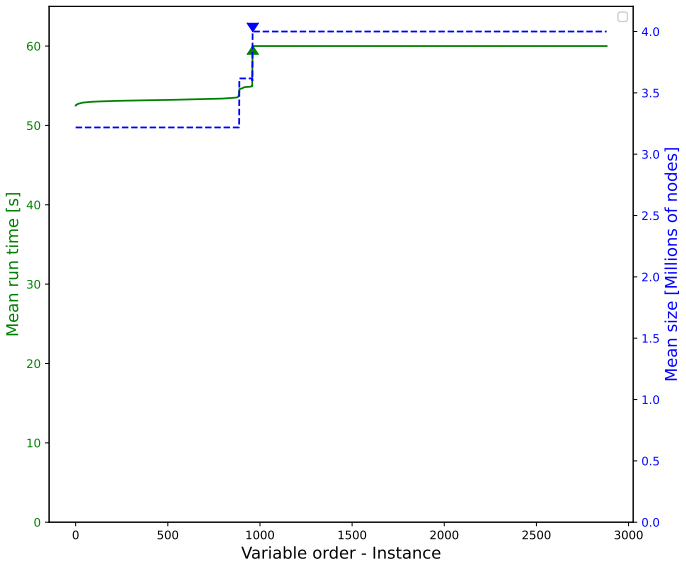


Fig. 8: Comparison of variable orderings for *edge encoding*. Variable orderings are ordered first by the build time then by the final size. The two triangles mark the values for the default heuristics used by CUDD.

6.1 Alternative Demand Path Encoding

In the baseline, the number of path variables $\overline{\mathbf{p}}$ scales linearly in the number of demands, as they are unary encoded. We now consider an alternative encoding of the $\overline{\mathbf{p}}$ variables to make the $\overline{\mathbf{p}}$ variables scale better with respect to demands.

The alternative encoding, called *edge encoding*, encodes the path of a demand directly onto the edges using binary encoding. The tradeoff is that wavelengths must now be unary encoded onto the edges as well. This means that, for each edge and wavelength combination, we have variables $\overline{\mathbf{p}_e^\lambda} = [\mathbf{p}_0^\lambda, \mathbf{p}_1^\lambda, ..., \mathbf{p}_{\lceil \log_2(|D|)\rceil - 1}^\lambda]$ such that $d \in [\![\overline{\mathbf{p}_e^\lambda}(d)]\!]$. The idea is that the $\overline{\mathbf{p}}$ variables now scale logarithmically in the number of demands.

Another benefit of this encoding is that an edge can at most encode one demand for each possible wavelength, and hence, checking for wavelength clashes between demands is no longer needed. The complete formalisation for *edge encoding* can be found in Appendix 1.

Results of *edge encoding* The alternative edge encoding is implemented using CUDD based on the formalism in Appendix 1. In Figure 8, the results are shown for a variable ordering analysis like the one performed on the baseline. It can be seen that the best variable orderings for *edge encoding* leads to much longer run times and larger sizes of the final BDD, than the best variable orderings for the baseline. Furthermore, it can be seen that the run using the default heuristics in CUDD is not able to complete before the timeout.

It can thus be concluded that despite the better demand scaling, for this experimental set up, the tradeoff is not beneficial enough. In Figure 9, we compare the number of variables needed to encode the demand paths for the baseline and *edge encoding* approach by plotting the number of $\overline{\mathbf{p}}$ variables for each approach in a network with 8 wavelengths and 40 edges. As shown, *edge encoding* first begins to use fewer variables beyond 40 demands, however, in practice, only problems up to 15 demands are solvable within the timeout. As such, the benefit of using *edge encoding* is never reached within the plausible number of demands.

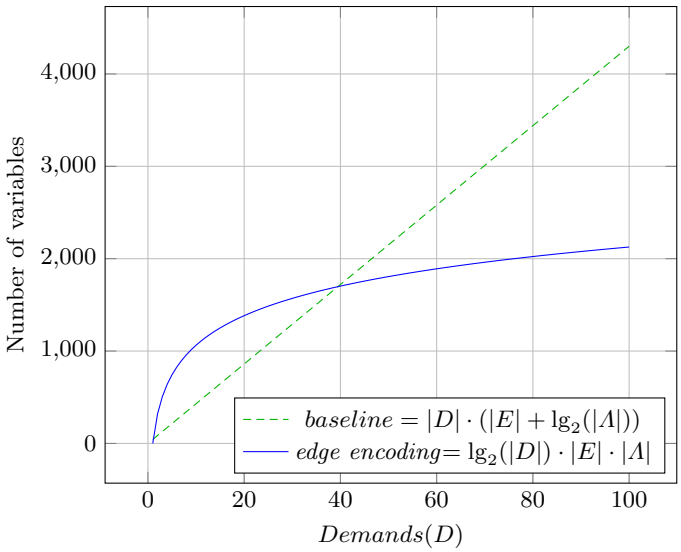


Fig. 9: This graph illustrates the relationship between the number of variables in the final BDD and demands (D), considering a graph with 40 edges (E) and 8 wavelengths (Λ).

6.2 Incremental Solve

The BDD *rwa* finds all network optimal solutions by finding all solutions to the RWA problem. Hence, if we are only interested in the network optimal solutions, non-optimal solutions must be pruned as is done using Algorithm 1. A different approach is to only encode network optimal solutions to begin with, when constructing *rwa*. Since this approach avoids computing solutions that are not network optimal, it can reduce the computation time of *rwa*. This is achieved by following an approach similar to the one presented in Algorithm 1, as described in Algorithm 2. The main difference is that Algorithm 2 iteratively computes *rwa*, incrementing the maximum number of wavelengths the BDD *rwa* may use for each iteration, instead of pruning solutions encoded by an already computed *rwa*. The algorithm takes a network topology G, demands D and wavelengths Λ as input, and outputs a BDD which encodes all network optimal solutions to the given RWA problem.

Algorithm 2 Incremental approach to solving the RWA problem

Input: Network topology G, demands D, wavelengths Λ.
Output: BDD *rwa-inc* encoding only network optimal solutions.

1: **procedure** IncrementalRWA(G, D, Λ)
2: Let *rwa-inc* be a BDD.
3: *rwa-inc* $\leftarrow$ *false*
4: $\ell \leftarrow 1$
5: **while** $[\![$*rwa-inc*$]\!] = \varnothing$ and $\ell \leq |\Lambda|$ **do**
6: Let $\Lambda' \subseteq \Lambda$ such that $|\Lambda'| = \ell$.
7: Compute *rwa* for the RWA problem with G, D and Λ' as input.
8: *rwa-inc* $\leftarrow$ *rwa*
9: $\ell \leftarrow \ell + 1$ ▷ Allow one more wavelength for next iteration.
10: **end while**
11: **return** *rwa-inc*
12: **end procedure**

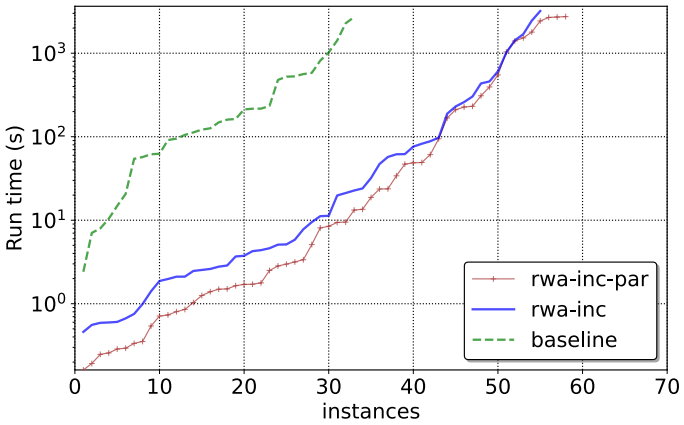


Fig. 10: *rwa-inc* and *rwa-inc-par* compared to baseline for 15 demands, 8 wavelength and timeout of 1 hour.

Lines 2-5 follow the same structure as Algorithm 1. Lines 6-9 iteratively construct an *rwa* BDD that encodes all solutions to the RWA problem for the network topology G, demands D and wavelengths $\Lambda' \subseteq \Lambda$, starting from $|\Lambda'| = 1$ and incrementing the size of Λ' by 1 for each iteration. The computed *rwa* is assigned to the BDD *rwa-inc* on line 8. If *rwa-inc* encodes at least one solution after an iteration, the loop terminates, and *rwa-inc* is returned, encoding all network optimal solutions. For example, if the loop terminates when $\ell = 3$, it means that the minimum number of wavelengths needed to solve the RWA problem for G, D and Λ is 3, as no solutions were found in the previous iterations. Otherwise, the loop terminates due to ℓ exceeding the total number of wavelengths, $|\Lambda|$, in which case there were no solutions to the given RWA problem.

Theorem 3. *Let the BDD* *rwa-inc* *be the output of Algorithm 2 given the inputs* G, D *and* Λ*. Then, a solution* (P, Ω) *to the RWA problem defined by* (G, D, Λ) *is a network optimal solution iff* $(P, \Omega) \in [\![\textit{rwa-inc}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$.

Proof. Obvious from analysis of Algorithm 2. □

It should be noted that while Algorithm 2 is defined in an consecutive manner, where each ℓ is checked one after another, the algorithm can easily be transformed into a parallel algorithm where lines 6-8 are run in parallel for all $1 \leq \ell \leq |\Lambda|$. The computation of all parallel jobs can then be stopped, when it is known for certain that the result of one of the completed jobs corresponds to the optimal solutions.

Results of Incremental A cactus plot for 15 demands and 8 wavelengths is presented in Figure 10, comparing *incremental* (*rwa-inc*) with the baseline. Clearly, *rwa-inc* is an improvement on the baseline, as it is able to solve more problems, and is consistently faster for all problems. This is expected, as most of the problems are solvable with less than 8 wavelengths, and it can be seen in Figure 5a on page 14 that the run time is significantly faster for fewer wavelengths. Furthermore, we see that the parallelized version of *rwa-inc*, which we refer to as *rwa-inc-par* in Figure 10, is consistently faster than both *rwa-inc* and the baseline.

6.3 Pruning Equivalent Solutions

In most cases, the baseline finds solutions to the RWA problem that can be considered equivalent. For example, if there are two demands and two wavelengths in a network, there are four different wavelength assignments in total. However, the four assignments can be split into two sets of equivalent solutions. Either both demands are assigned the same wavelength, or they are assigned different wavelengths. Computing these equivalent solutions is not necessary, and hence, pruning methods can potentially improve the solve time. A method of doing this is finding equivalence classes. An equivalence class represents a set of solutions that have similar behaviour. Following this approach, we can define each wavelength assignment in terms of a set of equivalence classes, and then perform pruning by selecting exactly one solution from each equivalence class.

Definition 14 (Wavelength Equivalent RWA Solutions). *Two solutions* (P_1, Ω_1) *and* (P_2, Ω_2) *to the RWA problem are* equivalent *if there exists a bijection* $\sigma : \Lambda \to \Lambda$ *such that*

$$P_1 = P_2 \text{ and } \sigma(\Omega_1(d)) = \Omega_2(d) \text{ for all } d \in D. \tag{19}$$

The number of equivalence classes can be calculated using the Stirling numbers, as it relates to how many ways a set of n elements can be partitioned into k subsets [19]. This corresponds to partitioning the set of n demands into k subsets, where the demands in each subset are assigned the same wavelength. Thus, the number of equivalence classes with demands D and wavelengths Λ can be calculated as:

$$\sum_{j=1}^{|\Lambda|} \sum_{i=0}^{j} \frac{(-1)^{j-i} i^{|D|}}{(j-i)!i!} \tag{20}$$

As seen in Equation 20, the number of equivalence classes grows exponentially with the number of demands. We are unable to find an efficient method to limit the solution space of the RWA problem to only a single wavelength assignment from each equivalence class. However, we are able to prune some of the equivalent solutions using less computationally expensive methods, when a natural ordering is assumed as $d_0 < d_1 < ... < d_{|D|-1}$ of the demands and $\lambda_0 < \lambda_1 < ... < \lambda_{|\Lambda|-1}$ of the wavelengths. In the following, we present two methods called *Wavelength Limit* and *Sequential*.

Overall, the approach is to limit the number of equivalent solutions by limiting the number of possible wavelength assignments for the demands. For *Wavelength Limit*, we enforce that demand d_i can only be assigned a wavelength λ_k where $k \leq i$. This change is applied to the BDD rwa as follows:

$$\mathit{rwa\text{-}lim}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D}) = \mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D}) \wedge \bigwedge_{d_i \in D} \Big(\bigvee_{k \leq i} \overline{\boldsymbol{\lambda}^{d_i}}(\lambda_k) \Big) \tag{21}$$

By Condition 21, observe that the only difference between the baseline and $\mathit{rwa\text{-}lim}$ is the number of wavelengths that are assignable for a given demand, being more restrictive in $\mathit{rwa\text{-}lim}$. Hence, we have that $[\![\mathit{rwa\text{-}lim}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!] \subseteq [\![\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$. We now show that $\mathit{rwa\text{-}lim}$ is safe and only prunes some equivalent solutions and never all equivalent solutions.

Theorem 4. *If* $(P, \Omega) \in [\![\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$ *then there exists a solution* $(P, \Omega') \in [\![\mathit{rwa\text{-}lim}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$ *s.t.* (P, Ω) *and* (P, Ω') *are equivalent.*

Proof. We construct a bijection σ. Let $(P, \Omega) \in [\![\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$, and let σ initially be undefined. Assume a natural ordering of demands and wavelengths. Then, for any step $0 \leq i < |D|$, if $\Omega(d_i) = \lambda$ and $\sigma(\lambda)$ is undefined, we define $\sigma(\lambda) = \lambda_i$. For any remaining wavelengths not defined in σ, we map each of these to a wavelength not yet mapped to by other wavelengths. This is always possible as σ maps from and to the same set. It is clear from this construction that σ is a bijection, and hence, the solutions (P, Ω) and $(P, \sigma \circ \Omega)$ are equivalent. Furthermore, $\sigma \circ \Omega$ must satisfy Condition 21, as any demand d_i is at most assigned a wavelength λ_i. Hence $(P, \sigma \circ \Omega) \in [\![\mathit{rwa\text{-}lim}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$. □

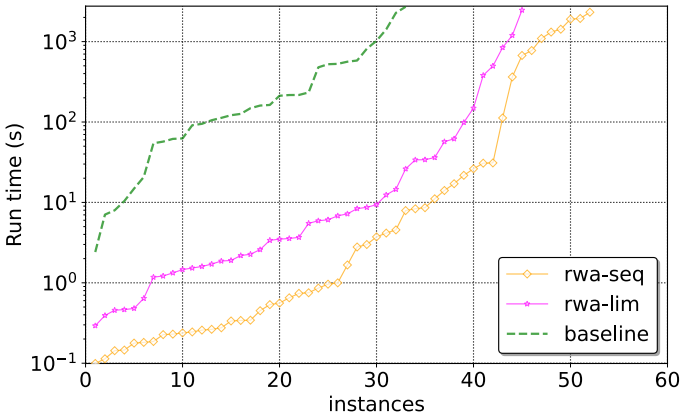


Fig. 11: *rwa-seq* and *rwa-lim* compared to baseline for 15 demands, 8 wavelength and timeout of 1 hour.

Another way to limit wavelength assignments is by removing the gaps allowed in *rwa-lim*. An example of a gap could be that demands d_1 and d_2 are assigned wavelength λ_1 but then demand d_3 is assigned wavelength λ_3 instead of λ_2. For the *Sequential* method, we define the BDD *rwa-seq* to remove such gaps.

$$\textit{rwa-seq}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D}) = \textit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D}) \wedge \bigwedge_{\lambda_i \in \Lambda \setminus \{\lambda_0\}} \left(\exists \overline{\boldsymbol{\lambda}^{d_j}}. \overline{\boldsymbol{\lambda}^{d_j}}(\lambda_i) \implies \exists \overline{\boldsymbol{\lambda}^{d_k}}. \overline{\boldsymbol{\lambda}^{d_k}}(\lambda_{i-1}) \wedge k < j \right) \quad (22)$$

Theorem 5. *If* $(P, \Omega) \in [\![\textit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$ *then there exists a solution* $(P, \Omega') \in [\![\textit{rwa-seq}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$ *s.t.* (P, Ω) *and* (P, Ω') *are equivalent.*

Proof. We modify the construction of the bijection σ described in Theorem 4. We initially define σ with one mapping, namely $\sigma(\Omega(d_0)) = \lambda_0$. For any step $0 < i < |D|$, if $\Omega(d_i) = \lambda$ and $\sigma(\lambda)$ is undefined, we define that $\sigma(\lambda) = \lambda_k$ where $k = 1 + max\{k' \mid \lambda_{k'} \in \sigma(\Lambda)\}$. Based on this construction of σ, the same arguments from Theorem 4 follow. Hence $(P, \sigma \circ \Omega) \in [\![\textit{rwa-seq}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})]\!]$. □

Comparison of *Wavelength Limit* and *Sequential* Both *Wavelength Limit* and *Sequential* are implemented by replacing the wavelength assignment in Condition 17 of *rwa* on page 10 with the wavelength constraint imposed by *rwa-lim* and *rwa-seq* respectively. In Figure 11, a cactus plot is shown for 15 demands and 8 wavelengths comparing *rwa-seq* and *rwa-lim* to the baseline. Both of the methods are faster than the baseline and solve more problems, which indicates that the equivalent solutions are pruned with negligible overhead. The computation of *rwa-seq* is also faster than for *rwa-lim*, indicating that although *rwa-seq* has a larger pruning-overhead, the time is gained back in having to find less equivalent solutions.

6.4 Parallelisation of BDD Computation

In the aforementioned methods, a single BDD is responsible for computing the paths and wavelength assignments for all demands. We now consider how to parallelize this process by splitting the set of demands into separate RWA subproblems. The approach entails solving the RWA problem for each partition of demands and afterwards combining the solutions to obtain the solutions for the original problem.

Formally, given two partitions of an RWA problem (G, D, Λ) and (G, D', Λ), let the BDDs $\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})$ and $\mathit{rwa'}(\overline{\mathbf{p}^{D'}}, \overline{\boldsymbol{\lambda}^{D'}})$ represent the solutions to the respective subproblems. Then, the combined solution to the problem defined by $(G, D \cup D', \Lambda)$ can be found by conjunction of rwa and $\mathit{rwa'}$, and by enforcing that the demands encoded in the two BDDs do not wavelength clash:

$$\mathit{rwa\text{-}conq}(\overline{\mathbf{p}^{D\cup D'}}, \overline{\boldsymbol{\lambda}^{D\cup D'}}) = \mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D}) \wedge \mathit{rwa'}(\overline{\mathbf{p}^{D'}}, \overline{\boldsymbol{\lambda}^{D'}}) \wedge \tag{23}$$

$$\bigwedge_{d\in D, d'\in D'} \exists \overline{\mathbf{d}}, \overline{\mathbf{d'}} \,.\, \overline{\mathbf{d}}(d) \wedge \overline{\mathbf{d'}}(d') \wedge \mathit{noClash}(\overline{\mathbf{d}}, \overline{\mathbf{d'}}, \overline{\mathbf{p}^d}, \overline{\mathbf{p}^{d'}}, \overline{\boldsymbol{\lambda}^d}, \overline{\boldsymbol{\lambda}^{d'}}) \tag{24}$$

Theorem 6. *Let* $\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})$ *represent all solutions of the RWA problem defined by* (G, D, Λ) *and let* $\mathit{rwa'}(\overline{\mathbf{p}^{D'}}, \overline{\boldsymbol{\lambda}^{D'}})$ *represent all solutions of the RWA problem defined by* (G, D', Λ)*. Then,* $(P, \Omega) \in [\![\mathit{rwa\text{-}conq}(\overline{\mathbf{p}^{D\cup D'}}, \overline{\boldsymbol{\lambda}^{D\cup D'}})]\!]$ *iff* (P, Ω) *is a valid solution to the RWA problem defined by* $(G, D \cup D', \Lambda)$

Proof. For the left to right implication, assume $(P, \Omega) \in [\![\mathit{rwa\text{-}conq}(\overline{\mathbf{p}^{D\cup D'}}, \overline{\boldsymbol{\lambda}^{D\cup D'}})]\!]$. Then, we know from Constraint 23 that P assigns a path and Ω assigns a wavelength to all demands in $(D \cup D')$. From Constraint 24 and the definition of rwa, we know that there is no wavelength clashing for all demands in the solution (P, Ω). Hence, (P, Ω) is a valid solution to the RWA problem $(G, D \cup D', \Lambda)$.

For the right to left implication, assume (P, Ω) is a valid solution to the RWA problem defined by $(G, D \cup D', \Lambda)$. Since (P, Ω) is a valid solution, none of the demands wavelength clash, ensuring that Constraint 24 is satisfied. Additionally, since (P, Ω) is a valid solution, we can split it into two valid solutions (P^D, Ω^D) and $(P^{D'}, \Omega^{D'})$ such that $(P^D, \Omega^D) \in [\![\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\Lambda}^D})]\!]$ and $(P^{D'}, \Omega^{D'}) \in [\![\mathit{rwa'}(\overline{\mathbf{p}^{D'}}, \overline{\boldsymbol{\Lambda}^{D'}})]\!]$, satisfying Constraint 23. Hence, $(P, \Omega) \in [\![\mathit{rwa\text{-}conq}(\overline{\mathbf{p}^{D\cup D'}}, \overline{\boldsymbol{\lambda}^{D\cup D'}})]\!]$.

□

Implementation and Results of *Divide and Conquer* In the computation of $\mathit{rwa\text{-}conq}$, it is possible to parallelise the computations of the two BDDs rwa and $\mathit{rwa'}$. In fact, it is possible to fully parallelise the computation of $\mathit{rwa\text{-}conq}$ following a divide-and-conquer approach in a method denoted by $\mathit{rwa\text{-}conq\text{-}par}$. In this method, the RWA problem is split into $|D|$ subproblems for maximum parallelisation. After splitting, the subproblems are combined in pairs, building solutions that in turn need to be combined with other solutions. In effect, this builds a binary tree of problems of height $log_2(|D|)$. The method terminates when the root is built, signifying that the solutions for all demands are combined. Hence, the total run time is the sum of the maximium run time at each level of the tree, since the computations at each level are run in parallel.

In Figure 12, a cactus plot is shown comparing $\mathit{rwa\text{-}conq\text{-}par}$ with the baseline. Clearly, the method outperforms the baseline on all instances and is able to solve more problems. This indicates that the overhead incurred in parallelizing the BDD computation is negligible in comparison to the speed up.

Dynamic RWA Problem While we are only considering the static RWA problem in this paper, we note that $\mathit{rwa\text{-}conq}$ can also be used to solve a limited version of the dynamic RWA problem in which demands can only be added to the network. Assuming a new set of demands needs to be added to an existing set, in Equation 23, rwa can represent the previously computed BDD for a given set of demands and $\mathit{rwa'}$ can represent a BDD containing the solutions for a new set of demands. The combination of the two BDDs can then be performed as defined by $\mathit{rwa\text{-}conq}$.

The effects of this on the run time can be seen in Figure 13 where a single demand is added to an rwa that encodes all the solutions to an RWA problem for 14 demands and 8 wavelengths. Specifically, we have the two BDDs $\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})$ and $\mathit{rwa'}(\overline{\mathbf{p}^{D'}}, \overline{\boldsymbol{\lambda}^{D'}})$ where $|D| = 14$ and $|D'| = 1$

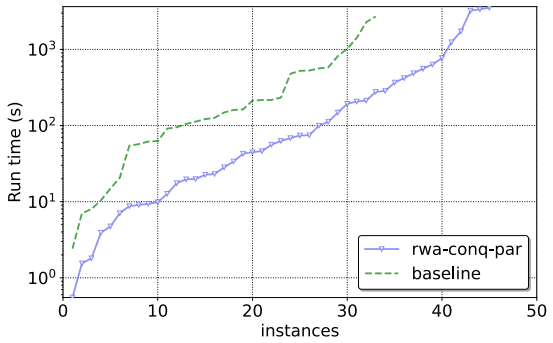


Fig. 12: *rwa-conq-par* vs baseline for 15 demands, 8 wavelengths, and 1-hour timeout.

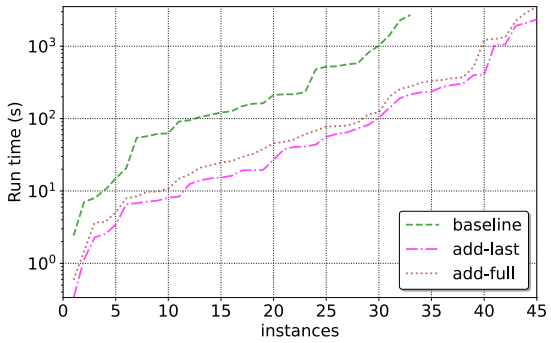


Fig. 13: Effect of adding the last of 15 demands with 8 wavelengths, and 1-hour timeout.

and use these to compute *rwa-conq*. We refer to the full run time of *rwa-conq* as *add-full* and the run time for adding the single demand as *add-last*, and compare it to the run time of computing *rwa* using the baseline for 15 demands. From *add-full*, it can be seen that decomposing the RWA problem into two sub-problems of 14 demands and 1 demand, and then combining these, yields an improvement compared to running the baseline on the full problem. However, from comparing *add-full* and *add-last*, it can be seen that there is only a small difference between the run times. Hence, the positive gain of having a precomputed version of the BDD $\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})$ is negligible.

We now consider why it is not possible to compute the BDD $\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})$ by removing the solutions for a set of demands D' from a BDD $\mathit{rwa\text{-}conq}(\overline{\mathbf{p}^{D\cup D'}}, \overline{\boldsymbol{\lambda}^{D\cup D'}})$ without recomputing the BDD $\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})$ as part of the operation. Here, we remark that it is generally impossible to create a Boolean expression C using only terms B and $(A \wedge B)$ such that $C \equiv A$ when $\neg(A \equiv B)$. This can be seen when $B = False$, since then $(A \wedge B) = False$ no matter if $A = True$ or $A = False$ and thus C will have the same truth value for both truth values of A.

Thus, if we let $\mathit{rwa}(\overline{\mathbf{p}^D}, \overline{\boldsymbol{\lambda}^D})$ be the Boolean expression A and let

$$\mathit{rwa'}(\overline{\mathbf{p}^{D'}}, \overline{\boldsymbol{\lambda}^{D'}}) \wedge \bigwedge_{d\in D, d'\in D'} \exists \overline{\mathbf{d}}, \overline{\mathbf{d'}} \,.\, \overline{\mathbf{d}}(d) \wedge \overline{\mathbf{d'}}(d') \wedge \mathit{noClash}(\overline{\mathbf{d}}, \overline{\mathbf{d'}}, \overline{\mathbf{p}^d}, \overline{\mathbf{p}^{d'}}, \overline{\boldsymbol{\lambda}^d}, \overline{\boldsymbol{\lambda}^{d'}})$$

be the Boolean expression B, it follows directly that the removal of demands from the network will always include re-solving the RWA problem for the updated set of demands.

6.5 Combination of Approaches

In the previous sections, we have described different approaches to improve the run time of the baseline. The approaches that yield the best results are *rwa-seq* and the parallelized *rwa-inc* (*rwa-inc-par*). Specifically, the BDD *rwa-seq* give the best speedup, while the *rwa-inc-par* solve more instances than any other approach. As such, we combine these two approaches. We refer to this combined approach as the BDD *rwa-inc-par-seq*. Additionally, we also combine the parallelized *divide-and-conquer* (*rwa-conq-par*) and the parallel incremental approach *rwa-inc-par* with the *rwa-lim* which we will refer to as *rwa-conq-inc-par-lim*. The reason for choosing *rwa-lim* instead of *rwa-seq* is that *rwa-seq* and *rwa-conq* cannot be combined, since the constraints enforced by *rwa-seq* can lead to scenarios where the optimal solutions are not found. The results can be seen on Figure 14 where we compare the two combined approaches to the approaches from the previous sections as well as the baseline. It is clear from Figure 14 that *rwa-inc-par-seq* is more efficient than the other approaches and solves just as many instances as *rwa-inc-par*. We also see that

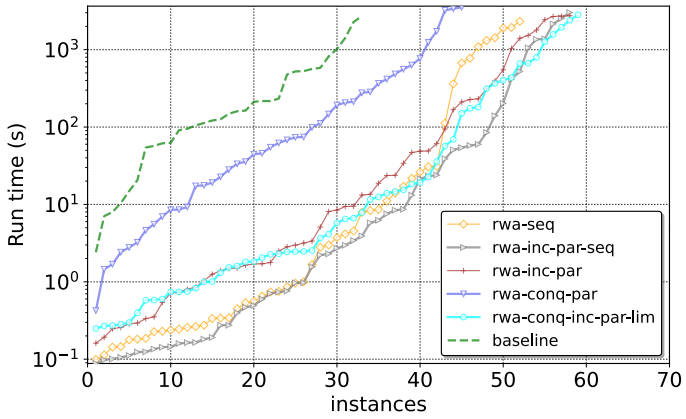


Fig. 14: Previous approaches compared to the combined approaches *rwa-inc-par-seq* and *rwa-conq-inc-par-lim* when solving the RWA problem for 15 demands and 8 wavelengths with an 1-hour timeout.

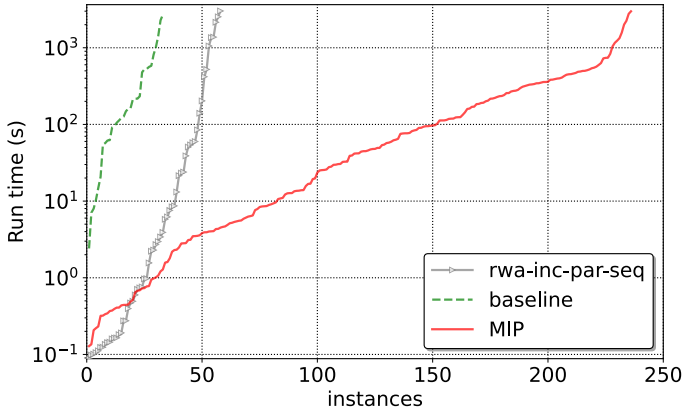


Fig. 15: Baseline and *rwa-inc-par-seq* compared to MIP when solving the RWA problem for 15 demands and 8 wavelengths with an 1-hour timeout.

rwa-conq-inc-par-lim is faster than *rwa-seq* after the 40th instance and is able to solve 1 more instances than any other approach. Since *rwa-inc-par-seq* is generally the most efficient approach, and *rwa-conq-inc-par-lim* only solves one more instance, we conclude that our best approach is *rwa-inc-par-seq*.

Lastly, comparing *rwa-inc-par-seq* to the state-of-the-art method MIP on Figure 15, the combined approach is faster than MIP for the first 25 instances despite finding all network optimal solutions as opposed to only one, which indicates that it has a smaller overhead. But, we also see that it does not scale as well as MIP.

7 Conclusion

We presented the tool AllRight to solve the RWA problem using a BDD based approach that provably finds all possible solutions to the RWA problem and represents them compactly using BDDs. Because AllRight finds all solutions, it differs from other tools in the domain of solving the RWA problem as they find only one solution. Since AllRight finds all solutions, we described an approach to find just the network optimal ones as well. Additionally, we showed different techniques to improve the run time by changing the variable ordering used when constructing the BDDs, reducing the number of equivalent solutions, parallelizing the computation of the BDDs, as well as restricting the solution space to only network optimal solutions. From our experiments, we found that the fastest run times are seen when combining multiple of these techniques. It can thus be concluded, that compared to a state-of-the-art method, AllRight finds all solutions at the cost of an increased run time.

8 Future Work

In this section, we propose different ways to extend and possibly improve the work we have presented in this paper. The section is split into two parts. In the first part, we focus on improving the BDDs with respect to solving the RWA problem. In the second part, we discuss variations of the RWA problem to which this work also can be applied.

8.1 Improve BDDs

A variety techniques were described to improve the BDDs, as seen in Section 6. We now discuss other potential approaches to improving the efficiency of construction the BDDs.

Pruning As described in Section 6.3, we attempted to reduce the number of equivalent RWA solutions by using the BDDs *rwa-lim* and *rwa-seq*. From the results, we know that both BDDs yielded a faster run time compared to the baseline. Thus, we expect that defining a BDD that efficiently prunes even more equivalent solutions can reduce the run time even further. We were, however, unable to find an efficient pruning method that removed more equivalent solutions than *rwa-seq* without introducing too much overhead in the pruning itself.

Edge encoded demands We described a different way to encode demands in Section 6.1, such that the number of $\overline{\mathbf{p}}$ variables scale logarithmically with the demands. If it becomes possible to handle more demands using our approach, then it seems that there is a lot to be gained from using the edge encoding, as seen in Figure 9 from Section 6.1. It would therefore be interesting to explore if any of the other techniques can also be applied to this alternative encoding in order to increase the feasible number of demands.

Binary encode edges in paths Currently, a path is represented in the BDDs by a unary set encoding of the edges. However, the edges used in a path can be binary encoded instead. We know from experiments that fewer variables yield faster computation time. If a given network topology has $|E|$ edges, we can represent any edge with $log_2(|E|)$ variables. Hence, for a given path using n edges, if $n < \lceil \frac{|E|}{\lceil log_2(|E|) \rceil} \rceil$, then the binary set encoding needs fewer variables to represent the path than the unary set encoding.

Sub optimal solutions A different approach is to no longer guarantee that our BDDs encode optimal solutions in favor of efficiency. This can be achieved by simplifying the RWA problem. There exists a range of different approaches to simplifying the RWA problem:

- **Decompose the RWA problem**: As described in [2], a classic approach is to first solve the routing assignment problem, and then solve the wavelength assignment problem. The routing assignment corresponds to building the BDD *demandPath* and choosing one solution from *demandPath* for each demand such that they combine to give an optimal routing. Then, the wavelength assignment corresponds to building *rwa* with only the chosen path for each demand.
- **Split the RWA problem into independent parts:** Following the approach presented in [20], we can split the given RWA problem into smaller, independent problems, each having a subgraph of the original graph, and a subset of the demands and wavelengths from the original RWA problem.
- **Fixed paths**: Our approach finds all possible simple paths for the demands. However, many of these are not necessarily of interest. In practice, often only a limited number of shortest paths w.r.t distance are considered for each demand. Hence, we could define a specific number of paths for each demand instead of considering all possible simple paths for each demand.

8.2 Extension of RWA Problem.

There are many different aspects to the general RWA problem that our formalization does not capture, where BDDs could be used. We discuss some of these aspects that can extend the RWA problem we described in Section 3.

Wavelength conversion As described in [21], some WDM optical fiber networks use Reconfigurable Optical Add/Drop Multiplexers (ROADM). A ROADM is a technology that can change the wavelength of a demand as it flows through a router in the network.

Large demands We have assumed that link congestion never occurs when routing the demands. However, in a real world application, this must be accounted for as well. Some demands have such a large throughput that they must be transmitted over multiple wavelengths as they do not fit one channel in a link.

Multi-core fibers The bandwidth of a WDM optical fiber network can be increased even further using multi-core fibers, which is a Space Division Multiplexer (SDM) technology [22]. For example, a single link can have four cores, each of which having some number of wavelength channels. The problem that arises from this is that neighboring cores using the same wavelengths can still cause interference between each other, due to cross-talking [23]. Hence, the RWA problem is extended to also assigning a core to each demand.

References

[1] Gerd E Keiser. “A review of WDM technology and applications”. In: *Optical Fiber Technology* 5.1 (1999), pp. 3–39.

[2] Hui Zang, Jason P Jue, Biswanath Mukherjee, et al. “A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks”. In: *Optical networks magazine* 1.1 (2000), pp. 47–60.

[3] Helmut Simonis. “Solving the static design routing and wavelength assignment problem”. In: *International Workshop on Constraint Solving and Constraint Logic Programming*. Springer. 2009, pp. 59–75.

[4] Nicola Di Cicco et al. "On deep reinforcement learning for static routing and wavelength assignment". In: *IEEE Journal of Selected Topics in Quantum Electronics* 28.4: Mach. Learn. in Photon. Commun. and Meas. Syst. (2022), pp. 1–12.

[5] Amiyne Zakouni, Jiawei Luo, and Fouad Kharroubi. "Genetic algorithm and tabu search algorithm for solving the static manycast RWA problem in optical networks". In: *Journal of combinatorial optimization* 33 (2017), pp. 726–741.

[6] Henrik Reif Andersen. "An introduction to binary decision diagrams". In: *Lecture notes, available online, IT University of Copenhagen* 5 (1997).

[7] Simon Knight et al. "The internet topology zoo". In: *IEEE Journal on Selected Areas in Communications* 29.9 (2011), pp. 1765–1775.

[8] Hansen Hyldgaard Sneholm and Hebsgaard. *AllRight source code*. `https://github.com/Hebbe1234/p9` [Accessed: 2024]. 2023.

[9] Imrich Chlamtac, Aura Ganz, and Gadi Karmi. "Lightpath communications: An approach to high bandwidth optical WAN's". In: *IEEE transactions on communications* 40.7 (1992), pp. 1171–1182.

[10] Rajiv Ramaswami and Kumar N Sivarajan. "Routing and wavelength assignment in all-optical networks". In: *IEEE/ACM Transactions on networking* 3.5 (1995), pp. 489–500.

[11] Biswanath Mukherjee. "Optical communication networks". In: *(No Title)* (1997).

[12] Ignacio Martín et al. "Machine learning-based routing and wavelength assignment in software-defined optical networks". In: *IEEE Transactions on Network and Service Management* 16.3 (2019), pp. 871–883.

[13] Kim Guldstrand Larsen et al. "AllSynth: Transiently Correct Network Update Synthesis Accounting for Operator Preferences". In: *International Symposium on Theoretical Aspects of Software Engineering*. Springer. 2022, pp. 344–362.

[14] *Python package tulip-control/dd*. `https://github.com/tulip-control/dd?tab=readme-ov-file` [Accessed: 2023]. 2023.

[15] Michael Rice and Sanjay Kulhari. "A survey of static variable ordering heuristics for efficient BDD/MDD construction". In: *University of California, Tech. Rep* (2008), p. 130.

[16] B. Bollig and I. Wegener. "Improving the variable ordering of OBDDs is NP-complete". In: *IEEE Transactions on Computers* 45.9 (1996), pp. 993–1002. DOI: `10.1109/12.537122`.

[17] *PuLP 2.7.0*. `https://pypi.org/project/PuLP/` [Accessed: 2023]. 2022.

[18] Martin Brain, James H Davenport, and Alberto Griggio. "Benchmarking Solvers, SAT-style." In: *SC*^{2} *@ ISSAC*. 2017.

[19] Ronald L Graham et al. "Concrete mathematics: a foundation for computer science". In: *Computers in Physics* 3.5 (1989), pp. 106–107.

[20] Dhritiman Banerjee and Biswanath Mukherjee. "A practical approach for routing and wavelength assignment in large wavelength-routed optical networks". In: *IEEE Journal on selected areas in communications* 14.5 (1996), pp. 903–908.

[21] Mark D Feuer, Daniel C Kilper, and Sheryl L Woodward. "ROADMs and their system applications". In: *Optical Fiber Telecommunications VB*. Elsevier, 2008, pp. 293–343.

[22] Dan M Marom and Miri Blau. "Switching solutions for WDM-SDM optical networks". In: *IEEE Communications Magazine* 53.2 (2015), pp. 60–68.

[23] Mirosław Klinkowski, Piotr Lechowicz, and Krzysztof Walkowiak. "A study on the impact of inter-core crosstalk on SDM network performance". In: *2018 International Conference on Computing, Networking and Communications (ICNC)*. IEEE. 2018, pp. 404–408.

Appendices

1 Edge encoded demands

The goal is still to encode the problem described in Section 3, using BDDs. We encode the network, and demands similar to in the baseline, meaning we reuse the BDDs in, out, source and target. Instead of having a path of edges for each demand, we instead encode a demand directly onto the edges. Specifically, for each edge e, we have $\overline{\mathbf{p}_e^\lambda} = [\mathbf{p}_0^\lambda, \mathbf{p}_1^\lambda, ..., \mathbf{p}_{\lceil \log_2(|D+1|)\rceil - 1}^\lambda]$, such that $d \in [\![\overline{\mathbf{p}_e^\lambda}(d)]\!]$, representing that demand d is routed through edge e with wavelength λ. The demand is therefore binary encoded directly onto the edge, meaning for a given wavelength, an edge can only encode a single demand. We also need to be able to specify, that no demand uses the given edge. The $\overline{\mathbf{p}_\mathbf{e}^{\boldsymbol{\lambda}}} = \overline{\mathbf{ff}}$ therefore corresponds to no demand using the edge with that wavelength, and d_0 is encoded by the binary value 1.

The BDD $\mathit{singleOut}$ has been changed to accommodate the new way of encoding paths, but it still encodes that none of the outgoing edges of a given node encode the same demand.

$$\mathit{singleOut}\,(\overline{\mathbf{v}}, \overline{\mathbf{d}}, \overline{\mathbf{p}^{\boldsymbol{\lambda}}}) = \forall d \in D,\ e, e' \in E\,.\ \exists \overline{\mathbf{e}}, \overline{\mathbf{e'}}. \tag{25}$$

$$\overline{\mathbf{d}}(d) \wedge \overline{\mathbf{e}}(e) \wedge \overline{\mathbf{e'}}(e') \wedge \mathit{out}\,(\overline{\mathbf{v}}, \overline{\mathbf{e}}) \wedge \mathit{out}\,(\overline{\mathbf{v}}, \overline{\mathbf{e'}}) \wedge \overline{\mathbf{p}_e^{\boldsymbol{\lambda}}}(d) \wedge \overline{\mathbf{p}_{e'}^{\boldsymbol{\lambda}}}(d) \implies \overline{\mathbf{e}} = \overline{\mathbf{e'}} \tag{26}$$

In the baseline, path encodes all simple paths from any source node s to any target node t. Since we encode the demand on the edge, path only makes sense in the context of a demand. Hence, we alter the definition of path due to the change in $\overline{\mathbf{p}}$ variables.

Definition 15 (Path). *The BDD* path *is defined as:*

$$\mathit{path}^0(\overline{\mathbf{s}}, \overline{\mathbf{d}}, \overline{\mathbf{p}^{\boldsymbol{\lambda}}}) = \exists d \in D.\big(\overline{\mathbf{d}}(d) \wedge \mathit{target}(\overline{\mathbf{s}}, \overline{\mathbf{d}}) \wedge \bigwedge_{e \in E} \neg \mathbf{p}_\mathbf{e}^\lambda(d)\big) \tag{27}$$

For $k > 0$ *we have*

$$\begin{aligned}
\mathit{path}^k(\overline{\mathbf{s}}, \overline{\mathbf{d}}, \overline{\mathbf{p}^{\boldsymbol{\lambda}}}) \ &= \exists d \in D\,.\overline{\mathbf{d}}(d) \wedge && (28)\\
&\Bigg(\big(\mathit{source}(\overline{\mathbf{s}}, \overline{\mathbf{d}}) \wedge \mathit{path}^{k-1}(\overline{\mathbf{s}}, \overline{\mathbf{d}}, \overline{\mathbf{p}^{\boldsymbol{\lambda}}})\big) \vee && (29)\\
&\Bigg(\exists \overline{\mathbf{v}}, \overline{\mathbf{e}}, \overline{\mathbf{p'^{\boldsymbol{\lambda}}}}.\ \mathit{out}(\overline{\mathbf{s}}, \overline{\mathbf{e}}) \wedge \mathit{in}(\overline{\mathbf{v}}, \overline{\mathbf{e}}) \wedge && (30)\\
&\neg \mathit{source}(\overline{\mathbf{v}}, \overline{\mathbf{d}}) \wedge && (31)\\
&\exists e \in E\,.\,\Big(\overline{\mathbf{e}}(e) \wedge \overline{\mathbf{p}_e^{\boldsymbol{\lambda}}}(d) \wedge \neg \overline{\mathbf{p'^{\boldsymbol{\lambda}}_e}}(d) \wedge \big(\bigwedge_{e' \in E \setminus \{e\}} \overline{\mathbf{p}_{e'}^{\boldsymbol{\lambda}}}(d) = \overline{\mathbf{p'^{\boldsymbol{\lambda}}_{e'}}}(d)\big)\Big) \wedge && (32)\\
&\mathit{path}^{k-1}(\overline{\mathbf{v}}, \overline{\mathbf{d}}, \overline{\mathbf{p'^{\boldsymbol{\lambda}}}}) \wedge && (33)\\
&\neg \mathit{target}(\overline{\mathbf{s}}, \overline{\mathbf{d}}) \wedge \mathit{singleOut}(\overline{\mathbf{s}}, \overline{\mathbf{d}}, \overline{\mathbf{p}^{\boldsymbol{\lambda}}})\Big)\Bigg) && (34)
\end{aligned}$$

The expression path encodes all possible simple paths from some source node s to the target of some demand d. A notable difference is in Condition 28, where we now only want the simple paths of length $k-1$ from s to the target of demand d. Condition 31 has also been added to ensure that we only get simple paths between nodes that are source and target for the demands, and not all the possible simple paths in the network.

In the case where $|D| + 1$ is not a power of 2, edges can encode demands that do not exist. We therefore define $\mathit{onlyLegalDemands}$, which encodes that if an edge do not encode any demands, it must encode that no demand uses it.

$$\mathit{onlyLegalDemands}(\overline{\mathbf{p}^{\boldsymbol{\lambda}}}) = \bigwedge_{e \in E} \exists \overline{\mathbf{e}}, \overline{\mathbf{e}}(e) \wedge \left(\neg (\bigvee_{d \in D} \overline{\mathbf{p}_{\mathbf{e}}^{\boldsymbol{\lambda}}}(d) \right) \implies (\overline{\mathbf{p}_{\mathbf{e}}^{\boldsymbol{\lambda}}} = \overline{\mathbf{ff}}) \right) \tag{35}$$

Definition 16 (Routing and wavelength assignment).

$$\mathit{rwa}(\overline{\mathbf{p}^{\boldsymbol{\Lambda}}}) = \Bigg(\bigwedge_{d \in D} \exists \overline{\mathbf{d}}, \overline{\mathbf{s}} \,.\, \mathit{source}(\overline{\mathbf{s}}, \overline{\mathbf{d}}) \wedge \overline{\mathbf{d}}(d) \wedge \tag{36}$$

$$\bigvee_{k \leq i} \left(\mathit{path}(\overline{\mathbf{s}}, \overline{\mathbf{d}}, \overline{\mathbf{p}^{\boldsymbol{\lambda}_k}}) \wedge \bigwedge_{\lambda' \in \Lambda \setminus \{\lambda_k\}} \bigwedge_{e \in E} \exists \overline{\mathbf{e}}.\overline{\mathbf{e}}(e) \wedge \neg \overline{\mathbf{p}_{\boldsymbol{e}}^{\boldsymbol{\lambda'}}}(d) \right) \Bigg) \wedge \tag{37}$$

$$\bigwedge_{\lambda \in \Lambda} \mathit{onlyLegalDemands}(\overline{\mathbf{p}^{\boldsymbol{\lambda}}}) \tag{38}$$

The BDD rwa encodes all solutions to the RWA problem. Condition 37 enforces that when a demand uses a given wavelength, the demand is not encoded to use any other wavelength. Lastly, Condition 38 enforces that only legal demands are encoded for all wavelengths.