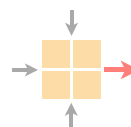




Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



Networked Systems  
ETH Zürich — seit 2015

# Extending NetComplete

Master Thesis

MA-2020-13

Author: Robin Berner

Tutor: Rüdiger Birkner, Busse-Grawitz Coralie

Supervisor: Prof. Dr. Laurent Vanbever

April 2020 to September 2020

## **Abstract**

Network operators are often given only a few high-level requirements and have to write the corresponding low-level configurations by hand, giving room for potential errors. While already small errors in network configuration can cause local network outages, causing frustration and financial loss for the network operators, bigger errors can cause outages for entire countries. To address the impact of human error in network outages, researchers have shown an increased interest on configuration verification and configuration synthesis tools.

In this thesis we extend the capability of NetComplete, a tool to generate and complete network configurations out of high-level requirements. NetComplete not only ensures the error free configurations, but also guarantees that all user requirements are met if the topology and other requirements allow it. We extend the current support of OSPF as only IGP by RIP and IS-IS and a common framework which connects all IGPs.

Furthermore we also add support for new OSPF features, such as stubby area configuration and an area synthesis tool. The area synthesis tool is both able to complete existing area configurations and also generate area assignments from scratch by utilizing the Z3 SMT solver. All new additions and features still respect the scalability to large networks which allows us to synthesize configurations for networks of multiple hundred routers in minutes.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Tasks . . . . .	2
<b>2</b>	<b>Background And Related Work</b>	<b>3</b>
2.1	IGPs . . . . .	3
2.1.1	RIP . . . . .	3
2.1.2	IS-IS . . . . .	3
2.1.3	OSPF . . . . .	4
2.1.4	Common Features And Differences . . . . .	6
2.2	SMT Solver . . . . .	7
2.3	Related Work . . . . .	7
<b>3</b>	<b>Design</b>	<b>9</b>
3.1	IGP Additions . . . . .	9
3.1.1	The Starting Point: NetComplete . . . . .	9
3.1.2	RIP . . . . .	10
3.1.3	IS-IS . . . . .	11
3.1.4	Common core . . . . .	11
3.1.5	Hierarchical OSPF . . . . .	12
3.2	Area Synthesis . . . . .	13
3.2.1	General Area Synthesis . . . . .	14
3.2.2	Area Connectivity Via Reachability . . . . .	15
3.2.3	Area Connectivity Via Shortest Paths . . . . .	18
3.2.4	ABR Generation . . . . .	18
<b>4</b>	<b>Implementation</b>	<b>20</b>
4.1	IGP Additions . . . . .	20
4.1.1	RIP, IS-IS And Common Core . . . . .	20
4.1.2	Hierarchical OSPF . . . . .	21
4.2	Area Synthesis . . . . .	22
4.2.1	General Area Synthesis . . . . .	22
4.2.2	Area Connectivity Via Reachability . . . . .	23
4.2.3	Area Connectivity Via Shortest Paths . . . . .	23
4.2.4	ABR Generation . . . . .	23

<b>5</b>	<b>Evaluation</b>	<b>24</b>
5.1	Setup . . . . .	24
5.1.1	Topologies . . . . .	24
5.2	IGP Additions . . . . .	27
5.3	ABR Generation . . . . .	27
5.4	Area Synthesis Reliability . . . . .	38
5.5	Area Synthesis Performance . . . . .	44
5.5.1	Number Of Areas Available To The Synthesis . . . . .	44
5.5.2	Topology Size . . . . .	50
5.5.3	Threshold For ABR Generation . . . . .	52
5.5.4	Topology Shape . . . . .	57
5.5.5	Manual Area Input . . . . .	60
5.6	Discussion . . . . .	61
<b>6</b>	<b>Conclusion And Outlook</b>	<b>63</b>
6.1	Conclusion . . . . .	63
6.2	Outlook . . . . .	63
	<b>References</b>	<b>65</b>

# Chapter 1

## Introduction

In this chapter we will talk about the motivation behind this work (Section 1.1) and what tasks need to be solved to achieve our goal of expanding NetComplete (Section 1.2).

### 1.1 Motivation

In recent years the importance of the Internet has grown substantially, not only for personal usage but also for businesses. Some of the most valuable companies, such as Amazon, Google and Facebook are heavily dependent on the internet to a point where network downtimes cause large financial damages. Network failures are surprisingly easy to bring about by simple mistakes, such as a misconfiguration of a single metric on a router interface or a spelling mistake that assigns an unintended area or IP to a router, which can cause it to be unreachable. Managing a network properly is a complex task, as network operators are often only given a few high-level requirements that describe the behavior of the whole network. The specific low-level configurations for every router inside the network have to be designed accordingly, often by hand. To reduce the chance of network outages due to human error, research has recently been progressing on configuration verification [15] [19] and synthesis [12] [14] to help detect and avoid low-level configuration mistakes. Configuration synthesis is especially promising since it ensures the correctness of the configuration by completely removing the risk of spelling mistakes and misconfigurations. Furthermore it also guarantees that the high-level requirements are met.

NetComplete [14] presented an approach on configuration synthesis that respects already existing partial configurations of the network. This makes the configuration output more readable and keeps the provided configuration sketch intact. NetComplete simply fills in the missing elements, which allows the user to recognize his input in the output and relate the generated configuration to it. On the other hand partial configurations together with heuristics allow NetComplete to scale to large networks of a few hundred routers.

However, most configuration synthesizers such as Propane [12] and Genesis [18] do not support a great variety of protocols and only support a limited amount of features for each protocol. In this thesis we want to extend the support for different protocols supported in NetComplete. Currently the only IGP that is supported is OSPF, which is not enough and needs to be extended by other protocols such as RIP and IS-IS. Furthermore the supported version of OSPF only includes a very limited amount of features, missing one of the most essential ones, hierarchical OSPF.

## 1.2 Tasks

In order to extend the number of supported protocols and features, we want to address the following tasks during this thesis:

A Extend the number of supported protocols

I Add support for RIP

II Add support for IS-IS

III If possible create a common synthesis core that allows quick addition of similar IGP protocols

B Hierarchical OSPF

I Autocomplete stubby area configurations

II Generate area assignment configuration for a given ABR and area assignment

i Given all ABRs and a complete area assignment

ii Given all ABRs and a partial or no area assignment

iii Given only the topology

## Chapter 2

# Background And Related Work

This section gives an overview over the IGPs the thesis includes (Section 2.1) and SMT solvers (Section 2.2). Furthermore it describes the related work (Section 2.3).

### 2.1 IGPs

This section goes over the theoretical aspects and limitations of the three IGPs featured in the thesis, RIP (Section 2.1.1), IS-IS (Section 2.1.2) and OSPF (Section 2.1.3). Lastly we also provide an overview over the common features and the differences between those IGPs (Section 2.1.4).

#### 2.1.1 RIP

The Routing Information Protocol (RIP) is a distance-vector routing protocol, which means that the routing information only consists of the next hop and the distance to the target. Neighboring routers exchange this information periodically and update their routing tables respectively.

##### Metric

RIP measures the distance by the hop count by default, meaning the distance increases by one for each router that the way to the target. As the hop count in the RIP information exchange packets is limited to a maximum of 15, the maximum distance of a target cannot exceed 15. RIP allows to offset its cost metric for every interface on a router [8], which is used to set the weight of each individual edge in the network topology.

#### 2.1.2 IS-IS

Intermediate System to Intermediate System (IS-IS) is a link-state routing protocol that runs on the OSI Layer 2. In link-state routing protocols each router floods its link state information through the network, allowing every router to build their own database of the network topology by collecting all flooded information. Based on this database each router computes a shortest path for all destinations and packets are forwarded along the computed path.

##### Metric

There are two metric styles for IS-IS, narrow and wide. Narrow metric is the default metric, which uses a weight of 10 for each interface by default. It is possible to reconfigure the metric to use weights between 1 to 63. The maximum metric of a path, for all hops combined, is 1023. The wide

metric expands these limitations to a maximum value of 16,777,215 for individual interfaces and 4,261,412,864 for the total path metric. If there is a mismatch between the two metric styles the adjacency is maintained but routes with a different metric style are not accepted.

### Level

IS-IS routers can be part of either level 1 (L1), level 2 (L2) or both (L1/L2). L2 routers are the backbone of the network, while the L1 routers represent the non-backbone areas. L1/L2 routers connect the L1 areas with the backbone L2 area. Adjacencies can only be formed according to Table 2.1.

Router combination	Adjacency
L1 + L1	Only form L1 adjacency if area ID matches
L2 + L2	Always form L2 adjacency
L1 + L1/L2	Only form L1 adjacency if area ID matches
L2 + L1/L2	Always form L2 adjacency
L1/L2 + L1/L2	Always form L2 adjacency, additionally form L1 adjacency if area ID matches

Table 2.1: Adjacency of IS-IS routers

### NET

Each IS-IS router needs a Network Entity Title (NET) as address, consisting of the area ID, the system ID and the network service access point selector (SEL). The NET address is represented in hexadecimal and its length can vary from 8 to 20 octets. Following Cisco's common addressing practices [5], the NET structure has a leading octet which usually defaults to 49 but can be picked at will, followed by two octets representing the area code. After this there are six octets representing the router loopback address, followed by the final octet which is zero. An example NET which follows this scheme for a router in area 1 with the loopback address of 192.168.1.1 looks like Figure 2.1.

Area ID	Loopback address	SEL
49.0001	.1921.6800	.1001.00

Figure 2.1: NET structure example

### Advertisements

L1/L2 routers do not advertise L2 routes to L1 routers by default, meaning an L1 router does not know about targets outside of its area. As we will see in Section 2.1.3, this behavior is similar to totally stubby areas in OSPF. The configuration of route redistribution circumvents this behavior.

#### 2.1.3 OSPF

Open Shortest Path First (OSPF) is another link-state routing protocol similar to IS-IS, but runs on top of IP and is an OSI Layer 3 protocol. As OSPF is a link-state routing protocol, changing the edge weights has a direct influence on the shortest paths inside the topology.



## Metric

The edge costs are directly configured on each interface of a router. The link-state advertisement (LSA) has a 16-bit field for the interface cost, which limits the cost to a maximum of 65,535.

## Hierarchical OSPF

Hierarchical OSPF benefits the scalability of the protocol greatly [4] [6]. A router only has to share an identical link-state database with all other routers in its area. This reduces the size of the database and therefore saves memory on the router. On top of this the router also has to process less LSAs which reduces the workload on the CPU. Finally the flooding is largely limited to the area as the routers must only maintain the database in their areas, reducing the bandwidth needed in the network.

When working with multiple areas in OSPF there are a few things that have to be considered such as area types, traffic types, ABRs, route advertisement prioritization and stubby areas.

**Area Types** A router can either be part of the backbone (area 0), meaning all interfaces of said router belong to the backbone, it can be an area border router (ABR), meaning it has at least one interface belonging to the backbone and at least one interface belonging to a non-backbone area, or it can be a non-backbone router, meaning none of its interfaces belong to the backbone.

**Traffic Types** Traffic can be distinguished into three types. Intra-area traffic is traffic where the packet never leaves the area it originated from. Inter-area traffic is defined as traffic that passes different areas along its path. External traffic is the last type of traffic, it is called external traffic since it either originates from outside the OSPF domain or it leaves the OSPF domain along its path. External traffic can be split into external type 1 which includes both the external path cost and the internal path cost for the total path cost calculation, and external type 2 which only includes the external path cost.

**ABR** ABRs play a very important role, as they summarize the information about the topology of all areas attached to it and propagate the collected information to the other areas. Due to this behavior all inter-area traffic has to go through the backbone. Non-backbone areas are not allowed to exchange packets directly. An ABR also summarizes routes and only provides the shortest path route to its areas. Similarly it also only advertises the shortest routes to the outside.

**Route Advertisement Prioritization** When routers receive advertisements they adapt their routing tables according to three criteria. Most important is the longest prefix match, more specific advertisements are always preferred. Second, intra-area routes are preferred over inter-area routes which are preferred over external type 1 and external type 2. The last criteria is the accumulated path cost from the edge weights whereby lower cost is preferred over higher cost.

**Stubby Areas** Last but not least non-backbone areas can be configured to be stubby. We differentiate between five types of stubbiness. Configuring an area to be stubby means the ABR connected to the stub area will block certain types of LSAs. For some configurations the ABR also injects a default route to itself into its areas. An overview over the different types of stubbiness is given in Table 2.3. An overview over the first seven different LSA types can be found in Table 2.2 [7]. The remaining LSA type 8 to type 11 are either for IPv6 usage or for application specific purposes and therefore omitted.

Type	Name	Description
1	Router LSA	Router announces its links, only flooded across their own area
2	Network LSA	Designated router lists all other routers which are part of the segment, only flooded across their own area
3	Summary LSA	Summarization routes injected by ABR
4	Summary ASBR LSA	Needed since type 5 LSA are flooded with ASBR's ID as source, which is not propagated to other areas. Type 4 LSA provide reachability to ASBRs
5	External LSA	Routes imported into OSPF domain
6	Multicast LSA	Used for multicast extensions, only supported on few routers
7	NSSA External LSA	Routers in not-so-stubby-area do not receive external LSAs but use type 7 LSA to send external routing information to the ABR. The ABR translates it to LSA type 5.

Table 2.2: Types of LSAs

Type	Allowed (LSA + Default)	Blocked (LSA + Default)
Stub	1, 2, 3, Default	4, 5, 7
Stub no summary	1, 2, Default	3, 4, 5, 7
Not so stubby	1, 2, 3, 7	4, 5, Default
Not so stubby default information originate	1, 2, 3, 7, Default	4, 5
Not so stubby no summary	1, 2, 7, Default	3, 4, 5

Table 2.3: Types of stubbiness

In a stub area, type 5 LSA are blocked, meaning that external routes are not visible. As external routes are not visible to a stub area, LSA type 4 which provide information on how to reach the ASBR are also not needed and blocked by the ABR as well. External destinations can still be reached over the default route to the ABR. Stub areas with no summary, also known as totally stubby areas, additionally block LSA type 3 which correspond to inter-area routes. Inter-area routes as well as external routes can be reached via the default route. Not so stubby areas (NSSA) block external routes received from other areas but allow LSA type 7 which are used to advertise its own external routes to other areas. No default route is injected. NSSA with **default information originate** is similar to regular NSSA but the ABR injects a default route on top of it. Not so stubby areas with **no summary**, also known as totally not so stubby areas, additionally block inter-area routes on top of injecting a default route, which can be used to reach the inter-area routes. For all stub areas, every router in the area needs to configure the area as stub, however only the ABR needs to configure the **no summary** or **default information originate** options [1].

#### 2.1.4 Common Features And Differences

All three of the presented IGP protocols have quite a few common features. Table 2.4 shows the most important aspects which can be different between the protocols and have to be considered in this thesis. As we can see the only big difference lies in the limitations of RIP and narrow metric IS-IS compared to OSPF or wide metric IS-IS. OSPF and wide metric IS-IS have high limits on

edge weights and total path costs which allows a more or less free choice of edge weights. When working with RIP or the narrow metric of IS-IS one has to consider those limiting factors.

Feature	RIP	IS-IS	OSPF
Edge weight	Set for each edge Maximum 15	Set for each edge Maximum 63 (narrow) Maximum 16,777,215 (wide)	Set for each edge Maximum 65,535
Path cost	Maximum 15	Maximum 1023 (narrow) Maximum 4,261,412,864 (wide)	No Limit
Path cost calculation	Summation of all edge weights	Summation of all edge weights	Summation of all edge weights

Table 2.4: Common features of IGPs

The decision which protocol to use can not be answered globally. Each protocol comes with its own range of advantages and disadvantages [9] [9] [16]. RIP is a fairly simple and easy to understand protocol that is supported on most routers. Due to it being a distance-vector routing protocol, it has a fairly low memory and CPU requirements which comes at the cost of a higher bandwidth usage. IS-IS and OSPF are very similar in its functions but also have quite a few differences. IS-IS is generally more flexible, at the cost of being harder to configure, and offers a higher security since it runs on OSI Layer 2 and does not rely on IP. The scalability of IS-IS can be greatly improved by setting the lifetime of its link state protocol data unit to up to 18 hours, leading to a lot less data transfer compared to OSPF, which has a lifetime of 20 minutes on its LSAs. Lastly most big internet service providers (ISPs) use IS-IS which leads to faster updates and fixes. On the other hand OSPF is more widely used and therefore there is a lot of information and help available online.

## 2.2 SMT Solver

Satisfiability modulo theories (SMT) solvers are a powerful tool. They can be used to verify the correctness of expressions or for synthesizing missing parts by searching through the space which satisfies the logic constraints. They can solve problems expressed with simple first-order logic expressions. In this thesis we leverage the possibility of synthesizing configurations out of basic logic constraints. In this thesis we use the Z3 SMT solver [13] which synthesizes a configuration in a single step and repeats the process until a valid solution is found. This essentially means that the solver assigns values to all variables and only then checks if this solution satisfies all of the constraints. The Z3 SMT solver is non-deterministic, meaning the same input does not always produce the same output since there is randomness involved when breaking ties during the resolution.

## 2.3 Related Work

**NetComplete** [14] synthesizes low level network configurations from incomplete or nonexistent configurations and high level requirements. It supports static routes, OSPF and BGP. With the help of the powerful Z3 SMT solver [13], NetComplete is able to synthesize link weights and BGP policies from high level requirements and policy sketches. It was a first and important step into network-wide configuration synthesis supporting multiple protocols and the possibility of including

and completing pre-existing configurations. In recent years research lead to a few similar approaches on automatically generating configurations for networks.

**Propane** [12] is one of them. It can generate configurations out of network-wide abstractions such as paths to cause the routing decisions to prefer certain paths over others. It is also able to compile and implement global specifications through the use of BGP for any case where the limitations of BGP as a path vector protocol allow to do so.

**Genesis** [18] also synthesizes network configurations in this case switch forwarding tables in data-center networks. It takes high level policies and synthesizes the corresponding low level configurations for the switch, using the advancement and the speed of SMT solvers.

Similar to NetComplete, both take away the error-prone step of configuring every single switch manually and provide the user with synthesized configurations for the whole network. But both Propane and Genesis focus on rather specific tasks. Propane is limited to BGP, Genesis is specialized at synthesizing configurations for datacenters. With the extension of NetComplete we aim at providing a larger variety of protocols with a single tool.

# Chapter 3

## Design

The following chapter will first present the designs to extend the supported IGP of NetComplete by RIP and IS-IS and utilize the similarity of the IGP to create a common core. Additionally we extend the already implemented OSPF protocol with the support for hierarchical OSPF (Section 3.1). In the second part of the thesis we extend the project with a completely new functionality which allows a user to synthesize an area assignment and generate ABRs for the use of hierarchical OSPF (Section 3.2).

### 3.1 IGP Additions

First we will provide an overview of the already existing implementation of OSPF and describe the workflow (Section 3.1.1). After NetComplete was extended by adding support for RIP (Section 3.1.2) and IS-IS (Section 3.1.3) in a first step, we realized the design for the different IGP protocols was fairly similar. Based on this discovery we decided to reuse as much of the code as possible with the help of a common core (Section 3.1.4). Additionally to the support of two new protocols, OSPF was extended to support basic features of hierarchical OSPF (Section 3.1.5).

#### 3.1.1 The Starting Point: NetComplete

All changes are made based on the initial OSPF design of the NetComplete project. The workflow is shown in Figure 4.1 and starts with the input of path requirements and the network topology, edge costs of partial configurations are an optional input. It then synthesizes IP addresses for all interfaces and updates the topology with all the information about a node's interface. Next it generates all necessary constraints for the SMT solver to synthesize the missing edge weights. The constraints needed are the limitations of each individual edge weights being greater than 0 and the total path cost of alternative paths needs to be higher than the path cost of the requirement. The synthesized edge weights update the corresponding edge weights in the topology. Lastly it generates the configuration for each router, based on the edge weights and interface addresses saved in the topology. The configuration consists of enabling OSPF for each router and the IP address and edge cost of all interfaces.

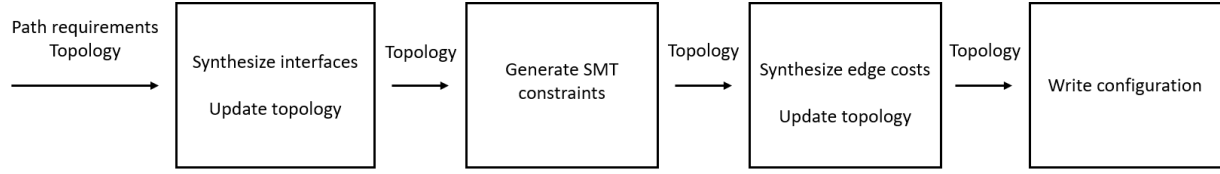


Figure 3.1: Workflow of IGP synthesis

### 3.1.2 RIP

#### Metric

Based on the insights we gained from the overview over the IGP protocols in section 2.1, RIP is implemented fairly similar to OSPF. The first change we need to make is for the edge costs, as RIP has a maximum cost for edges and total paths of 15. This results in additional constraints we need to generate for the SMT solver to respect this limitation for both, individual edge costs, as well as for entire path requirements. We do not limit paths in the topology that are not in the requirements. If any path cost is higher than 15, RIP considers the destination as unreachable over this path. As a consequence of this design choice we have the possibility to generate edge weights which make it impossible for two routers to communicate. Figure 3.2 shows an example that describes this issue. If the user only specifies a path from  $R1$  to  $R16$  as  $(R1, R2, R3, \dots, R15, R16)$  the SMT solver will set all edge weights on this path to one as the path is 15 hops long and this is the only way to satisfy the constraints. The only alternative path  $(R1, R20, R21, R16)$  has constraints limiting it to a higher cost than the specified path. One possible solution that solves this problem is  $edge\_cost(R1, R20) = 15$ ,  $edge\_cost(R20, R21) = 1$  and  $edge\_cost(R21, R16) = 1$ , which results in a total path cost for  $R1, R20, R21, R16$  of  $edge\_cost(R1, R20) + edge\_cost(R20, R21) + edge\_cost(R21, R16) = 17$ . This solution results in  $R1$  and  $R21$  being unreachable to each other, as both possible paths between them have a cost of 16. To resolve this issue the user needs to specify a path from  $R1$  to  $R21$ .

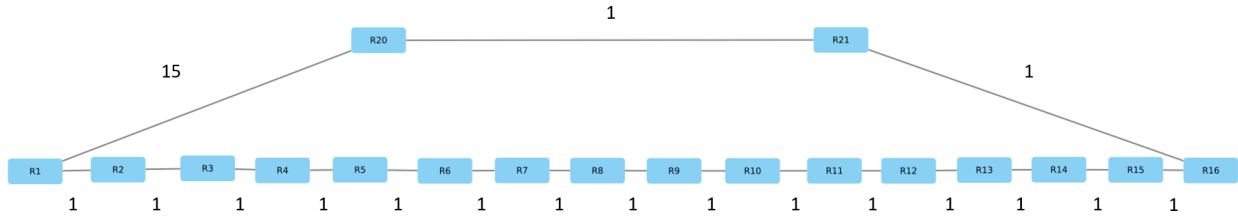


Figure 3.2: Example topology for unreachable destination in RIP

#### Configuration Generation

The second and already last change required is the generation of the configuration files for the individual routers. Besides activating RIP on all routers, we need to configure the interfaces with their IP subnets similar to OSPF. However the edge costs are not configured on the interfaces directly, instead we need to configure them using the `offset-list <access-list number> <in/out> <offset amount> <interface>` command. Any incoming or outgoing advertisement that matches the access list has its metric increased by the `<offset amount>` which we define as  $edge\_cost - 1$ . As `<access-list>` we define `access-list 1 permit ip any any` which will match

for all advertisement packets.

### 3.1.3 IS-IS

#### Metric

Even though Cisco recommends using the wide metric style [11] we still want to give the user the option to choose between the two metric styles. The narrow metric adds, similar to RIP, additional constraints to the SMT solver as it limits individual edge costs to a maximum of 63 and total path costs to a maximum of 1023. The wide metric loosens those restrictions and allows costs up to 6,777,215 for individual edges and 4,261,412,864 for entire paths. Similar to the OSPF design we consider those limitations for the wide metric high enough to not add any additional constraints. The total path cost limitation apply only to the paths from the requirements. Likewise to RIP, we can generate cases where the maximum value is exceeded for certain paths. However, if values above the maximum are calculated they will simply be replaced by the maximum value, which means the destination is still reachable.

#### Configuration Generation

When generating the configuration files, we have to enable IS-IS, set the IP subnets and configure the cost metric for each interface. We configure the cost metric as absolute value and not as offset as in RIP. Also we need to assign each router to a level and configure a NET address and the metric style it uses. We default the level assignment to level 2 as we are not using hierarchical IS-IS. The NET address, as described in 2.1.2, starts with a leading 49, the area defaults to area 1 as we are not using hierarchical IS-IS, followed by the loopback address and the final zero octet. We configure the metric style according to the user's choice to either narrow or wide.

### 3.1.4 Common core

The addition of RIP and IS-IS showed quite some similarities with a lot of duplicated code. Therefore we decided to combine all three IGPs into a single common core design, which aims to reuse as much code as possible and avoid unnecessary code duplications. As the protocol is already specified in the path requirements we don't need any additional user input to help differentiate between the IGPs.

#### Metric

As we already discussed in Section 2.1.4 the limitations to the cost metric are different for every IGP. Therefore we needed to introduce a function that adds the correct limitations according to the protocol specified.

#### Configuration Generation

As every IGP has its own commands to configure a router and its interfaces, we need to differentiate between the different IGPs when generating the configuration files. Since we separate the interface configuration from the router configuration it is also important to consider that IS-IS has to be configured for every interface whereas RIP and OSPF configure it once for the entire router. Additionally RIP configures its interface costs on a router level with the help of offset lists while OSPF and IS-IS configure the costs directly on the interface.

### 3.1.5 Hierarchical OSPF

After we extended the supported IGP protocols by RIP and IS-IS and added the common core, we wanted to focus on adding additional utility to the supported protocols. An important feature of OSPF is the possibility to configure a hierarchical routing structure. If we want to leverage the benefits of smaller routing tables and less CPU and bandwidth usage we have to consider a few general restrictions introduced by the hierarchical structure. Additional restrictions apply, depending on the path requirement type. We described the theoretical aspects behind those restrictions in Section 2.1.3. Furthermore we want to allow a user to specify certain areas to be stubby and automatically complete this configuration for all routers in the same area.

#### Restrictions

To check the hierarchical OSPF restrictions we need all path requirements as well as a list containing every node and the areas it is assigned to. We recognize ABRs by the fact that they have more than one area assigned. Depending on the input we want to check if it is possible to generate a valid output. Table 3.1 shows us all restrictions and what we need to check.

Restriction type	Description
General	ABR always needs to have at least one interface belonging to the backbone
General	Two routers can only communicate if they share a common area
Simple path	If the requirement specifies an inter-area path there cannot exist an intra-area path alternative as it would always be preferred
ECMP	Intra-area and inter-area paths cannot be mixed in an ECMP requirement as intra-area would always be preferred
ECMP	If the requirement only specifies inter-area paths there cannot exist an intra-area path alternative as it would always be preferred
Any-path	If the requirement only specifies inter-area paths there cannot exist an intra-area path alternative as it would always be preferred
Ordered	If the requirement specify at least one inter-area path all intra area paths have to be specified as well
Ordered	All inter-area paths specified in the requirement need to have a lower priority than any intra area path

Table 3.1: Hierarchical OSPF restrictions

**General** In all cases the general restrictions have to be checked. For routers that have multiple areas defined but are not assigned to the backbone we have and ABR without an interface that belongs to the backbone, resulting in a failure. In the case of two neighboring routers not sharing a common area, no adjacency is formed and the edge connecting them cannot be used for communication and will be removed for the rest of the synthesis.

**Simple path** For simple path requirements that specify an inter-area path we need to check that no alternative intra-area path exists as it would always be preferred and result in a failure.

**ECMP** In a load balancing scenario using equal-cost multi-path routing (ECMP) where a list of multiple paths is given, which are required to have the same lowest cost, we need to ensure the list does not contain a mix of intra-area and inter-area paths, as intra-area paths would always be preferred and no edge cost could ever achieve a load balancing between the two types. Furthermore if the list only contains inter-area paths we need to ensure there exists no alternative intra-area path.



**Any-path** For any-path requirements, where the input is a list of paths from which one of the specified paths has to be used for routing, we need to ensure that if we have only inter-area paths in the list, there exists no intra-area path alternative. In the case where we have intra-area and inter-area paths in the requirement and there exist additional intra-area paths the synthesis makes sure that the specified intra-area paths have a lower cost than the remaining ones, resulting in one of the specified intra-area routes being used.

**Ordered** Finally for ordered path requirements, where the input is a list whereas the position of the element represents its priority, we need to make sure that we have all intra-area paths specified if at least one inter-area path is specified. Additionally there cannot exist an intra-area path with a lower priority than an inter-area path.

### Stubby Areas

We also designed an auto-completion feature which allows the user to specify a default stubbiness for the whole network. Additionally the user can also specify individual nodes to be part of a stubby area and the tool automatically adds the stubbiness to all other routers inside the same area. We wanted to keep the stubbiness separated from the actual area numbers, as the user may not always know which router belongs to which area. This is the case if the user has an incomplete configuration file or utilizes the area synthesis described in Section 3.2. If an ABR is specified as a node that is part of a stubby area, the stubbiness will be applied to all areas it is part of with the exception of the backbone area. We need to apply the stubbiness for each area on each individual router which has an interface in the respective area. Table 3.2 gives an overview over the necessary commands.

Type	ABR	Non-ABR
Stub	<code>area &lt;number&gt; stub</code>	<code>area &lt;number&gt; stub</code>
Stub no summary	<code>area &lt;number&gt; stub no-summary</code>	<code>area &lt;number&gt; stub</code>
Not so stubby	<code>area &lt;number&gt; nssa</code>	<code>area &lt;number&gt; nssa</code>
Not so stubby default information originate	<code>area &lt;number&gt; nssa</code> <code>default-information originate</code>	<code>area &lt;number&gt; nssa</code>
Not so stubby no summary	<code>area &lt;number&gt; nssa no-summary</code>	<code>area &lt;number&gt; nssa</code>

Table 3.2: Commands to configure stubbiness on router

## 3.2 Area Synthesis

After ensuring a basic functionality of hierarchical OSPF we decided that we want to lower the required user input to a point where we support everything in between full area configurations down to no area configuration at all. We first developed a design to synthesize an area to router assignment, given a list of all ABRs in the topology (Section 3.2.1). The proposed design ensures connectivity between neighboring routers and either minimizes or maximizes the amount of areas. It does not guarantee the connectivity of individual areas, which is why we propose additional constraints in section 3.2.2. After implementing the generation of additional constraints to guarantee area connectivity, we realized some constraints can verify each other essentially removing their functionality which makes it possible to generate disconnected areas. Section 3.2.3 bypasses this flaw with a more extensive constraint generation and represents the final functional version of the

area synthesis. After verifying that we had a correctly working design to synthesize an area assignment, we also investigated the possibility of removing the input of an ABR list to the synthesis and generating it ourselves (Section 3.2.4), further reducing the user input.

### 3.2.1 General Area Synthesis

Since we are working with the Z3 SMT solver to generate an area assignment, we are limited to utilizing basic linear equations and inequalities. To represent the area synthesis within those limitations, we need to generate constraints for area membership variables, number of areas that are assigned to a router, neighboring relationships, non-empty areas and finally the total amount of areas.

#### Area Membership Variables

We represent a router's area with a set of area membership variables, each in the form of an integer written as *Router\_{router name}\_Area\_{area number}*. They are integers that are limited to the values 0 and 1. Value 0 signals that the router is not part of the area, while value 1 signals that it is part of the area. The membership variables also offer an easy way to manually input some area assignments into the synthesis by simply adding a constraint which sets the corresponding variable to 1. The example in Table 3.3 describes ABR *R1* as part of area 0 and 1, and non-ABR *R2* as part of area 1.

Variable	Value
Router_R1_Area_0	1
Router_R1_Area_1	1
Router_R2_Area_0	0
Router_R2_Area_1	1

Table 3.3: Area membership variables for R1 in area 0 and 1 and R2 in area 1

As we limit those memberships variables to either value 0 or 1 we add constraint 3.1 and 3.2 for each router *X* and area *Y* combination.

$$Router\_X\_Area\_Y \geq 0 \quad (3.1)$$

$$Router\_X\_Area\_Y \leq 1 \quad (3.2)$$

#### Number Of Areas

Non-ABR routers *X* can only be part of a single area, which we express with constraint 3.3. We limit the number of areas of ABRs *Z* according to recommendations by Cisco [2] to a total of three, resulting in constraint 3.4. Since every ABR *Z* has to be part of the backbone we add constraint 3.5 for all of them.

$$\sum_Y (Router\_X\_Area\_Y) == 1 \quad (3.3)$$

$$\sum_Y (Router\_Z\_Area\_Y) \leq 3 \quad (3.4)$$

$$Router\_Z\_Area\_0 == 1 \quad (3.5)$$

### Neighboring Relationships

Next we set the relations between neighbors to ensure that they can communicate with each other and avoid separating the topology into disconnected parts. To achieve this we make sure that two neighbors always share at least one area (two neighboring ABRs can share more than one area). Constraint 3.6 is added for any two neighboring nodes X and Z, making sure that they have at least one area in common. If needed, we interpret the memberships variables as Boolean values by comparing them to 1, meaning they are true if they are 1 and false if they are 0.

$$\bigvee_Y [ \bigwedge [ Router\_X\_Area\_Y, Router\_Z\_Area\_Y ] ] \quad (3.6)$$

### Non-Empty Areas

Additionally we want to make sure that every area has at least one non-ABR router. This removes the possibility for ABRs to just randomly create empty areas and is important when maximizing the amount of areas in a last step. We achieve this by adding constraint 3.7 for every area Y and every ABR Z. It makes sure that either the ABR Z is not part of area Y or it has at least one non-ABR neighbor X that is part of area Y as well.

$$\bigvee [ \neg(Router\_Z\_Area\_Y), \bigwedge_X [ Router\_Z\_Area\_Y, Router\_X\_Area\_Y ] ] \quad (3.7)$$

### Total Number Of Areas

Finally we have the option to either maximize or minimize the amount of areas. We calculate the total number of areas utilized with equation 3.8 and use it in a maximizing or minimizing constraint. When maximizing the number of areas we have many small areas, while minimizing gives us fewer but larger areas. Each comes with its own benefit. Many small areas reduce the CPU usage for the shortest path calculation and reduces flooding, while few big areas offers better ways for route summarization and can therefore reduce the routing table size when utilizing route summarization. When minimizing the amount of areas assigned we utilize another recommendation from Cisco which states that no area should have more than 50 routers. We use this constraint to avoid an assignment that puts all routers into the same area. We add constraint 3.9 for all areas Y over all routers X. We avoid this constraint when maximizing as it is only a recommendation and can lead unsatisfiable problems for certain combinations of topologies and ABR lists. One example of such an unsatisfiable problem is a chain of more than 50 routers attached to an ABR.

$$\sum_Y [ \bigvee_X [ Router\_X\_Area\_Y ] ] \quad (3.8)$$

$$\sum_X [ Router\_X\_Area\_Y ] \leq 50 \quad (3.9)$$

#### 3.2.2 Area Connectivity Via Reachability

As previously mentioned, the area synthesis proposed in section 3.2.1 does not ensure the connectivity of individual areas. We show why this is an issue and how to solve the problem by creating a reachability chain and ensure that every router can reach all other routers inside the area. Finally we describe why this design does not achieve its intended purpose.

### Disconnected Area Issues

Since the backbone area has to be fully connected we need to ensure area connectivity. Fixing the area connectivity in the current design is not trivial, as reassigning one of the disconnected parts to a new area can cause a violation of some of the constraints, such as a maximum of three areas per ABR. Furthermore, since every ABR belongs to the backbone area by default, one can easily create a topology and ABR input where the backbone is disconnected. Figure 3.3 shows an example of a disconnected backbone. If we choose  $R1$  and  $R3$  as ABR, marked in red, and maximize the amount of areas, we will get a non-backbone assignment for  $R2$  which disconnects the two ABRs.



Figure 3.3: Disconnected backbone between ABRs  $R1$  and  $R3$  by non-backbone router  $R2$

### Reachability Chain Version 1

In order to solve this issue we designed additional constraints ensuring that a router can only be part of an area if it reaches all other routers in the same area over an intra-area path inside said area. We added additional variables for every pair of routers over all areas in the form of  $Reachable_{\{router\ 1\}\{router\ 2\}\{area\}}$ , which track if the router pair can reach each other over the specified area. The reachability variables are identical in both directions, meaning  $Reachable_{\{router\ 1\}\{router\ 2\}\{area\}} = Reachable_{\{router\ 2\}\{router\ 1\}\{area\}}$ . We initialize the reachability by setting the  $Reachable$  variables to true for common areas between neighboring routers. To check the reachability for all router and areas we create a chain verification. In the first version we designed, the remaining non-initialized  $Reachable$  variables for any pair of routers  $S$  and  $T$  are defined using constraint 3.10 for all areas  $Y$ . If all three routers  $S$ ,  $T$  and  $Z$  are in area  $Y$  and both  $S$  and  $T$  can reach router  $Z$ , they can also reach each other over router  $Z$ .

$$Reachable_{S,T,Y} = \bigwedge [ Router_{S\_Area\_Y}, Router_{T\_Area\_Y}, \bigvee [ \bigwedge_{Z \neq S,T} [ Reachable_{S\_Z\_Y}, Reachable_{Z\_T\_Y}, Router_{Z\_Area\_Y} ] ] ] \quad (3.10)$$

To ensure the connectivity of the whole area we need to check that every router inside the area can reach all other routers inside the area. Constraint 3.11 needs to be added for all routers  $S$  and for every area  $Y$ . It ensures that, if router  $S$  is part of area  $Y$ , it can reach all other routers  $T$  that are part of area  $Y$ .

$$\bigvee [ \neg(Router_{S\_Area\_Y}), \bigwedge [ \bigvee_{T \neq S} [ \neg(Router_{T\_Area\_Y}), Reachable_{S,T,Y} ] ] ] \quad (3.11)$$

### Reachability Chain Version 2

To reduce the amount of constraints and the complexity of the problem we designed a second version, which takes physical connections into account by only searching connections over a router's neighbors instead of searching over every combination of routers. Furthermore, it does not check if every router can reach every other router inside an area but simply checks that it can reach every ABR which is part of the area. This means constraint 3.10 is replaced by 3.12 and constraint 3.11 is replaced by 3.13.

$$Reachable\_S\_T\_Y = \bigwedge [ Router\_S\_Area\_Y, Router\_T\_Area\_Y, \bigvee [ \bigwedge_{Z=nbr(S)} [ Reachable\_S\_Z\_Y, Reachable\_Z\_T\_Y, Router\_Z\_Area\_Y ] ] ] \quad (3.12)$$

$$\bigvee [ \neg(Router\_S\_Area\_Y), \bigwedge [ \bigvee_{T=ABR} [ \neg(Router\_T\_Area\_Y), Reachable\_S\_T\_Y ] ] ] \quad (3.13)$$

### Design Flaw

After implementing both proposed versions we realized that they both suffer from the same issue. As pointed out in section 2.2, the SMT solver assigns all values in one go and only then checks for its correctness. As the reachability is built on top of a verification chain, it requires an iterative behavior. A simple example where the proposed design fails can be shown with the line graph of Figure 3.4 where the ABRs  $R2$  and  $R4$  are again marked in red.



Figure 3.4: Example topology for failing reachability chain

When the SMT solver tries to verify the assignment of Table 3.4, we expect it to recognize it as unsatisfied as  $R2$  and  $R4$  are not connected over the backbone.

Router	Areas
R1	0
R2	0,1
R3	1
R4	0,1
R5	0

Table 3.4: Area assignment for each router

The reachability between the two ABRs is expressed with the  $Reachable\_R2\_R4.0$  variable. Since the two ABRs are not neighbors the reachability is not initialized but defined via constraint 3.12, which simplifies to 3.14 when using the assignment. We also note down the simplified form of  $Reachable\_R1\_R4.0$  in 3.15.

$$Reachable\_R2\_R4.0 = \bigvee [ \bigwedge [ Reachable\_R2\_R1.0, Reachable\_R1\_R4.0 ] ] \quad (3.14)$$

$$Reachable\_R1\_R4.0 = \bigvee [ \bigwedge [ Reachable\_R1\_R2.0, Reachable\_R2\_R4.0 ] ] \quad (3.15)$$

When simplifying constraint 3.13 by using the values from the assignment, we get constraint  $Reachable\_R1\_R4.0$  for  $R1$  and  $Reachable\_R2\_R4.0$  for  $R2$ . Both of those reachability variables are no initialized and only defined by each other as well as  $Reachable\_R1\_R2.0$ , which is initialized to true as  $R1$  and  $R2$  are neighbors and share area 0. Since there are no other limitations, nothing keeps the SMT solver from setting both of them to true. If both  $Reachable\_R1\_R4.0$  and  $Reachable\_R2\_R4.0$  are set to true they verify each other and all constraints are satisfied. This makes the SMT solver believe that Table 3.4 is a valid assignment. It works analogously on the other side with  $R4$  and  $R5$ .

### 3.2.3 Area Connectivity Via Shortest Paths

In order to prevent the issue of the reachability chain we replace it by physical path exploration. This means instead of having a chain form along a path we simply check actual paths that exist in the topology. Since the number of possible paths between two routers grows extremely fast with the size of the topology, we chose to limit the design to only use all shortest paths. This significantly reduces the number of paths and therefore also the size of the constraints. However, the downside is that we limit at least one of the shortest paths to ensure connectivity. If there are non-shortest paths that already establish connectivity they will be ignored and the SMT solver will still pick one of the shortest paths. To verify that two routers  $S$  and  $T$  are reachable over a certain area  $Y$ , we can simply check if all routers on any of the shortest paths are assigned to area  $Y$ , leading to constraint 3.16.

$$Reachable\_S\_T\_Y = \bigvee [ \bigwedge_{shortest\ paths} [ \forall Z \in shortest\ path \mid Router\_Z\_Area\_Y ] ] \quad (3.16)$$

We can conclude that every non-ABR reaches at least one ABR that shares the same area if we take a look at the combination of constraint 3.3 and constraint 3.6. Constraint 3.3 tells us that every non-ABR router is assigned to a single area. Constraint 3.6 ensures that neighbors always share at least one common area. This allows us to further reduce the complexity as we only need to ensure that all ABRs belonging to an area can reach each other. We automatically know this holds true for all non-ABRs of the same area. To ensure the connectivity of a whole area  $Y$  we therefore utilize constraint 3.17 which is added for every area  $Y$ .

$$\bigwedge [ for S, T \in ABR \mid \bigvee [ Reachable\_S\_T\_Y, \neg(Router\_S\_Area\_Y), \neg(Router\_T\_Area\_Y) ] ] \quad (3.17)$$

At this point we have a design that is capable of generating an area assignment with only an ABR list as input.

### 3.2.4 ABR Generation

As we still need an ABR list as input for the area synthesis, we investigate the possibility of synthesizing ABRs as well to reduce the user input even further. Usually ABRs are more central routers where a lot of traffic passes through, making them important in holding the network together. Although this is not a necessary feature and any router can be configured as an ABR, we focus on this aspect as it offers a great way to separate the network into multiple areas only connect through those central ABRs. To recognize central routers where a lot of traffic passes through we use the concept of dominator nodes used in graph theory. More specifically we have a look at immediate

dominators in directed graphs. We define the required expressions to explain what an immediate dominator is as follows:

1. A Node  $d$  dominates node  $n$  if every path from the entry node into the graph to node  $n$  goes through node  $d$ .
2. A node  $n$  is strictly dominated by a node  $d$  if  $d$  dominates  $n$  and  $d$  does not equal  $n$ .
3. A node  $d$  is the immediate dominator of a node  $n$  if  $d$  strictly dominates  $n$  but does not strictly dominate any other node that strictly dominates  $n$ .

Using this concept we design our algorithm to generate ABRs by first transforming the topology into a directed graph. This is done by taking every edge in our topology and replacing it by two directed edges, one in each direction. Next we take every node as an entry node once and generate all immediate dominators for the topology. Except for the entry node, which either has its immediate dominator defined as none or itself, every node has an immediate dominator. Now we count which node appeared as immediate dominator how often. Nodes appearing the most frequent are the best candidates to choose as ABR as they separate the largest subset of nodes. For the last step we need a threshold which determines how many ABRs will be generated. Either the user specifies a threshold to use or we take a default threshold which we define as 0.7 (Section 5.3). We keep adding the most frequent immediate dominator to our list of ABRs until the sum of immediate dominator counts for all the ABRs is larger than the sum of all immediate dominator counts multiplied by the threshold. Equation 3.18 describes the stopping criteria and once reached, the ABR generation terminates. Algorithm 1 shows a pseudocode version of the ABR generation algorithm.

$$\sum_{node \in ABR} \text{dominator count}(node) \geq \text{threshold} * \sum_{node \in graph} \text{dominator count}(node) \quad (3.18)$$

---

**Algorithm 1** ABR generation algorithm

---

```

1: dominator_counter = [ ]
2: ABR_list = [ ]
3: for node in topology do
4:   tmp_dominator = generate_dominator(node, topology)
5:   update_dominator(dominator_counter, tmp_dominator)
6: end for
7: while sum(ABR_list) < threshold * sum(dominator_counter) do
8:   ABR_list.add(max_dominator(dominator_counter))
9: end while
10: return ABR_list

```

---

## Chapter 4

# Implementation

In this chapter we will go into some more detail about the implementation of the designs discussed in chapter 3. In section 4.1 we start with the addition of RIP and IS-IS combined with the common core, followed by the hierarchical OSPF checks and stubby area configurations. Section 4.2 concludes the chapter with details to the area synthesis using Z3.

### 4.1 IGP Additions

RIP, IS-IS as well as the common core require very similar changes in the same spots, therefore we grouped them together into section 4.1.1. Section 4.1.2 goes into some more detail about the path requirement limitations and the stubby configuration.

#### 4.1.1 RIP, IS-IS And Common Core

Figure 4.1 shows the workflow of the IGP synthesis again and marks the common core elements in green. We will now discuss the changes that are required in the input and the two blocks which are not contained in the common core.

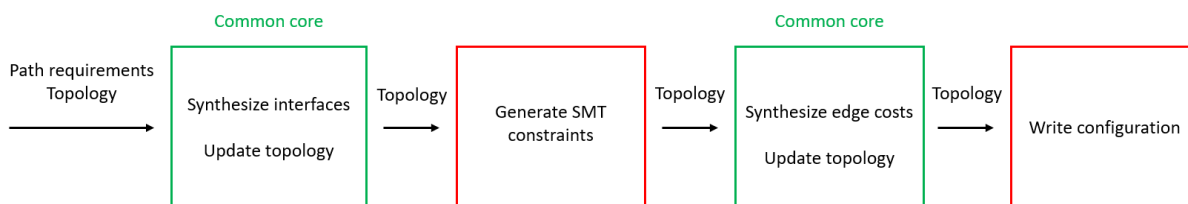


Figure 4.1: Workflow of IGP synthesis with common core marked in green

### Input

The IGP synthesis takes two inputs, the topology and the path requirements. The user specifies the path requirements with the protocol he wishes to use, the path to the destination and if the path is strict. A strict path means that all traffic to the target is dropped if the specified path is not available. As the protocol is specified in the path requirements, we can utilize this input to determine which protocol to use. No further input is required.



## Generate SMT Constraints

When calling the main synthesis we first push general graph constraints to the solver which limit individual edge costs, independent of any paths in the requirement. Here we have to consider the limitation for RIP and narrow metric IS-IS as discussed in section 3.1. Since the solver object is already configured as `solver = z3.Solver()` by the existing code, we can simply add constraints using `solver.add(constraint)`. In a next step the constraints for the path requirements are pushed. We have two different synthesis methods available in NetComplete. On one hand we have the concrete synthesis, considering all possible alternative paths from a specified source to destination, making sure that every alternative has a higher cost than the specified path. On the other hand there is counter-example guided synthesis (CEGIS), which aims at reducing the amount of generated path by only considering a subset of all alternative paths. The synthesis results will then be checked for validity and in case that there are cheaper alternative paths the synthesis is repeated adding additional constraints ensuring the previously cheaper paths have a higher cost as well. Both methods are implemented by adding constraint `cost(path_requirement) < cost(path_alternative)` for any alternative path. As RIP and narrow metric IS-IS limit the total path cost we can simply add an additional constraint at this point to ensure `cost(path_requirement) < limit`.

## Write Configuration

The last change required concerns the writing of the configuration files. Besides providing the protocol specified by the user, all other information required such as edge cost and interface information is stored in the topology. We already described the protocol specific changes in the respective Sections 3.1.2 and 3.1.3.

### 4.1.2 Hierarchical OSPF

To support hierarchical OSPF a few changes are required. Additional user input is required, we have to check the hierarchical OSPF limitations and we need to adjust the configuration generation.

#### Input

In our hierarchical OSPF implementation we added the support to assign areas to routers and configure their stubbiness. Two additional user inputs are required to achieve this. First the user needs to provide a list that contains the area assignment for every router in the topology. Second if the user wishes to configure the stubbiness of certain areas he needs to provide at least one router inside the area and the type of stubbiness that needs to be configured.

#### Hierarchical OSPF Limitations

We need to check the limitations defined in the previous chapter in Table 3.1 to ensure the path requirements do not violate any limitation. We additionally provide a utility function which allows us to check if a path is intra-area or inter-area.

#### Configuration Generation

When writing the configuration files, we have to additionally write the area configuration for the interfaces. There are two ways to do this. Either directly on the interface or specify the IP subnet in the router configuration and it will match accordingly. We decided to write the areas directly in the interfaces, as it gives the possibility to configure multi-area adjacency, meaning two neighboring

routers can configure the link between them to be part of multiple areas.

In order to configure stubby areas according to the user input, we have to additionally generate a list containing all ABRs. We can generate the list of ABRs by checking for routers with more than one area assigned to them. Since stubbiness is configured differently on ABRs and non-ABRs, it is important to differentiate between the two. Table 3.2 provides all the necessary information we need to configure the stubbiness correctly.

## 4.2 Area Synthesis

This section will provide additional information on the code, complementing the theory of the design described in section 3.2. We show some of the limitations that we need to consider when working with the code and how one could adapt the code to modify those limitations.

### 4.2.1 General Area Synthesis

We already discussed the general area synthesis quite extensively in section 3.2.1 and there is only few things to consider.

#### Solver Initialization

Since we want to maximize or minimize the amount of areas utilized, the solver has to be initialized as `solver = z3.Optimize()`, instead of `solver = z3.Solver()`.

#### Maximum Number Of Areas

The maximum number of areas that can be utilized is given by the fact that every ABR can have a maximum of three areas assigned and every router can introduce a maximum of one new area. As every ABR is also part of the backbone it can only introduce a maximum of two new areas. This gives us `maximum_areas = min(number of nodes, 2*(number of ABRs))`.

#### Integer And Boolean Conversion

Since the membership variables in the form of *Router\_{router name}\_Area\_{area number}* are defined as integers, we need to convert them into Boolean expressions for certain cases where they are utilized in logic operations. We achieve this by comparing the integer to one, such as `z3.Int(var) == 1`, which returns true for variables equal to one and false for variables equal to zero. We sometimes also need to convert Boolean expressions to integers. To achieve this we defined a function that returns one when the input is true and zero when the input is false. Two constraints, one for each Boolean value, determine the behavior of the function and therefore fully define it. The corresponding constraints are `solver.add(function(True) == 1)` and `solver.add(function(False) == 0)`.

#### Constraint Functions

A convenient feature of the python interface of Z3 allows us to use lists as function arguments, which gives us an easy way to write `z3.Or(list[0], list[1],...) = z3.Or(list)`. Note that sometimes we convert logical-or constraints of the form `z3.Or(Router_X_Area_Y == 1, ...)` into constraints of the form `z3.If(Router_X_Area_Y == 1, ..., True)`. This conversion does not change the meaning of the constraint.

### 4.2.2 Area Connectivity Via Reachability

In both versions of the reachability constraint generation it is possible to half the amount of constraints by considering that reachable variables do not depend on the direction, which allows us to skip the constraint generation for half of the router combinations. To utilize this we could either push constraints setting `Reachable_S_T_Y = Reachable_T_S_Y` for any combination of routers S and T and area Y or we keep an ordered list of all router names. With the help of the ordered list we can generate the unique name for both, `Reachable_S_T_Y` and `Reachable_T_S_Y`, allowing us to keep down the amount of constraints even further. As already mentioned in chapter 3, both versions suffer from the problem that the Z3 SMT solver has no iterative behavior and can validate wrong assignments as described in section 3.2.2. While we did not find a way to prevent this behavior we can also not rule out the possibility to add additional constraints which resolve this issue.

### 4.2.3 Area Connectivity Via Shortest Paths

It is again possible to utilize the fact that reachable variables are neutral to direction, reducing the amount of generated constraints by half as well. Furthermore it is possible to skip the check for two ABRs that are directly connected to each other. The last factor in this implementation is the physical path exploration. In our design we reasoned to limit the physical paths to only use all shortest paths for scalability in bigger topologies. While this indeed lowers the amount of paths substantially, it is not a requirement to utilize shortest paths. The design works for any choice of paths, from utilizing all paths, over random path choices down to utilizing a single path. Generally speaking a bigger selection of paths offers the SMT solver more possibilities to maximize the amount of areas but increases the constraint generation time, the amount of constraints generated and most often also the synthesis time.

### 4.2.4 ABR Generation

The ABR generation is pretty well defined in section 3.2.4 and does not offer a lot of room for adjustments. Providing a topology is sufficient to generate an ABR list, alternatively the user can specify a threshold value. In the case where two routers have the same immediate dominator count, one of them is picked at random. There is no algorithm in place to help decide which of the routers is added to the ABR list. We do not modify the ABR list that is generated any further.

## Chapter 5

# Evaluation

In this chapter we want to evaluate a few aspects of the thesis. First, we want to see whether we implemented the new IGPs correctly. For that matter we created an automated testing framework using base cases (Section 5.2). Next we want to evaluate the error our ABR generation produces, to this end we measured the error for different topologies and thresholds (Section 5.3). Finally we want to evaluate the area synthesis. On one hand we want to see how good the results are that our area synthesis produces. To that end, we ran the synthesis for some tutorial topologies and compared the results to the given ground truth (Section 5.4). On the other hand we want to evaluate the performance for different tuning factors the user can influence (Section 5.5). An overview over the setup used for the area synthesis evaluation is given in section 5.1.

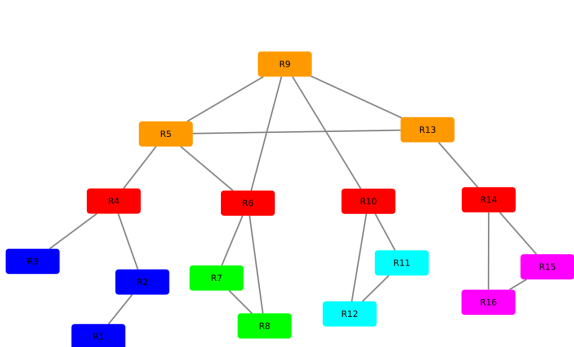
### 5.1 Setup

We use a virtual machine running Ubuntu 16.04 on a server with 16 GB of RAM, clocked at 2.3 GHz. Up to 4 cores are available although no parallel processing is involved. To visualize the topologies and area assignments we used Cytoscape [3], a free open source software that can visualize GraphML files and more. An overview over the topologies used to evaluate the project is given in section 5.1.1.

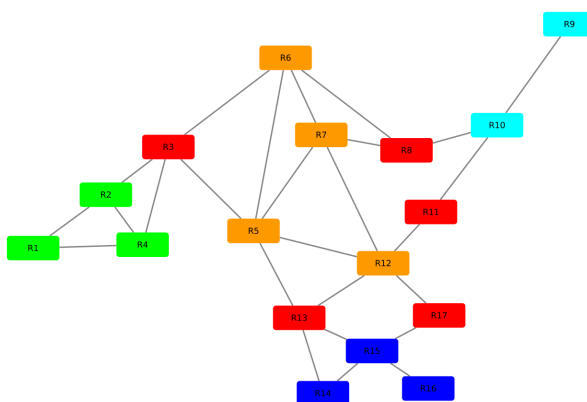
#### 5.1.1 Topologies

We are using eight different topologies to evaluate the ABR generation and area synthesis. Figure 5.1 shows five topologies that were taken from online tutorials and one additional topology we created ourselves. Additionally we took two topologies from the rocketfuel project [17] as shown in Figure 5.2. Note that we are only given the ABRs for `rocketfuel_small` and the backbone (and therefore we can conclude the ABRs) for `rocketfuel_big`.

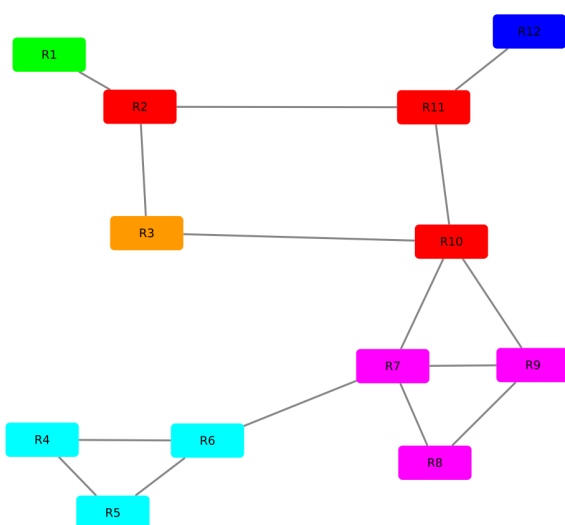
Except for heatmaps we always use the color red to mark ABRs and orange to mark backbone routers. Any other colors used represent the different non-backbone areas.



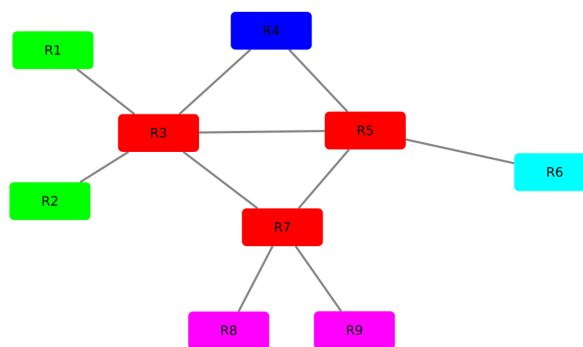
(a) tutorial\_1



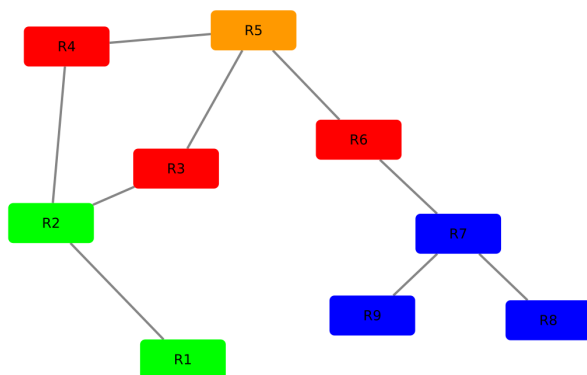
(b) tutorial\_2



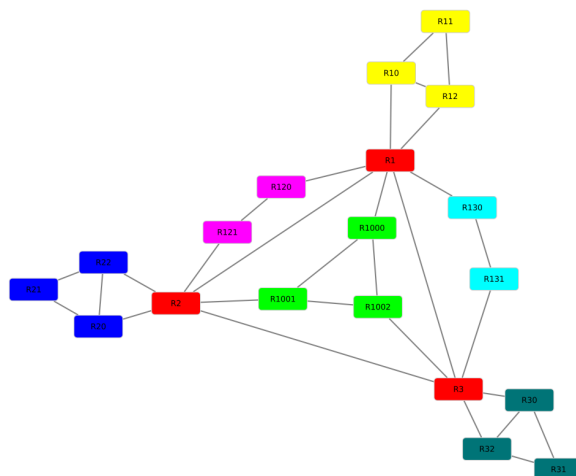
(c) tutorial\_3



(d) tutorial\_4

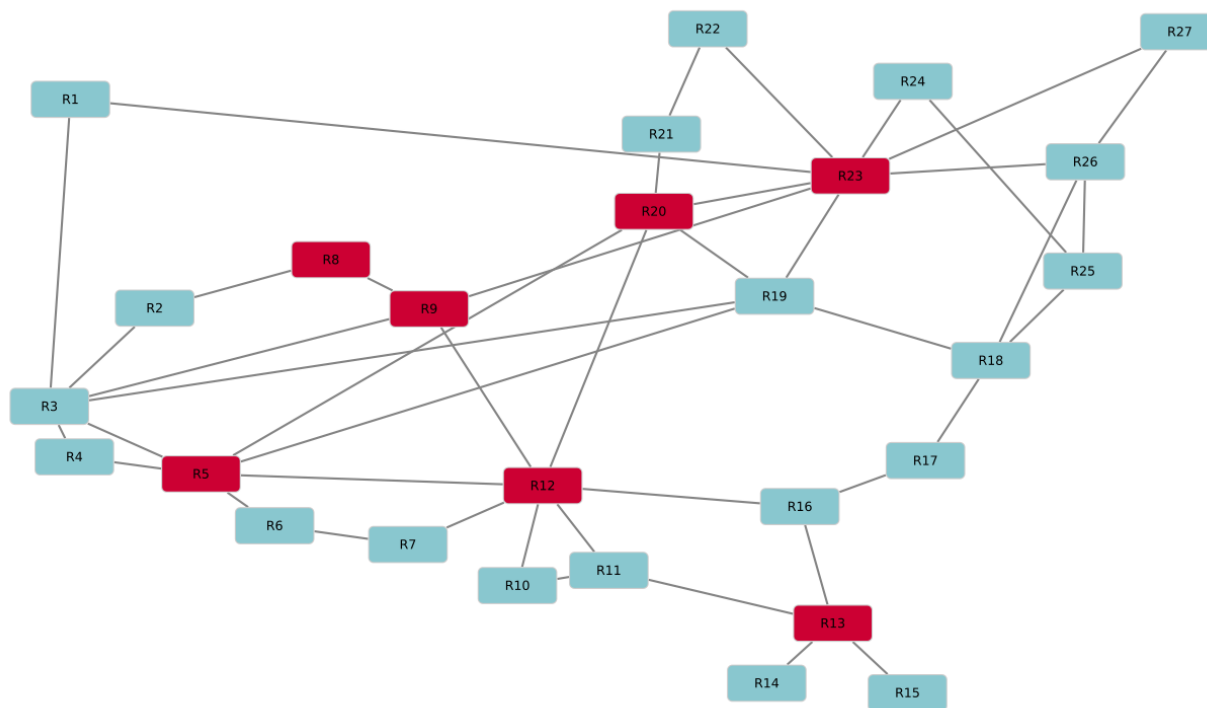


(e) tutorial\_5

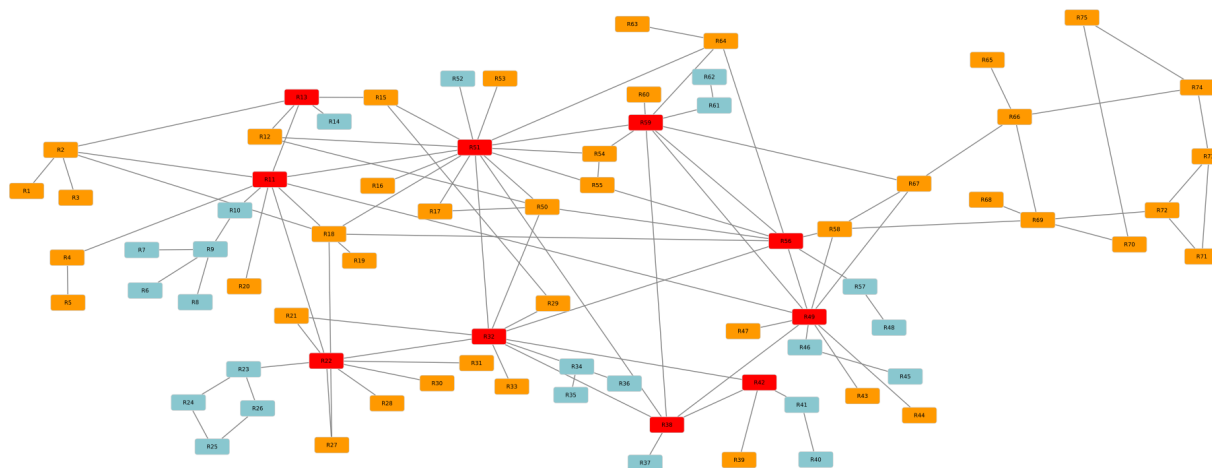


(f) custom\_1

Figure 5.1: Tutorial topologies



(a) rocketfuel\_small



(b) rocketfuel\_big

Figure 5.2: Rocketfuel topologies

## 5.2 IGP Additions

Since all IGP additions, specifically RIP, IS-IS and hierarchical OSPF checks and stubbiness configuration, are mainly based on the already existing code of the NetComplete project, we refrain from doing an extensive evaluation of this part of the thesis, as it can be found in the paper of NetComplete [14]. However, we provide a basic automated testing framework that verifies the correctness of the code on small example topologies and inputs.

For RIP and IS-IS additionally to checking the correctness for all types of path requirements, there is also a check in place that verifies that the synthesis fails if we specify paths that are too long. This means we fail in cases where there is a path cost of 16 or more for RIP and 1024 or more for narrow metric IS-IS. For the hierarchical OSPF restrictions we test if all restrictions fail the synthesis appropriately.

## 5.3 ABR Generation

In this section we evaluate the error of the ABR generation using all eight topologies previously described. The ground truth was shown in section 5.1.1 by the red marked routers.

### Setup

A few additional steps need to be taken to set up the environment for the measurement. The error we want to measure is defined in equation 5.1 as the sum of all ABRs that are chosen incorrectly and ABRs that were not chosen compared to the ground truth. It is measured before running the area synthesis and after the area synthesis is completed. This is important since the synthesis is only restricted to use at least the backbone area for an ABR. If no other area is assigned to the ABR it essentially loses the ABR functionality and becomes a pure backbone router. Therefore the area synthesis is able to remove ABRs.

$$error = (number\ of\ wrong\ ABRs) + (number\ of\ forgotten\ ABRs) \quad (5.1)$$

All measurements have been repeated five times due to the non-deterministic behavior of the Z3 SMT solver but no differences were measured. We configured a timeout for the measurement which aborts the synthesis after 20 minutes. All measurements were taken with the number of areas limited by the default value, given as `maximum_areas = min(number of nodes, 2*(number of ABRs))`. We also provide a heatmap for every topology to visualize the ABR selection. The heatmap shows us the routers with the highest immediate dominator count in red and the routers with the lowest immediate dominator count in blue with a gradual transition in between. As example the heatmap in Figure 5.3b shows *R4* and *R6* as the most red routers, meaning those two routers get selected as ABRs first. The slightly less red *R10* and *R14* follow next while blue routers like *R3* are chosen last.

### Results

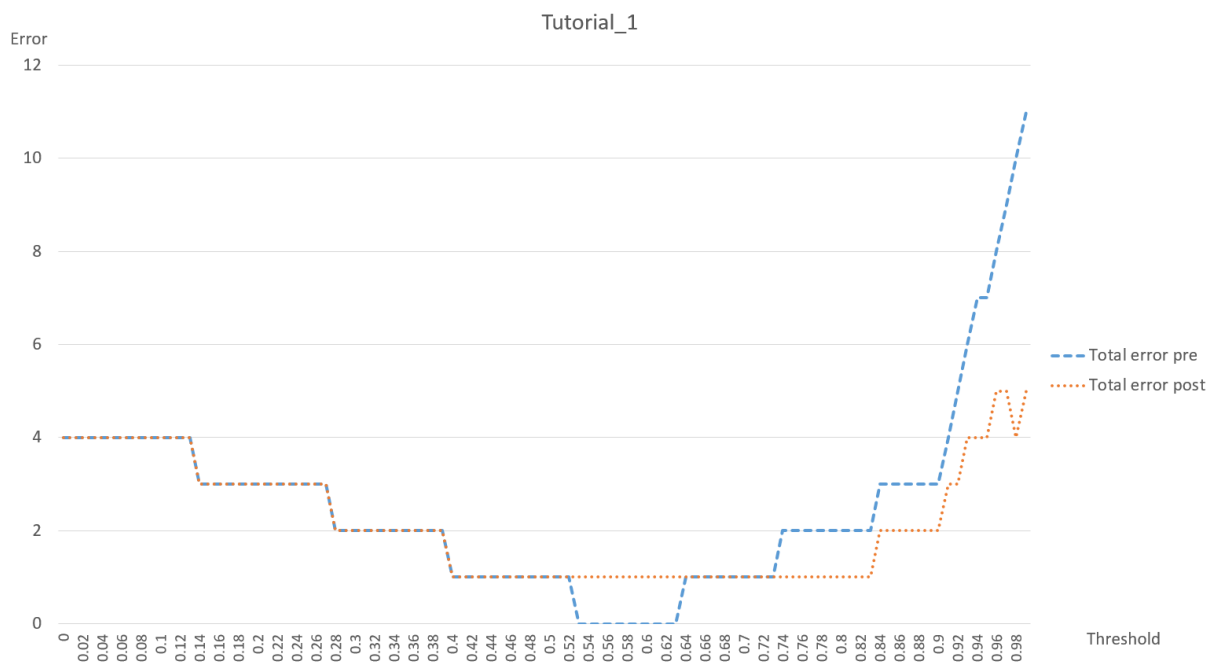
One interesting aspect can be observed best for `tutorial_2` (Figure 5.4), `tutorial_3` (Figure 5.5) and `rocketfuel_small` (Figure 5.9), where the post synthesis error is significantly lower than the pre synthesis error. This is a result of the mentioned fact that the area synthesis can remove ABRs by only assigning the backbone area to it. Generally speaking, the removal of ABRs during the synthesis is a great tool to keep the error small for big thresholds while not having an impact on small thresholds. Although there are cases as in `tutorial_5` (Figure 5.7), where we start off with

a bunch of wrong ABRs and we only start adding the correct ABRs around the threshold of 0.7 which leads to the decrease of the error for the pre synthesis results. In this example the area synthesis actually removes correct ABRs, keeping the error higher compared to the pre synthesis. This shows us that the synthesis does not exclusively remove wrong ABRs and therefore it is not beneficial for all cases.

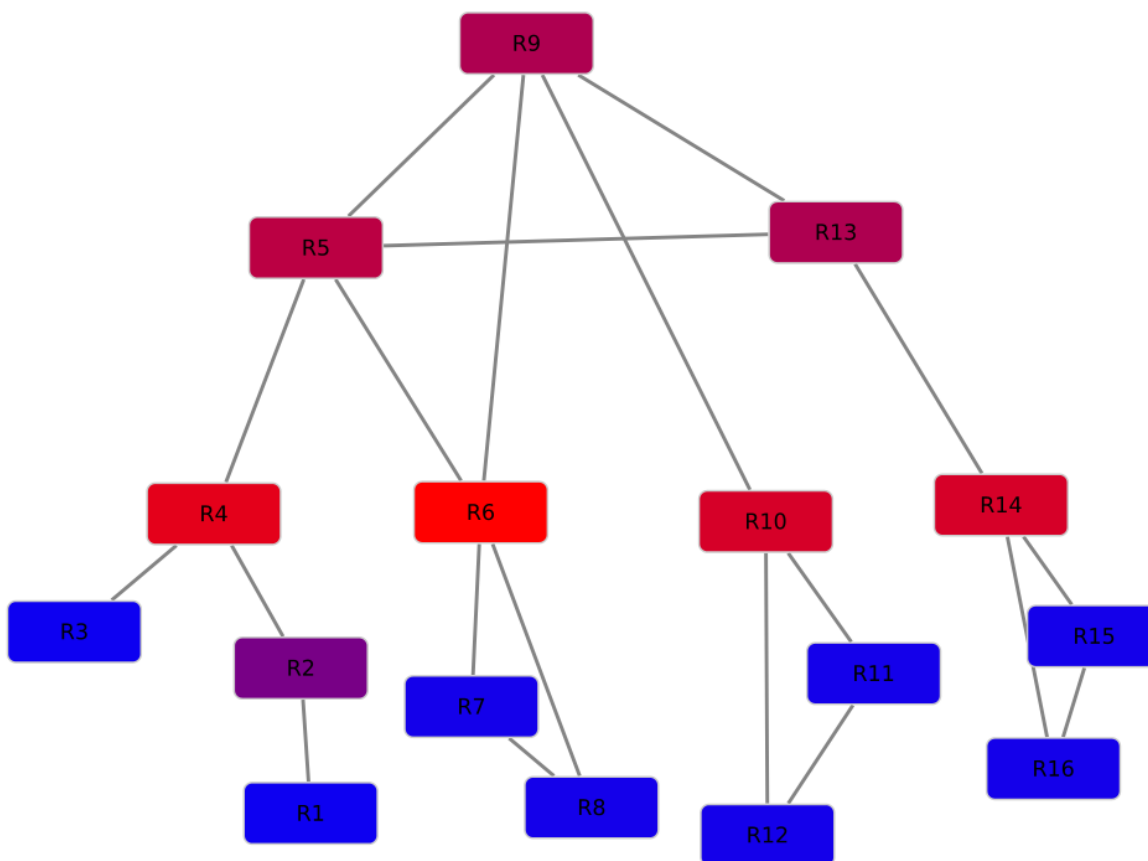
Topology tutorial\_5 (Figure 5.7) also shows a great example where the ABR synthesis generally performs bad compared to the ground truth, having a minimum error of around 50% of all nodes in the topology for most thresholds. This is generally the case in scenarios where multiple ABRs split an area off, such as *R3* and *R4* splitting off *R1* and *R2* in the ground truth. Since our algorithm for the ABR generation considers immediate dominators, we primarily detect single routers splitting off areas, such as *R5*, dividing the topology in the middle, or *R7*, splitting off *R8* and *R9*.

Lastly we also had a look at the minimum error for each topology after the area synthesis and combined them together in Figure 5.11. We want to use this insight to determine whether there is a best threshold that produces the minimum error for the most topologies. Again we see that higher thresholds are generally better, reaching the minimum error for three out of 8 topologies around the thresholds 0.46 and 0.75. When adjusting the number of areas available to the synthesis to ten, such that the topology *rocketfuel\_big* does not time out for thresholds bigger than 0.6, the peak around 0.75 in Figure 5.11b extends a bit further to the left, including thresholds 0.67 and 0.68. Based on the facts that the minimum error peaks around a threshold of 0.7 and it is a more stable peak with less drop to both sides than the peak around 0.45, we decided to use a default threshold of 0.7 in cases where the user does not specify any threshold value himself.



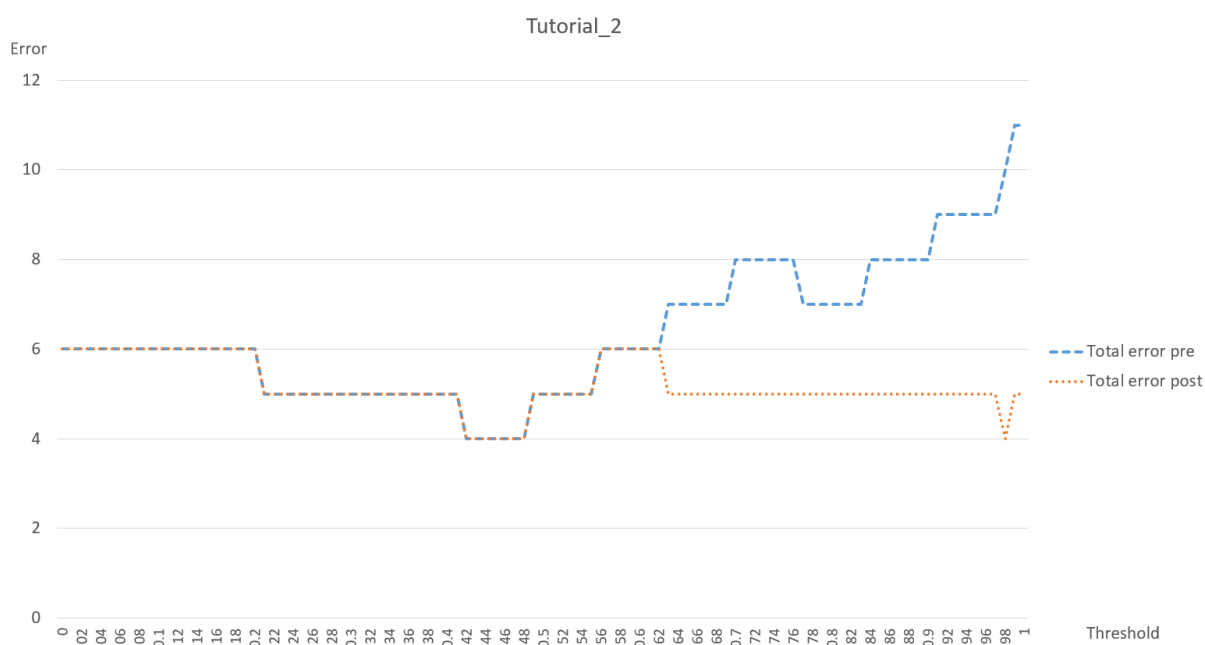


(a) Threshold error pre synthesis vs post synthesis

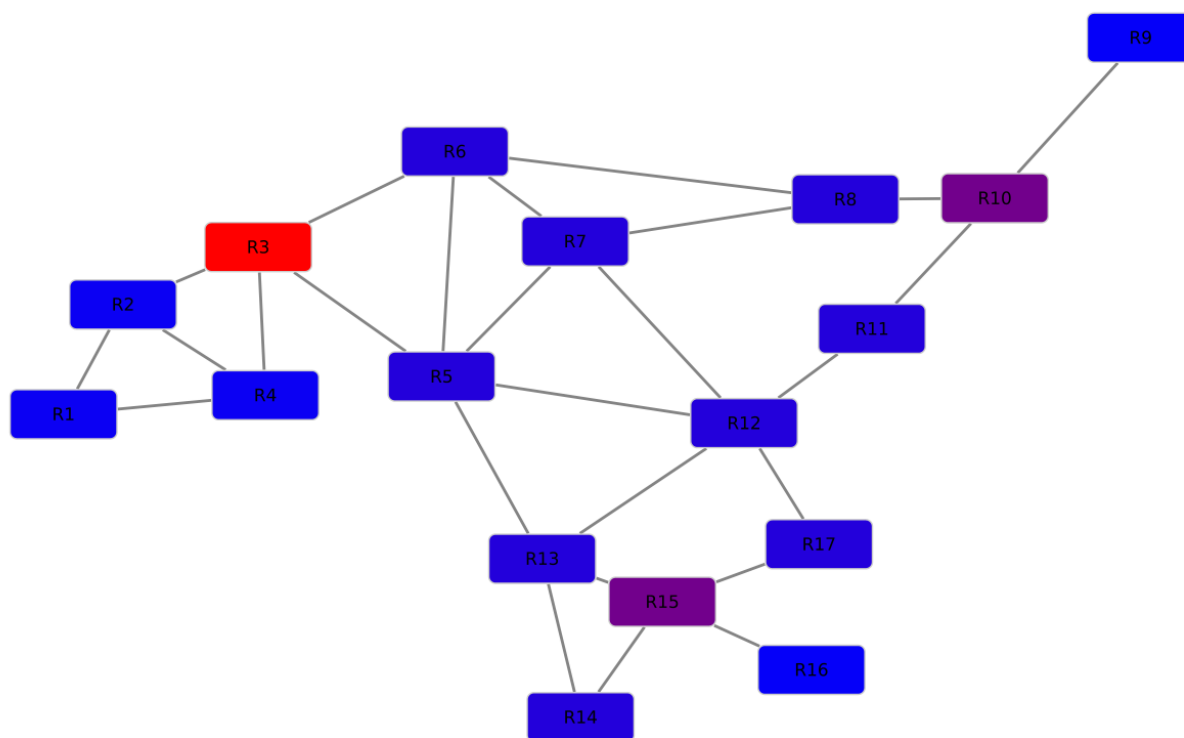


(b) Heatmap

Figure 5.3: ABR generation error tutorial\_1



(a) Threshold error pre synthesis vs post synthesis



(b) Heatmap

Figure 5.4: ABR generation error tutorial\_2

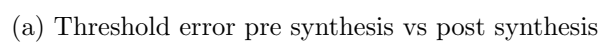
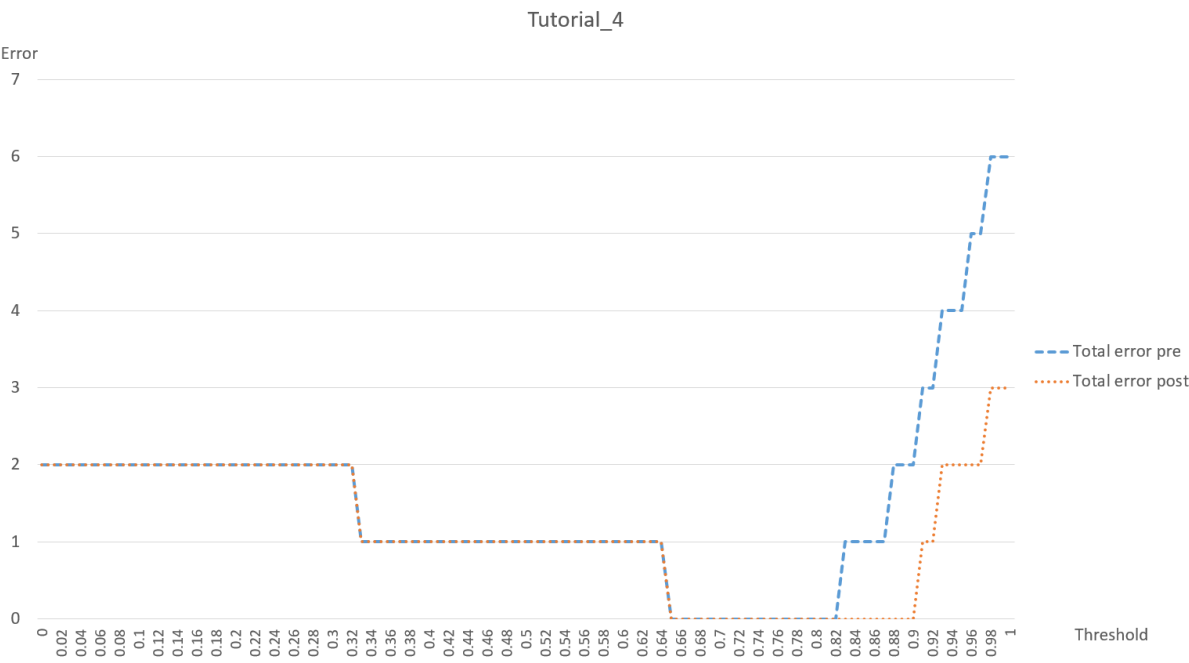
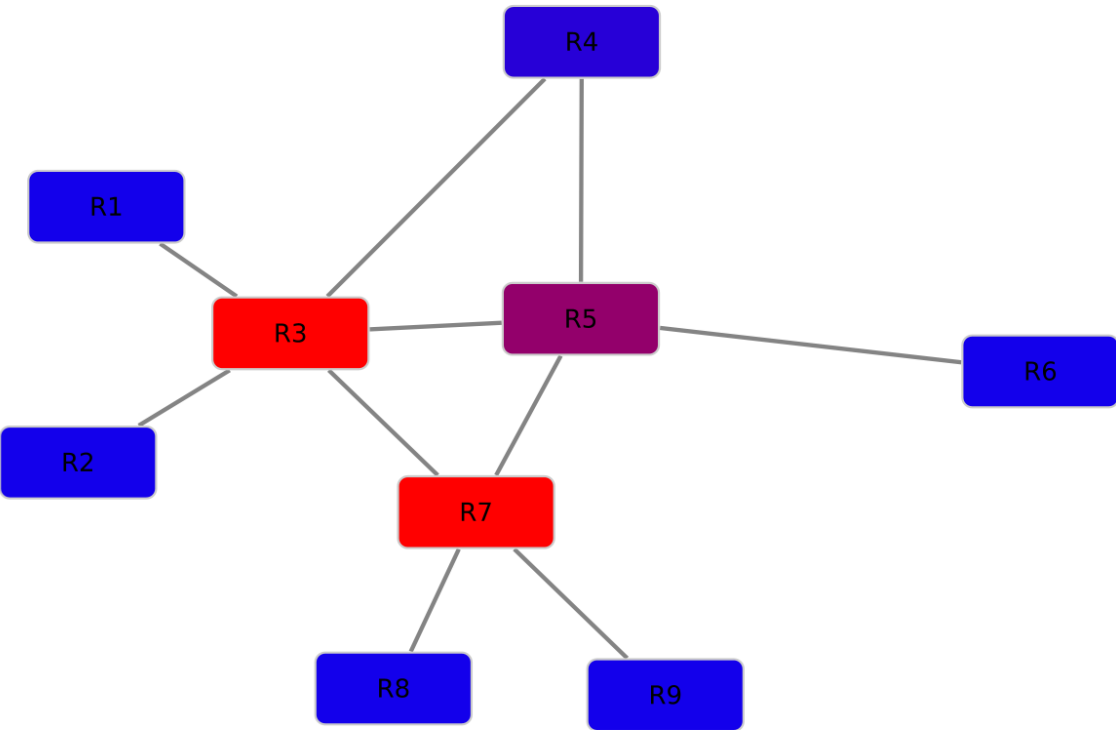


Figure 5.5: ABR generation error tutorial\_3

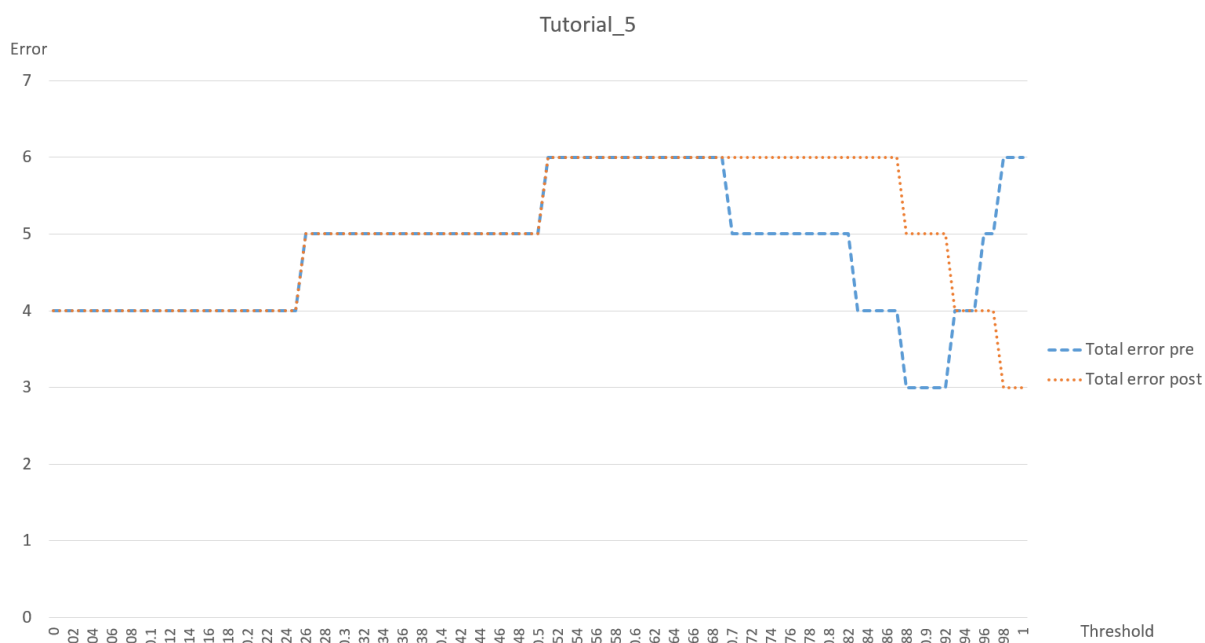


(a) Threshold error pre synthesis vs post synthesis

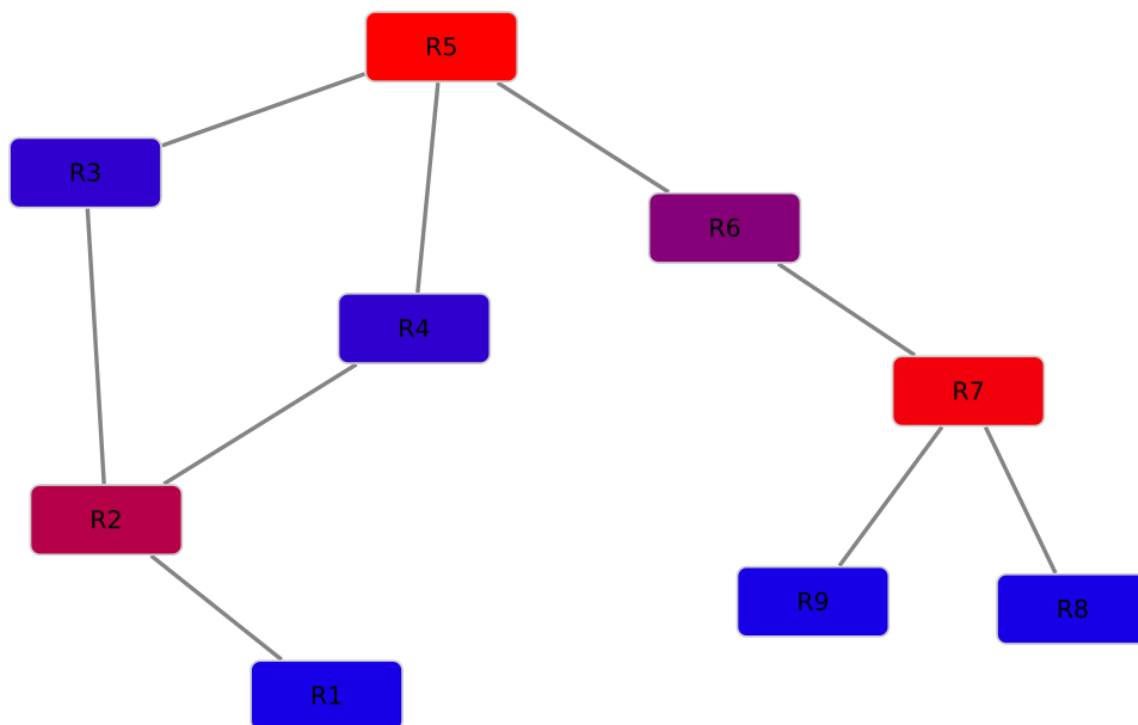


(b) Heatmap

Figure 5.6: ABR generation error tutorial\_4

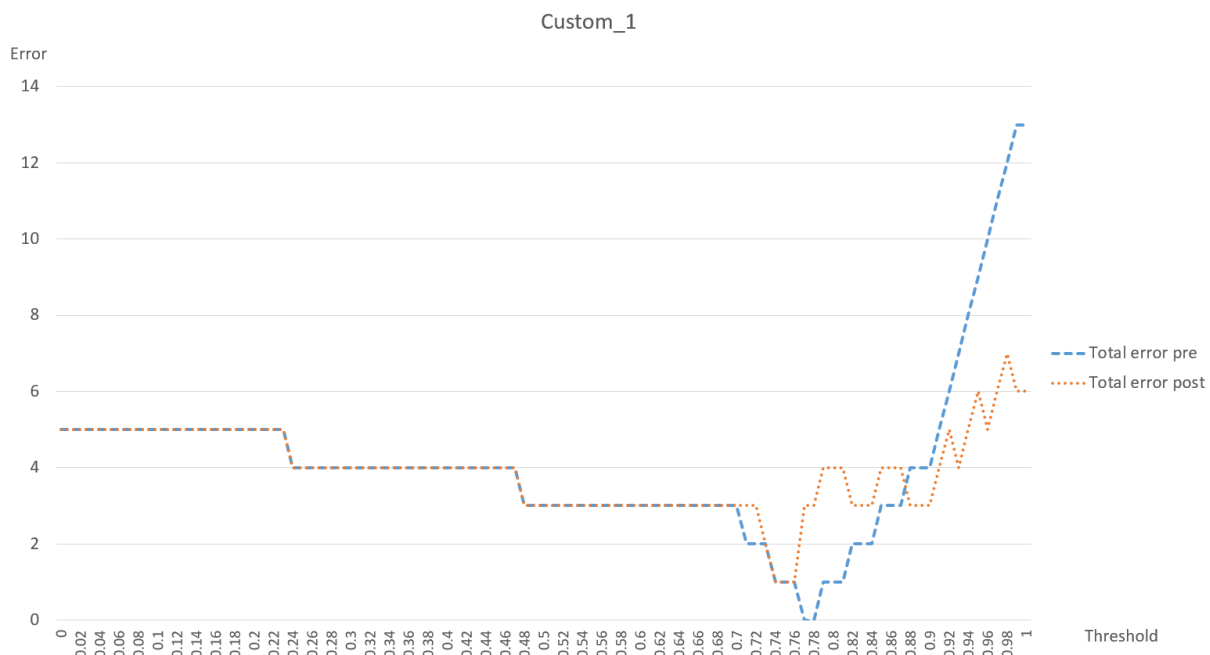


(a) Threshold error pre synthesis vs post synthesis

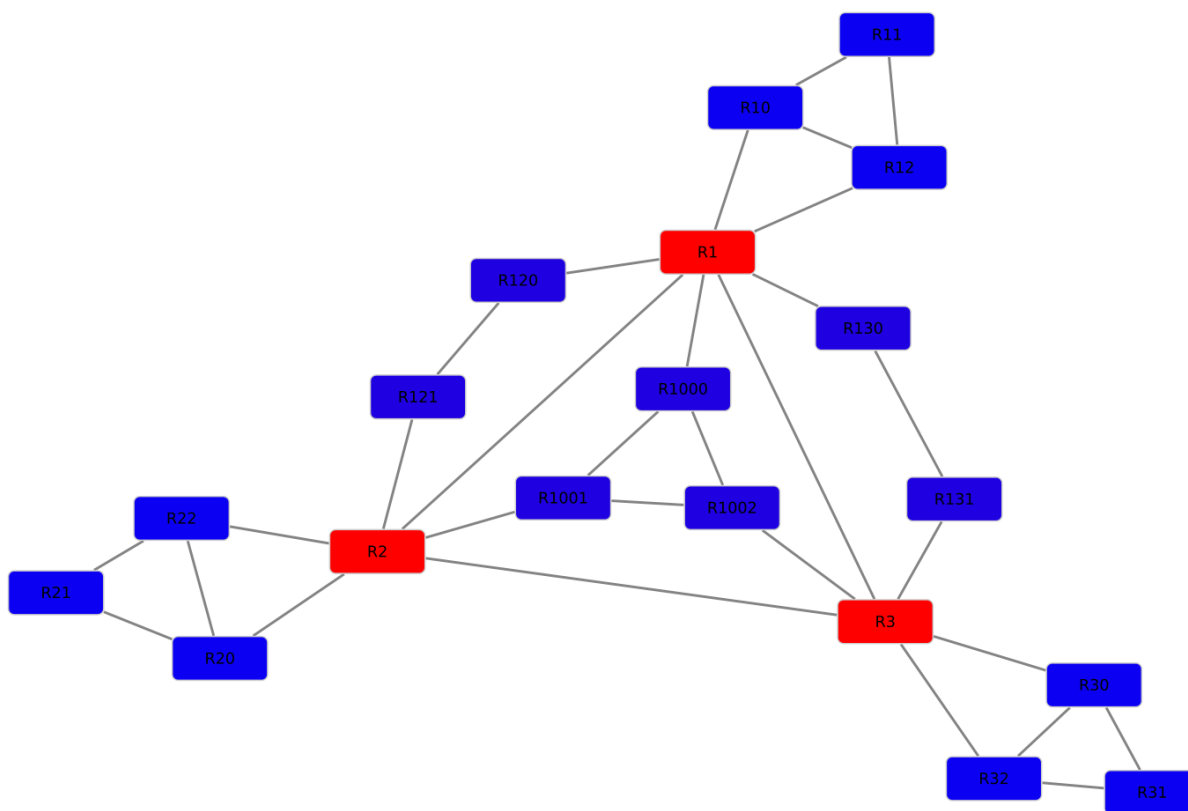


(b) Heatmap

Figure 5.7: ABR generation error tutorial\_5

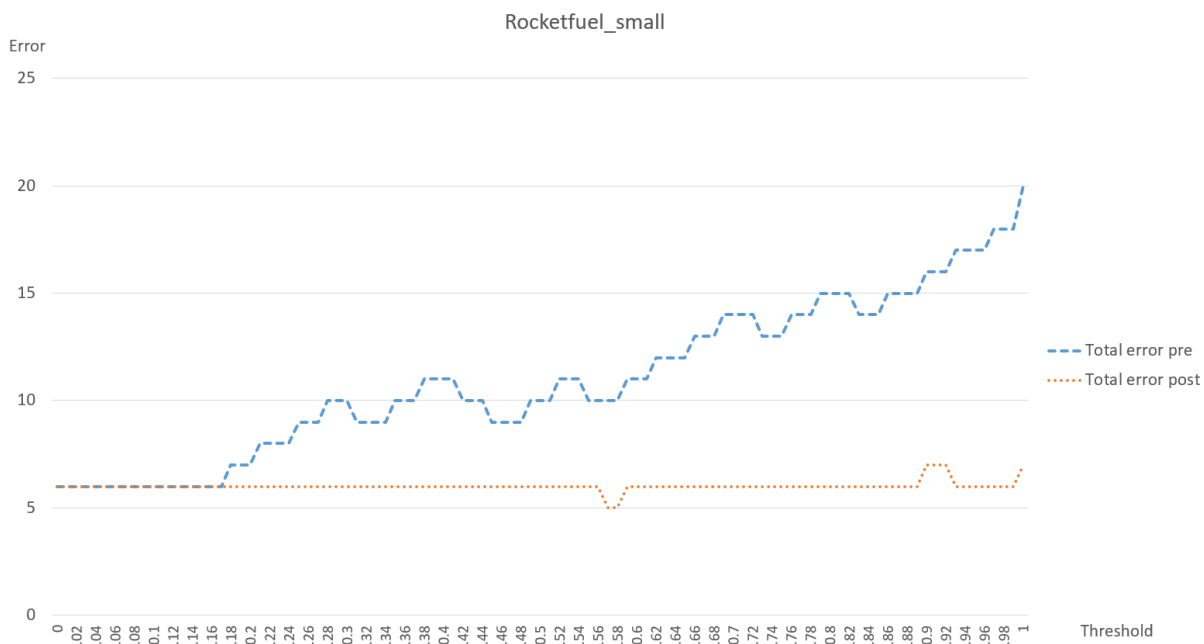


(a) Threshold error pre synthesis vs post synthesis

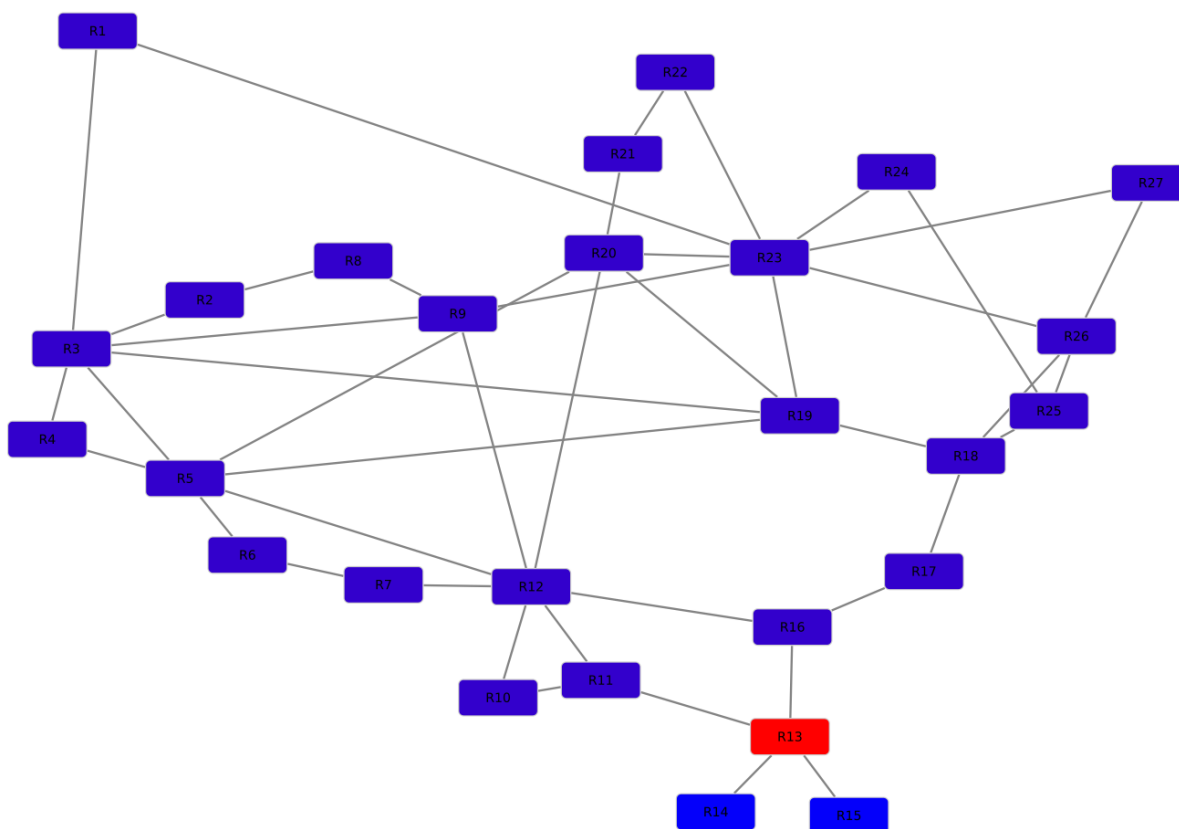


(b) Heatmap

Figure 5.8: ABR generation error custom\_1

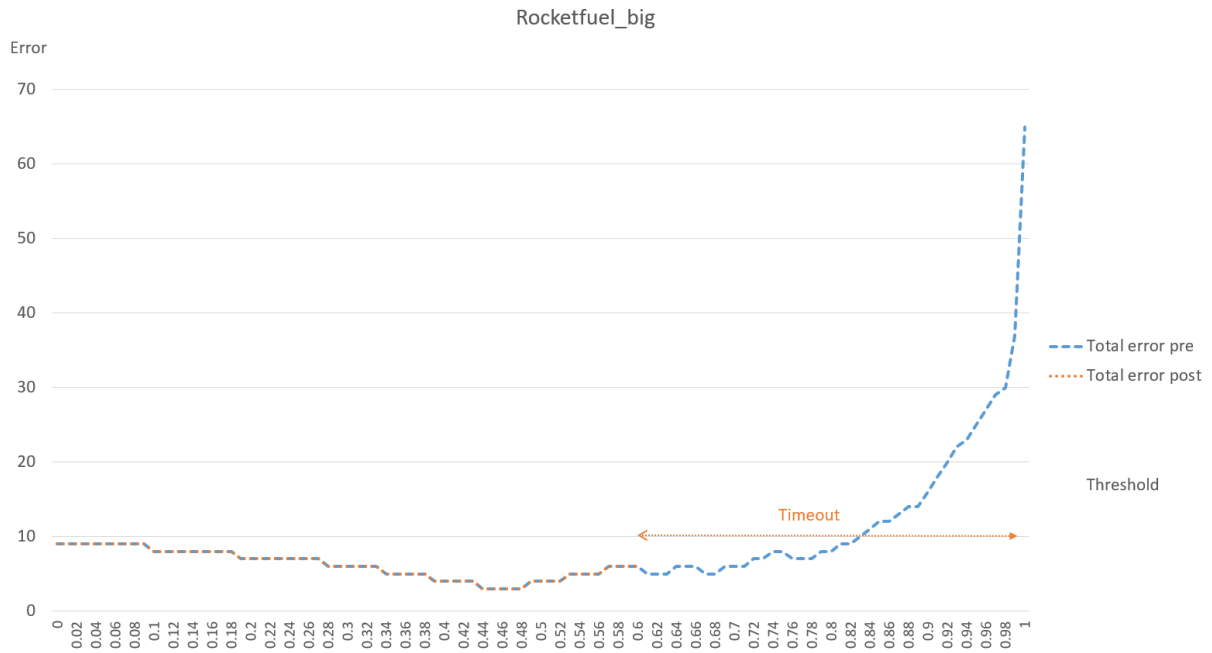


(a) Threshold error pre synthesis vs post synthesis

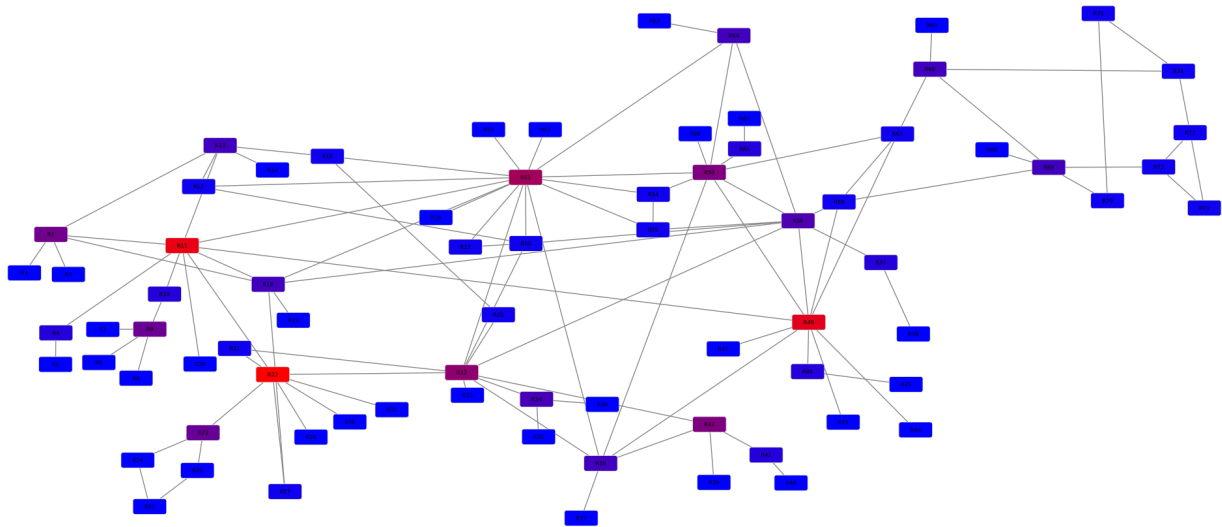


(b) Heatmap

Figure 5.9: ABR generation error rocketfuel\_small



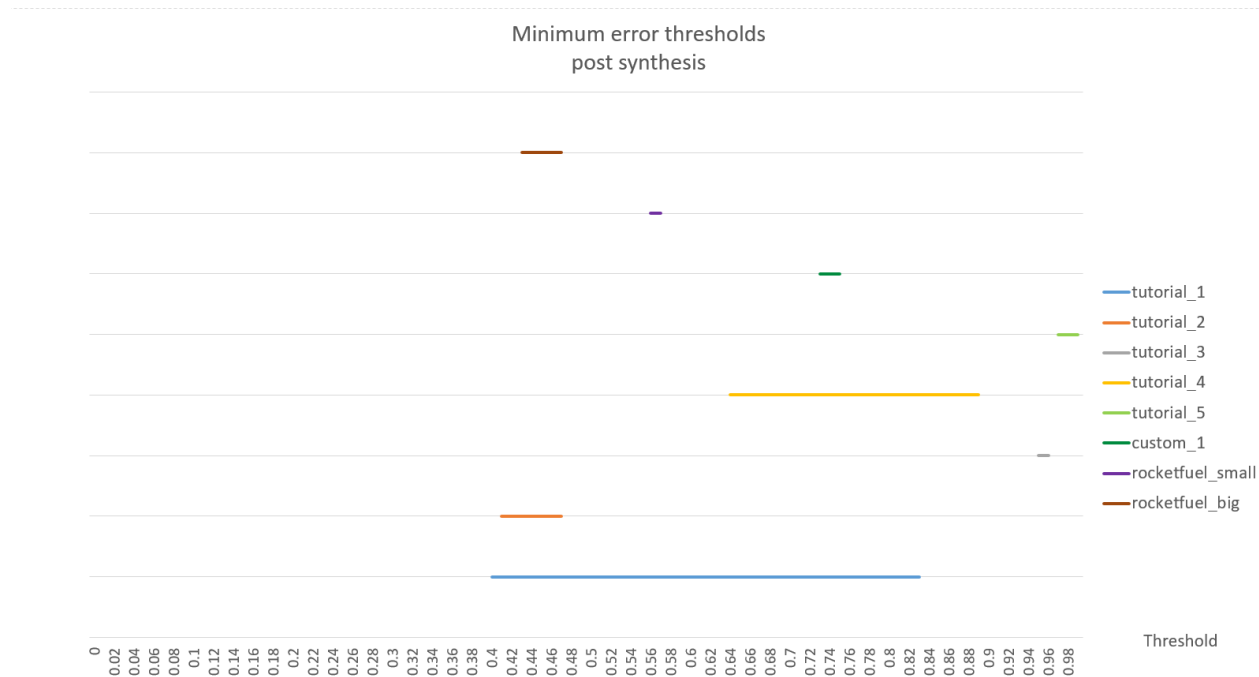
(a) Threshold error pre synthesis vs post synthesis



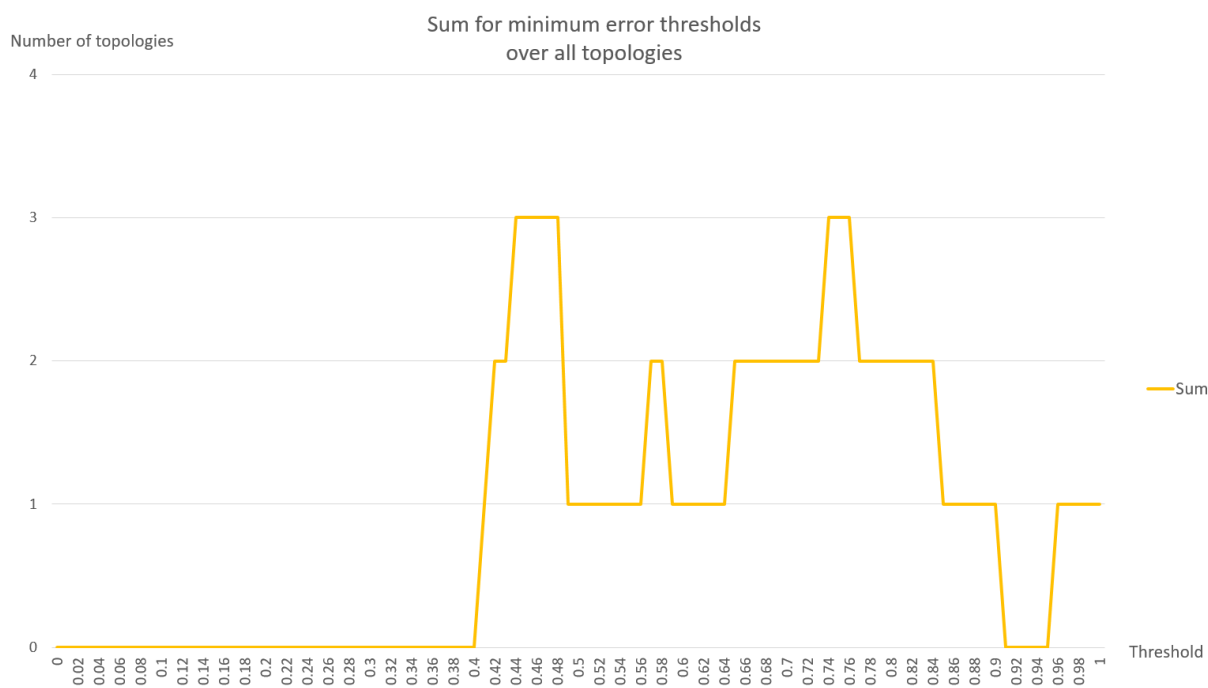
(b) Heatmap

Figure 5.10: ABR generation error rocketfuel\_big





(a) Minimum error for each topology



(b) Minimum error summed up for all topologies

Figure 5.11: Minimum error

## 5.4 Area Synthesis Reliability

In this section we want to evaluate the similarity of our area synthesis to a ground truth. This evaluation is important, as we want to make sure we produce results that are close to what the user expects. Furthermore we want the user to understand the limitations of the area synthesis.

### Setup

We will only utilize the synthesis maximizing the number of areas and compare three different input cases:

- Topology and ABR list (ABR)
- Topology and optimal threshold value for specific topology (threshold)
- Topology and default threshold value (default)

We will only consider the five tutorial topologies for this evaluation, as they are the only ones where we have a ground truth including all the individual areas. Lastly we will also provide insights into a limitation that has to be considered when working with big topologies.

### Tutorial\_1

Topology tutorial\_1 (Figure 5.12) produces the same output for all three input cases, whereas the only difference is the area assignment of router  $R3$ . This behavior can easily be explained by the fact that we are maximizing the amount of areas and ABR  $R4$  only has the backbone area and one non-backbone area assigned, without considering  $R3$ . Since  $R3$  has no connection to neither  $R1$  nor  $R2$ , the SMT solver is free to assign it a separate area.

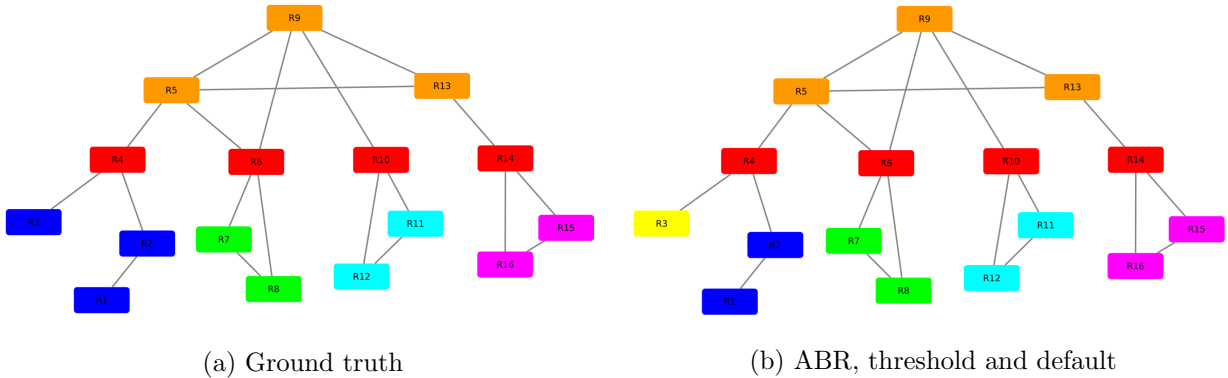


Figure 5.12: tutorial\_1 area synthesis

### Tutorial\_2

Topology tutorial\_2 (Figure 5.13) produces a slightly different result for the ABR input by assigning  $R10$  and  $R9$  to the backbone and therefore removing the ABR functionality of  $R8$  and  $R11$ . This behavior can be explained when considering the area connectivity. Since  $R8$  and  $R11$  need to have a connection over the backbone, constraints are added making sure that there exists a connection over one of the shortest paths. As there is only one shortest path going from  $R8$  to  $R10$  to  $R11$ ,

$R_{10}$  will be assigned to the backbone. Since  $R_9$  is directly connected to the non-ABR  $R_{10}$ , it will share the same area and is assigned to the backbone as well. When looking at the threshold and default inputs we can see that ABRs  $R_8$  and  $R_{11}$  are replaced by  $R_{10}$  and ABRs  $R_{13}$  and  $R_{17}$  are replaced by  $R_{15}$ . As already mentioned in section 5.3, our method of generating ABRs prefers routers that split the topology by themselves, instead of picking the router pairs, explaining this behavior.

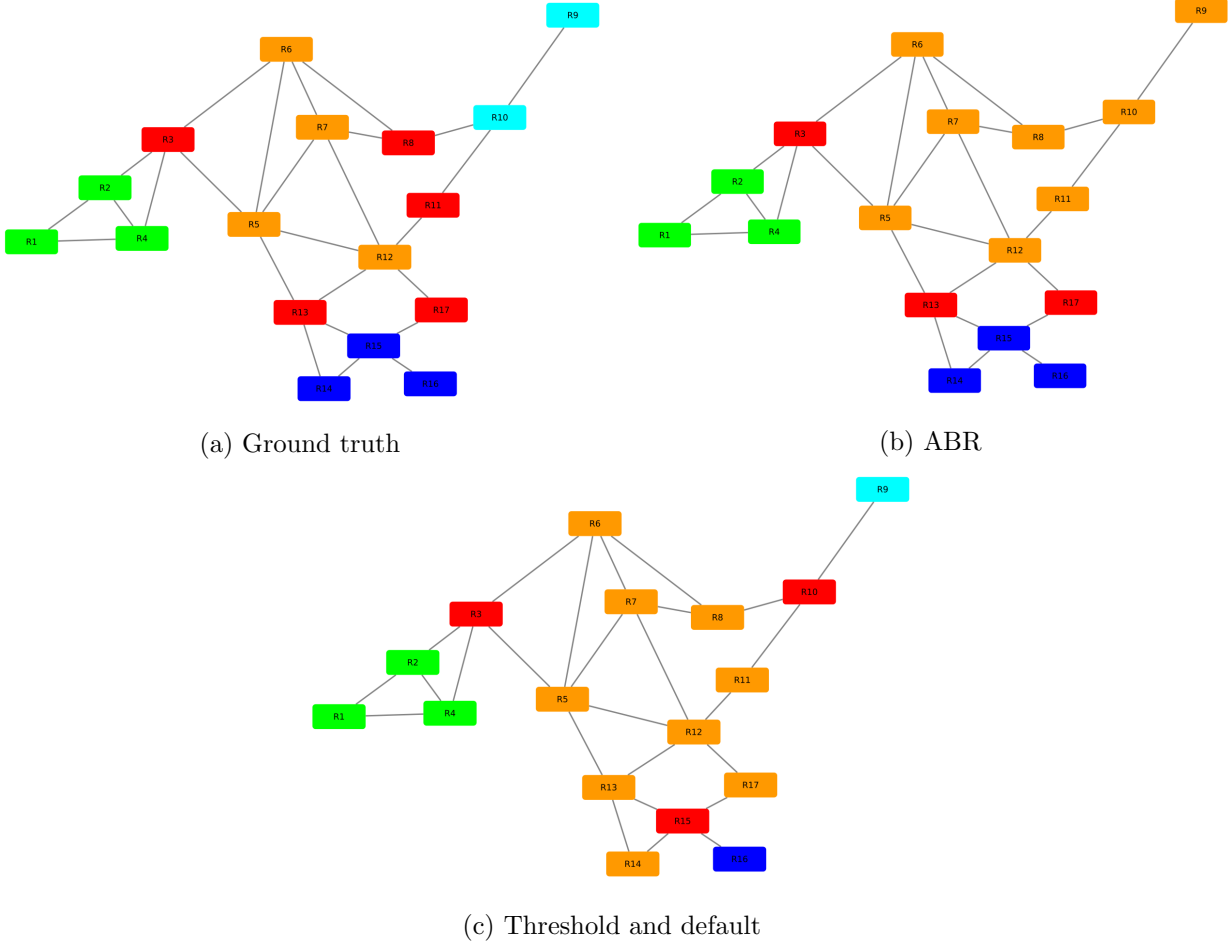


Figure 5.13: tutorial\_2 area synthesis

### Tutorial\_3

Topology tutorial\_3 (Figure 5.14) is a special case since the area on the bottom left, consisting of  $R_4$ ,  $R_5$  and  $R_6$ , is connected to the backbone using a virtual link. Our synthesis does not consider virtual links to be an option. Therefore when giving the ABRs as input, it will assign the aforementioned area to the same area as  $R_7$ ,  $R_8$  and  $R_9$ .  $R_3$  is not assigned to the backbone, as all three ABRs already satisfy the connectivity constraints through each other and both,  $R_2$  and  $R_{10}$ , have only two areas assigned when not considering  $R_3$ . Therefore  $R_3$  can be assigned to a non-backbone area to maximize the amount of areas. For the optimal threshold, which is 0.96 or 0.97, we essentially add all routers except  $R_1$  and  $R_{12}$  to the ABR selection. The area synthesis converts all ABRs except for  $R_2$  and  $R_{11}$  back to backbone routers. This leaves the error minimal

as only  $R_{10}$  is missing, although we can barely utilize the advantages of hierarchical OSPF at this point. For the default threshold we produce a higher error in regards to the ABR generation due to the previously mentioned behavior of prioritizing single routers that split off the largest areas in our ABR generation algorithm. It prioritizes  $R_6$  and  $R_7$  over  $R_{11}$  since both split off more routers. However, despite choosing different ABRs, it manages to produce the same areas and reduces the size of the backbone compared to the optimal threshold. In this case either  $R_3$  or  $R_{11}$  and  $R_{12}$  can be assigned to the backbone to ensure the connectivity of the backbone, which is randomly decided by the SMT solver.

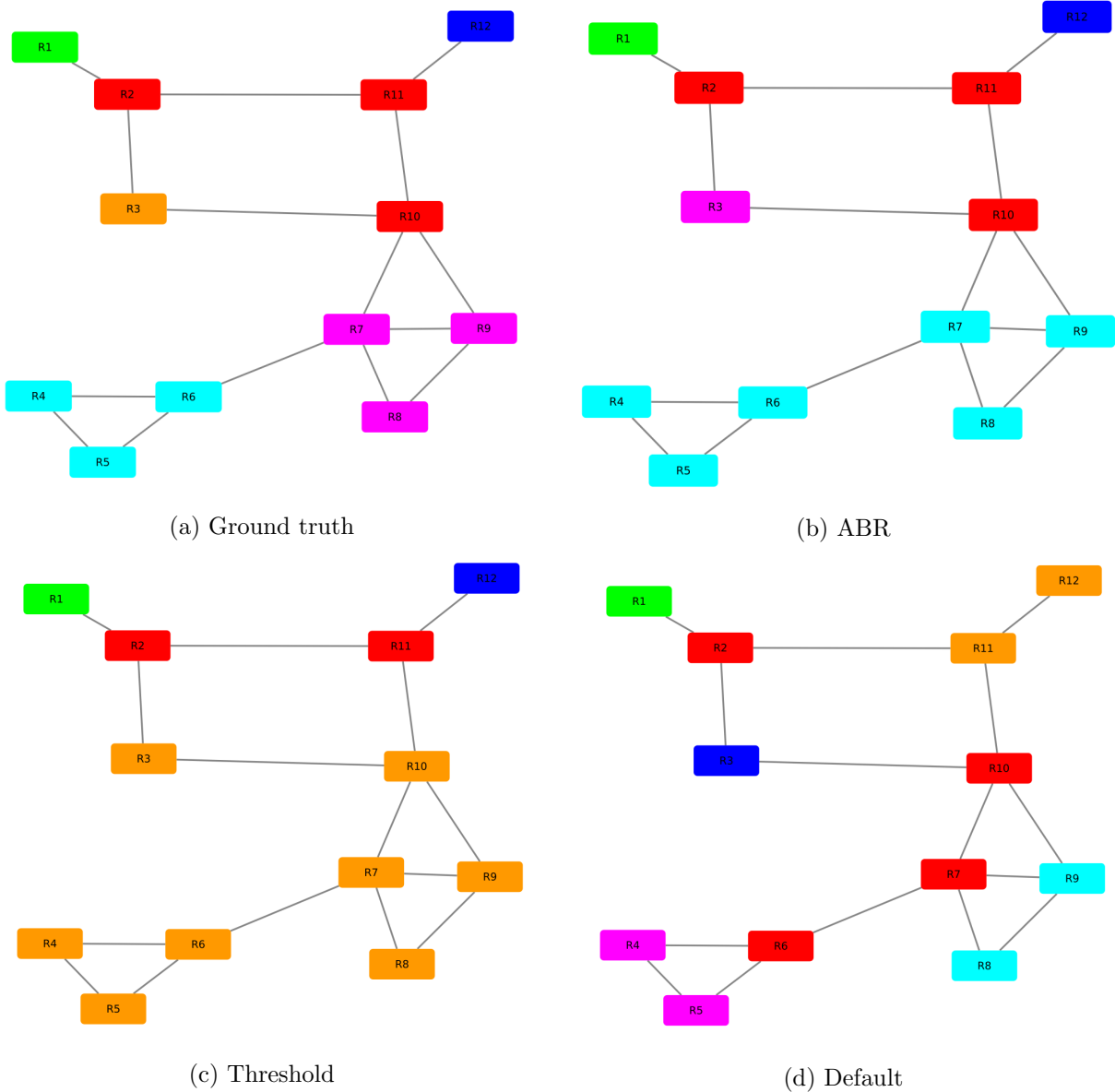


Figure 5.14: tutorial\_3 area synthesis

### Tutorial\_4

Topology tutorial\_4 (Figure 5.15) produces, similar to topology\_1, nearly the same area assignment as in the ground truth when the ABR list is given as input. The only difference again coming from the fact that we try to maximize the number of areas, resulting in different areas for  $R1$  and  $R2$  as well as  $R8$  and  $R9$ . Note that  $R2$  is assigned to the backbone due to the fact that ABR  $R3$  already has three areas assigned to it, due to it being an ABR and being neighbors with  $R1$  and  $R4$ . The only option for  $R2$  is to either get assigned to the backbone area or the same area as  $R1$  or  $R4$ , which is randomly decided by the SMT solver. The random assignment can be observed when comparing to the threshold and default inputs, where  $R4$  is assigned to the backbone instead of  $R2$ .

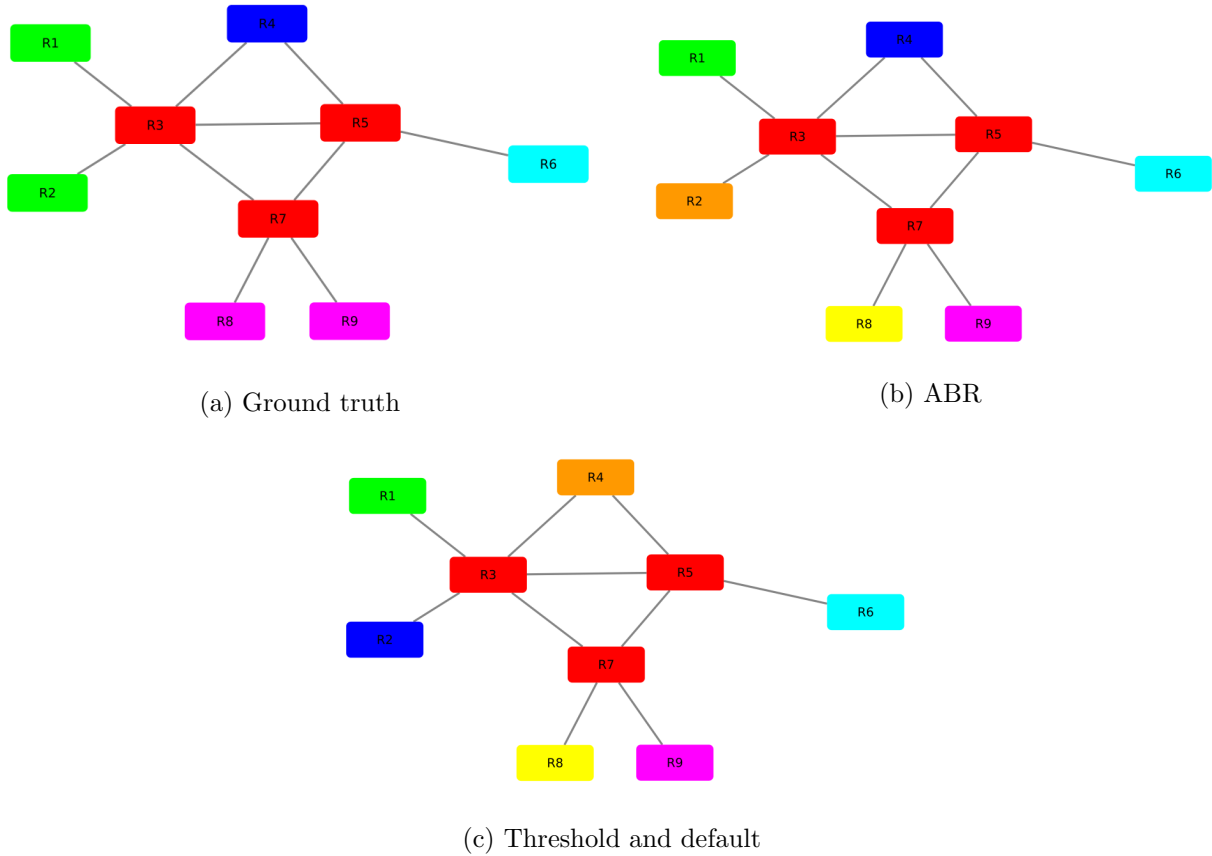


Figure 5.15: tutorial\_4 area synthesis

### Tutorial\_5

Topology tutorial\_5 (Figure 5.16) matches the ground truth when taking the ABRs as input, since  $R5$  needs to be assigned to the backbone to ensure the area connectivity and both non-backbone areas are connected in itself. When generating the ABRs for the optimal threshold and the default threshold we can again observe that routers which disconnect the topology are preferred over a pairs of routers, such as  $R3$  and  $R4$  in the ground truth. For the second time we can observe the randomness when breaking ties when looking at  $R3$  and  $R4$  for the threshold and default inputs, as both of them can be used to connect  $R2$  to  $R5$  over the backbone.

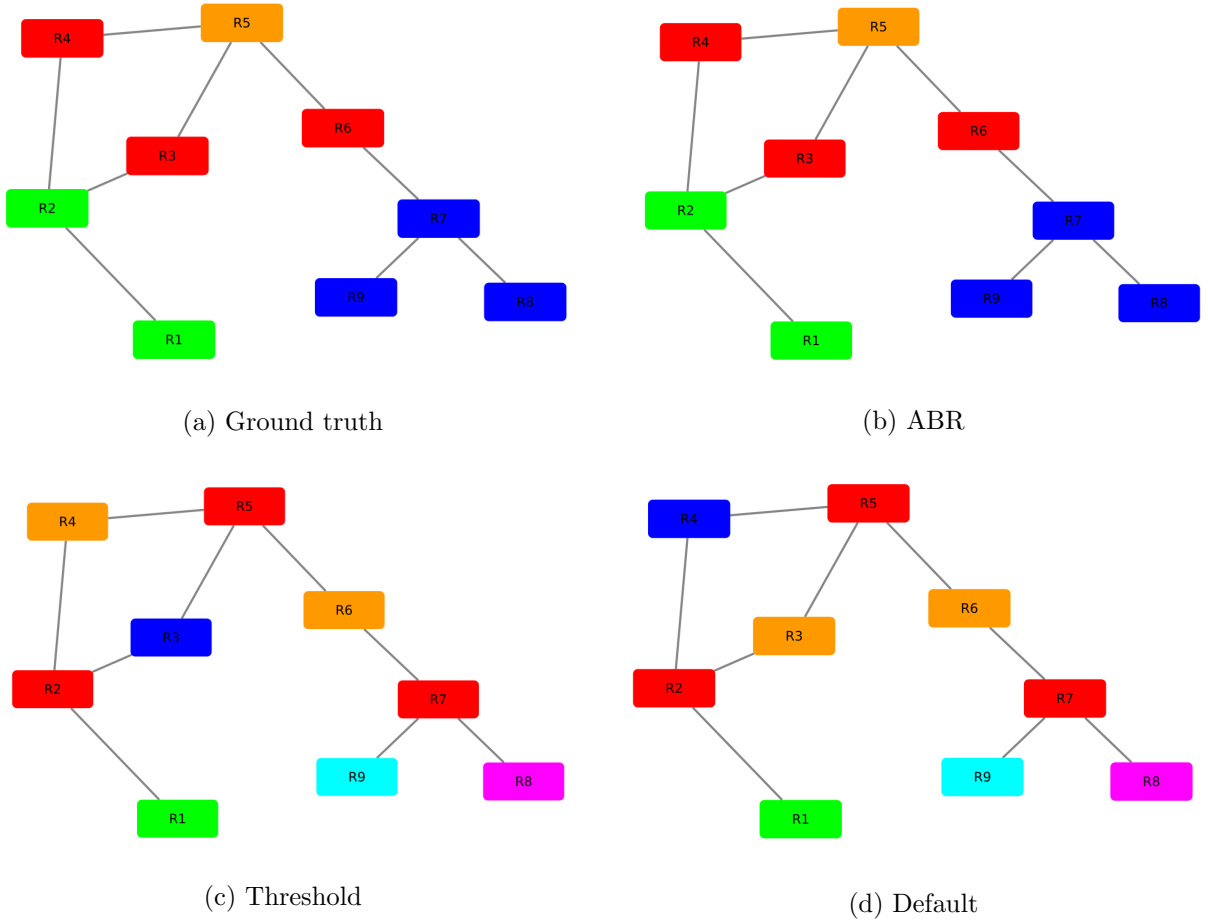
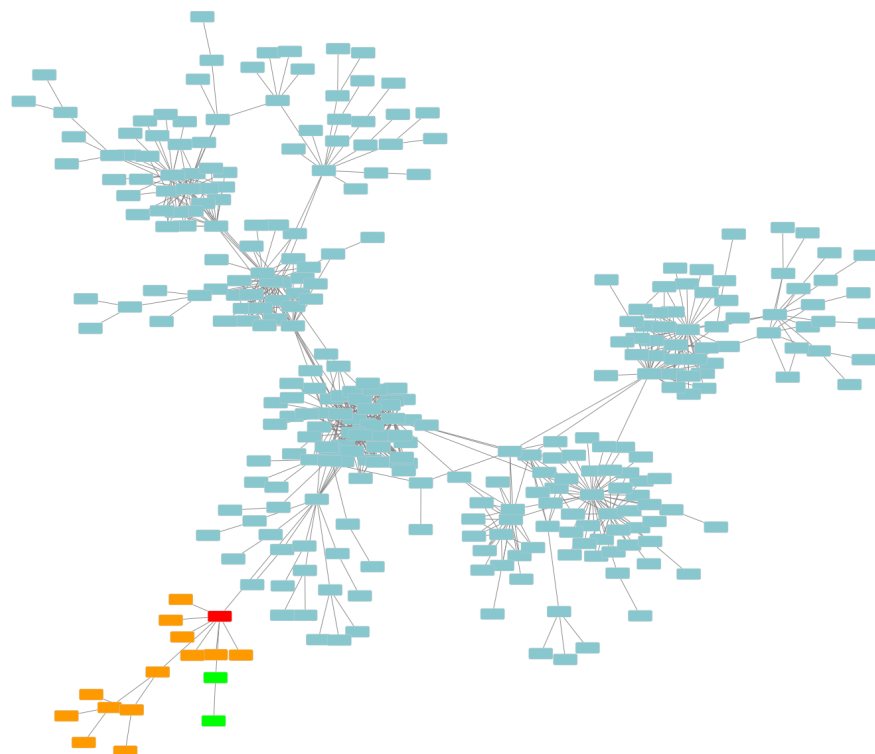


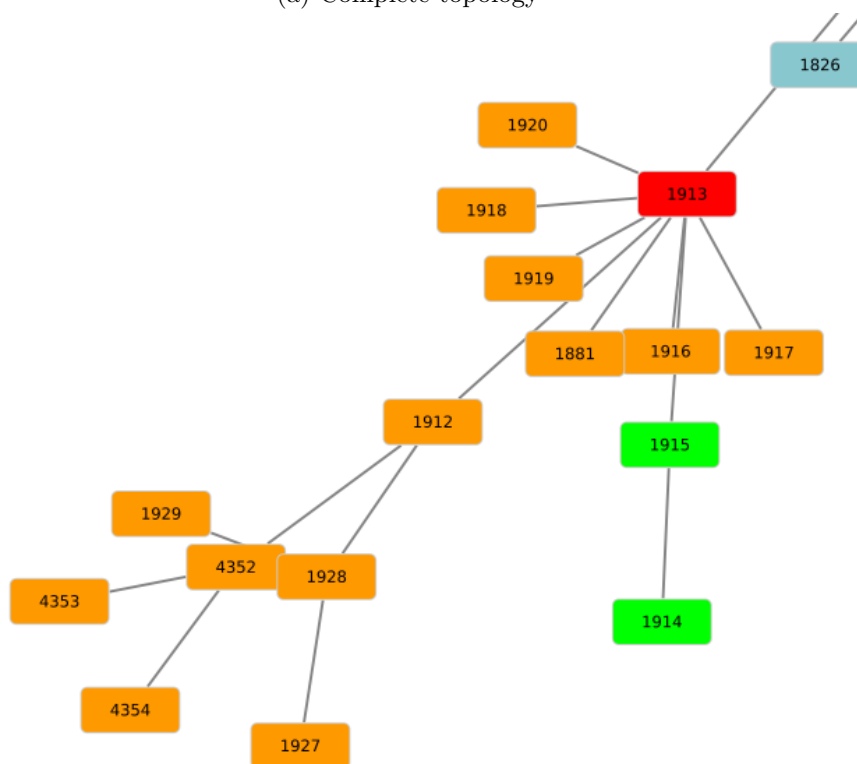
Figure 5.16: tutorial\_5 area synthesis

### Big Topologies

An interesting insight can be gained when looking at bigger topologies, such as `rocketfuel_1221_r0_cch` (Figure 5.17) with 318 routers. The zoomed in version shows an area assignment that was generated with a limitation of ten areas, which explains why only one non-backbone area is assigned. However the synthesis only assigned routers 1915 and 1914 to the non-backbone area and all other routers to the backbone. While this is indeed a valid assignment that maximizes the number of areas, we would generally expect all routers to be placed inside the non-backbone area, which would be another valid assignment. However the area synthesis only ensures that non-backbone areas have routers assigned to them that are no ABRs. As consequence both mentioned assignments are valid and equally optimal, so the SMT solver chooses at random. From the overall nine non-ABR areas a total of five areas have five or less routers assigned to them, including the ABRs. The bigger four areas have 16, 17, 20 and 31 routers assigned to them. Including ABRs a total of 252 routers are assigned to the backbone while 100 routers are assigned to non-backbone areas. Note that ABRs have multiple areas assigned, thus the sum of backbone and non-backbone routers is greater than the number of routers in the topology.



(a) Complete topology



(b) Zoomed in on bottom left corner

Figure 5.17: rocketfuel\_1221\_r0\_cch

## 5.5 Area Synthesis Performance

In this section we will evaluate the performance of the whole area synthesis, starting from the ABR generation and the constraint generation, finishing at the synthesis. We will evaluate different tuning parameters, such as the maximum amount of areas that can be used in section 5.5.1, the size of the topology in section 5.5.2, the threshold given to the ABR generation in section 5.5.3, the shape of the topology in section 5.5.4 and lastly giving manual area inputs for some routers in section 5.5.5.

### 5.5.1 Number Of Areas Available To The Synthesis

One user input to the synthesis is the number of areas that can be assigned. To test the impact of the maximum amount of areas, we measured the runtime of the ABR generation, the generation of the general area synthesis constraints excluding the area connectivity, the generation of the area connectivity constraints and lastly the synthesis time.

#### Setup

We ran the performance measurement over three different configurations, once limited by a maximum of five areas, once limited by a maximum of ten areas and once limited by the default given by `maximum_areas = min(number of nodes, 2*(number of ABRs))`. We ran each measurement 50 times and took the average over all runs. A timeout was configured to abort the synthesis after 20 minutes. We utilized the ABR generation with the default threshold of 0.7 as input. Note that the y-axis is individually scaled to the measurement for each topology.

#### Results

A general observation that can be made for all topologies is the fact that the runtime increases proportional to the number of areas for both, the general constraint generation and the connectivity constraint generation. This behavior is expected, as nearly all constraints have to be generated for every area. Furthermore the ABR generation only takes up a small amount of the total runtime. This generally the case as the ABR generation is a simple algorithm that does not utilize the SMT solver and does not need to generate any logical constraints. Additionally it is independent of the amount of areas.

When we observe the runtime for the actual synthesis part, we notice that it generally starts relatively low and explodes with the number of areas available. This is expected as the increase of the number of areas adds a lot more constraints and greatly increases the search space for solutions. The only exception to this behavior is `rocketfuel_small`, which has a fairly low synthesis time even when using the default amount of areas. Limited by the number of routers in the topology, we have a total of 27 areas available. If we consider the ABR input, shown in Figure 5.26, we can quickly see why the increase of the runtime does not occur. Only a small subset of nodes is not selected as ABR and they can only group to a total of three different areas, greatly limiting the search space and therefore the synthesis time.

Additionally a drastic increase in the synthesis runtime can sometimes be observed. Topology `rocketfuel_big` shows this best. For area limitations up to 24 the synthesis runtime slowly increases to a maximum of around one second for 24 areas. However, when limiting the areas to 25, the synthesis time suddenly jumps to over 20 minutes, causing a timeout. We explain this behavior by



considering the output of the area synthesis, whereas for all area limitations up to 24, an assignment can be found which utilizes all areas available. This means once the SMT solver found one valid assignment, utilizing the maximum amount of areas, it knows it cannot perform any better and terminates. However once we limit the number of areas to 25, no assignment utilizing all 25 areas can be found, causing the need for the SMT solver to verify that utilizing 24 areas is the best it can do, which takes a lot more time than finding a single optimal assignment.

It is important to note that for small topologies such as tutorial\_3 (Figure 5.20), tutorial\_4 (Figure 5.21) and tutorial\_5 (Figure 5.22) the default number of areas is lower than ten areas, which explains why the overall runtime for the default input is lower.

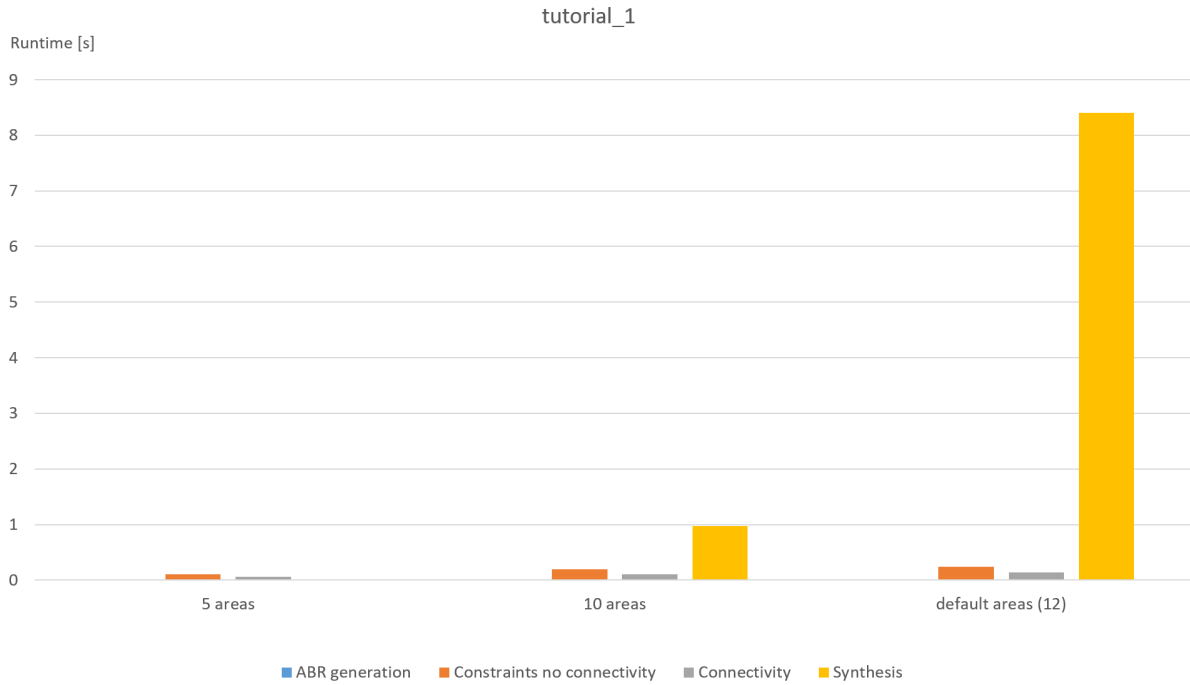


Figure 5.18: Runtime of tutorial\_1 with different area limitations

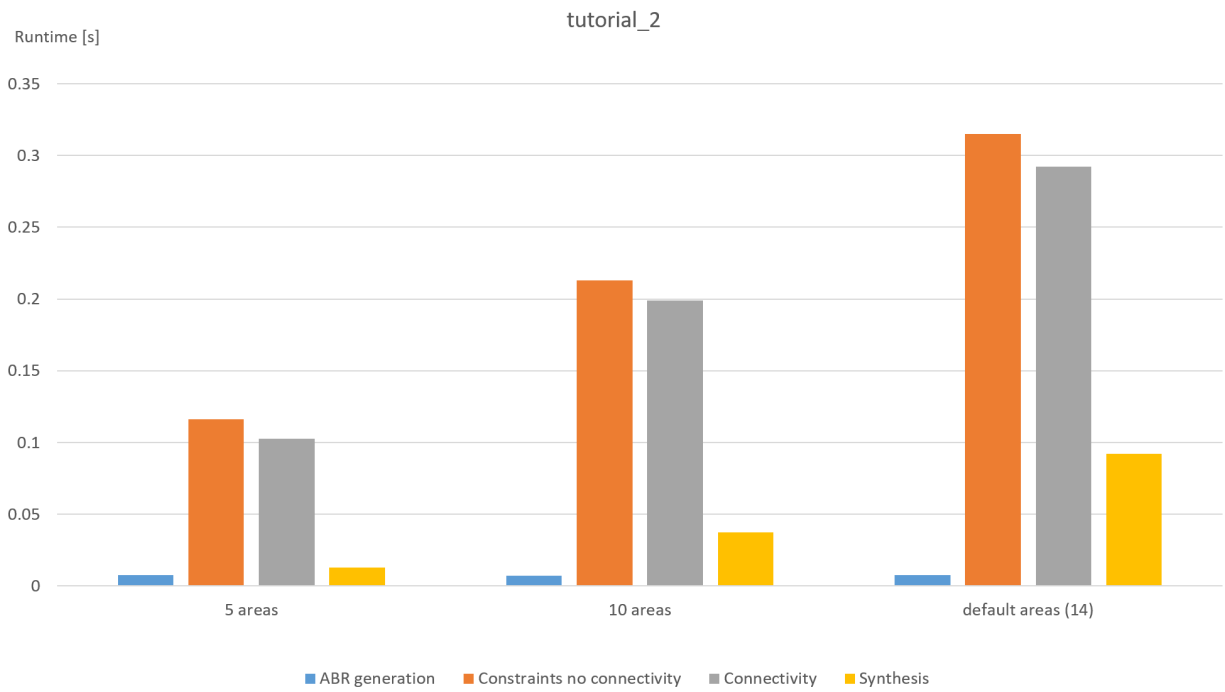


Figure 5.19: Runtime of tutorial\_2 with different area limitations

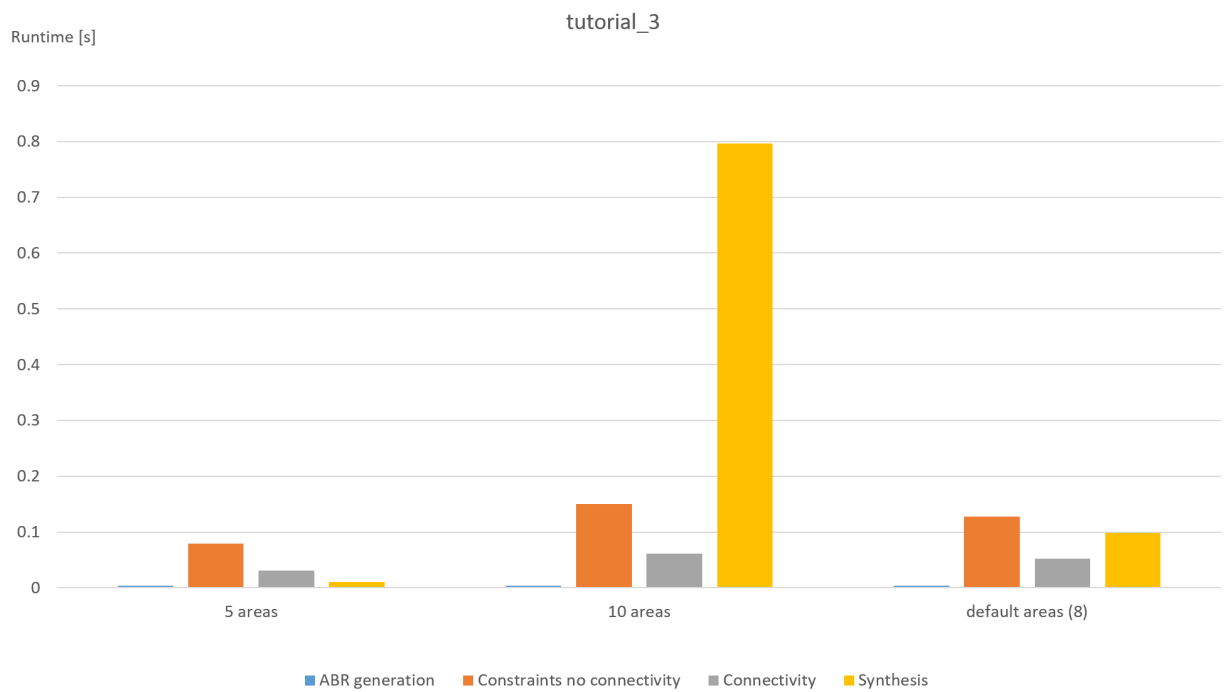


Figure 5.20: Runtime of tutorial\_3 with different area limitations

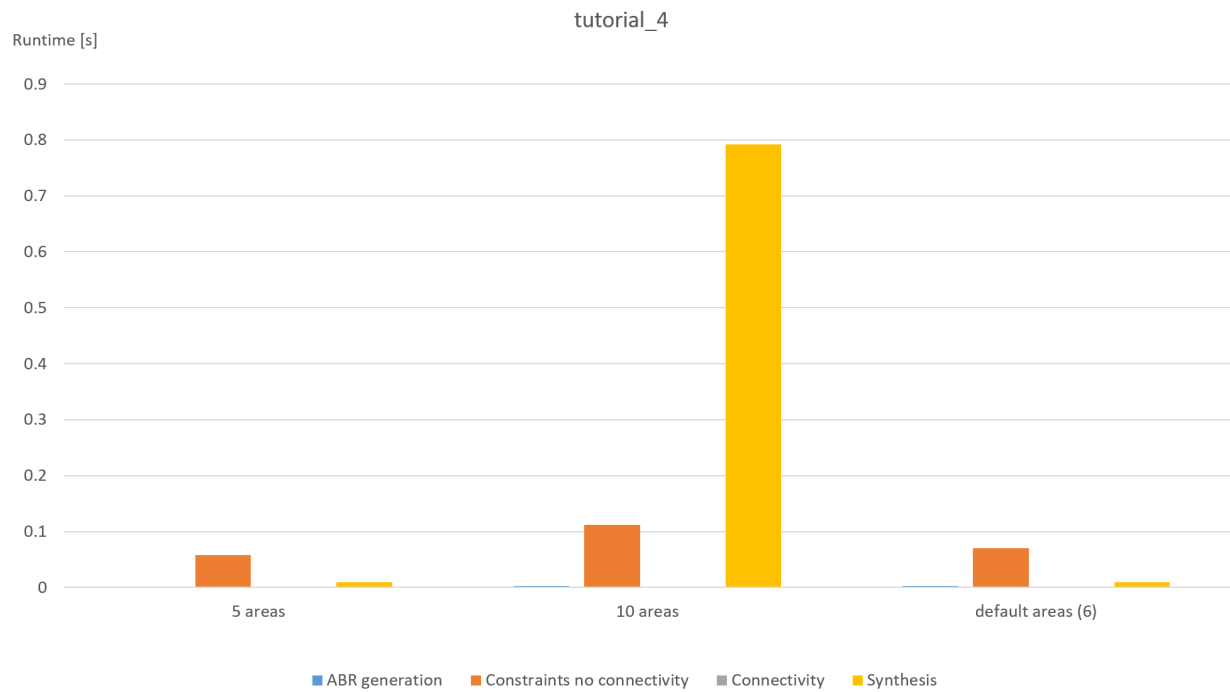


Figure 5.21: Runtime of tutorial\_4 with different area limitations

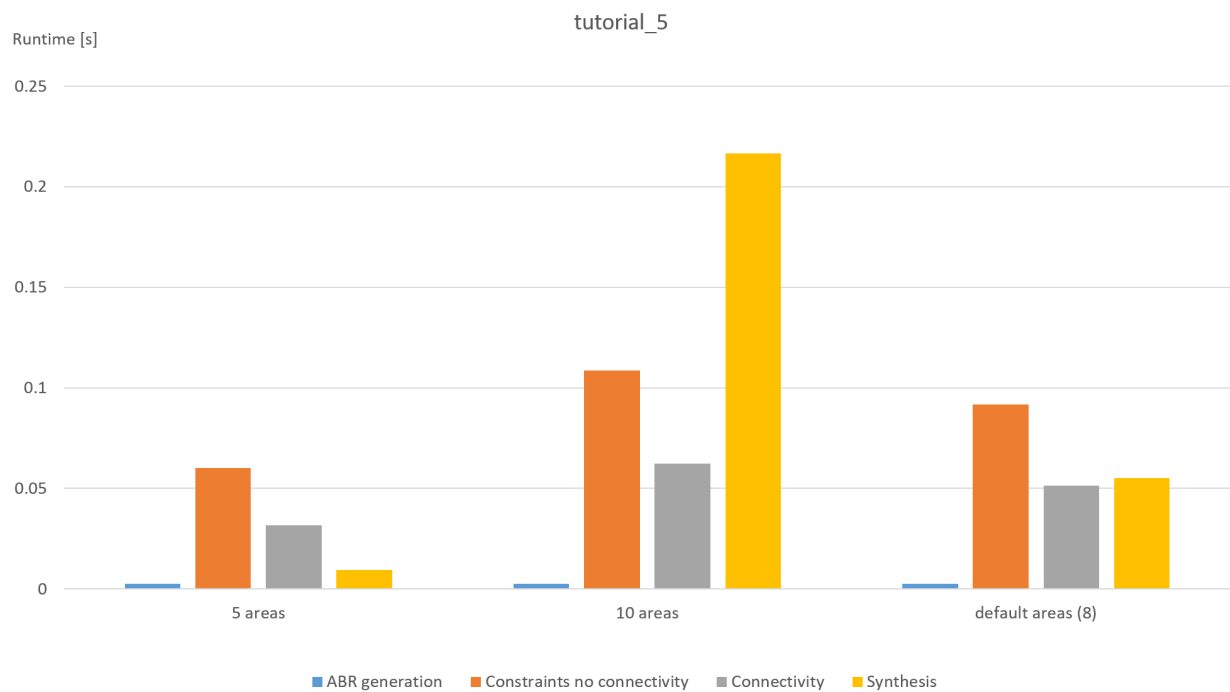


Figure 5.22: Runtime of tutorial\_5 with different area limitations

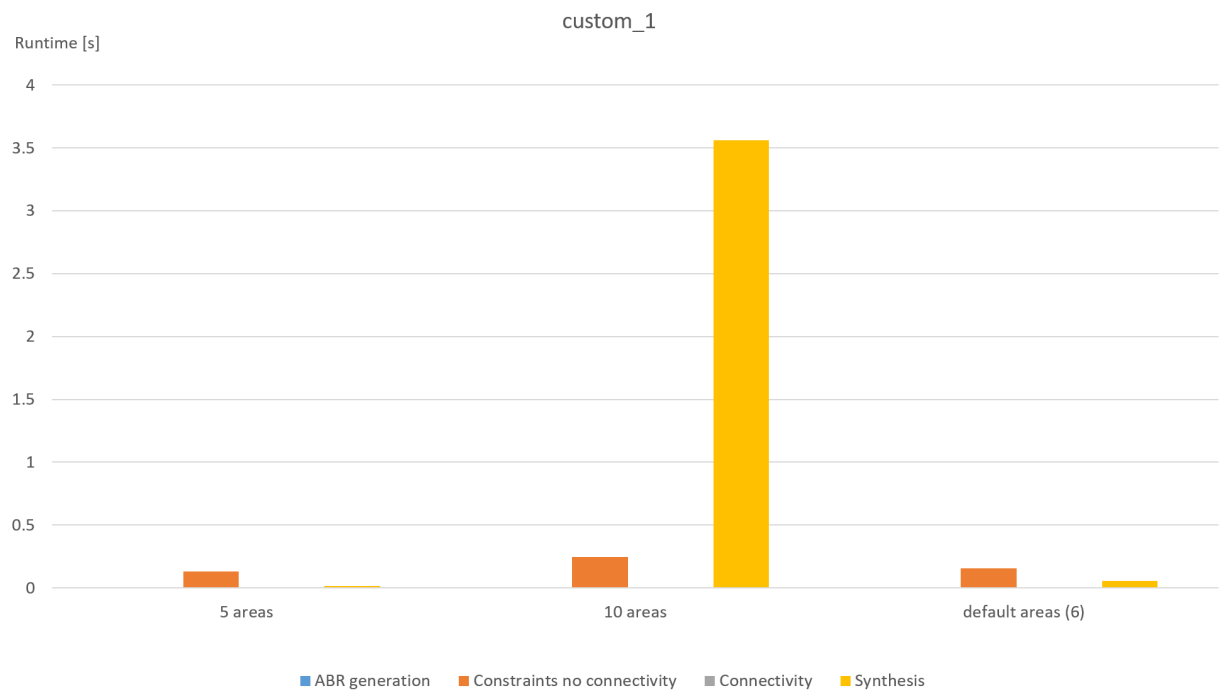


Figure 5.23: Runtime of custom\_1 with different area limitations

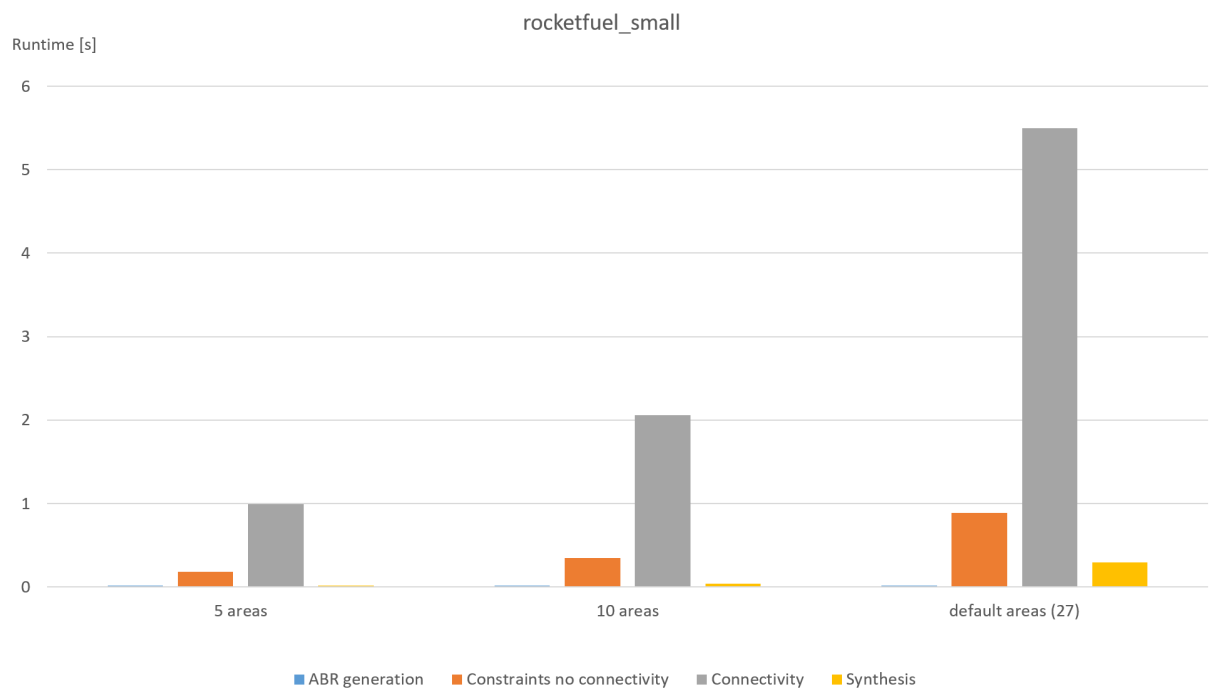


Figure 5.24: Runtime of rocketfuel\_small with different area limitations

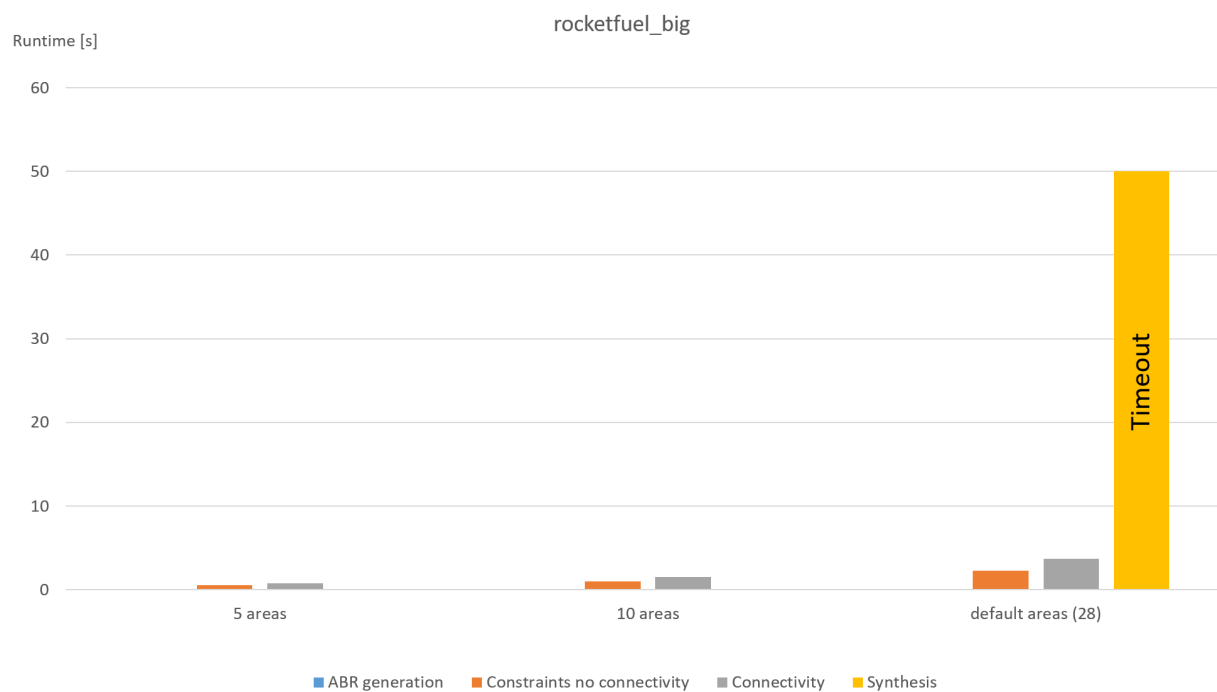


Figure 5.25: Runtime of rocketfuel\_big with different area limitations

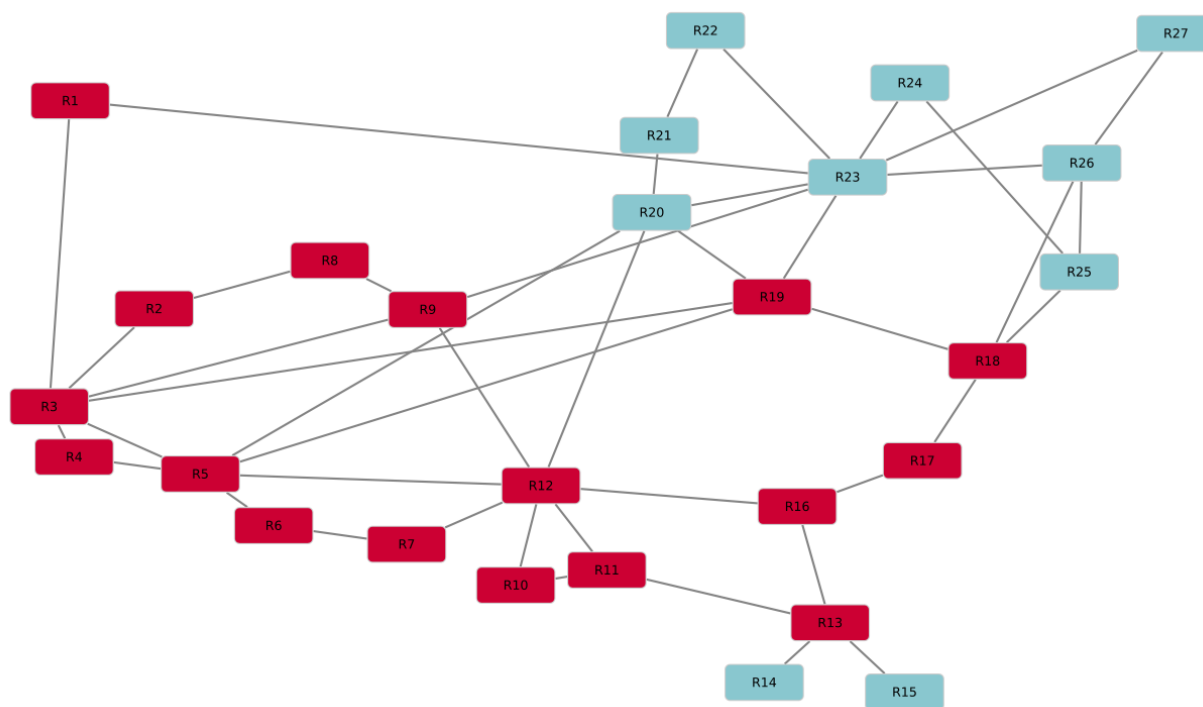


Figure 5.26: ABR (red) input for rocketfuel\_small

### 5.5.2 Topology Size

In this section we want to evaluate the impact of the topology size on the runtime to see if our design scales to large topology sizes.

#### Setup

We will use three different topologies sizes to measure the impact, `rocketfuel_1221_r0_cch` with 318 routers, `rocketfuel_1221_cch` with 3515 routers and `rocketfuel_1239_r1_cch` with 7303 routers. We take all three topologies from the `rocketfuel` project [10]. They represent topologies that are used in the real world. As we have seen in the previous section 5.5.1, the number of areas has a great impact on the performance, as we need to generate almost all constraints for every area individually. Therefore we decided to repeat the measurement in this section for three different configurations, once limited to 10 areas (Figure 5.27), once limited to 20 areas (Figure 5.28) and once limited by the default number of areas (Figure 5.29). We limit the runtime to a maximum of 12 hours total.

#### Results

Even for large topologies, the ABR generation only takes up a fraction of the total runtime and is no limiting factor. As the topology grows in size, the amount of neighboring relationships increases significantly thus increasing the runtime for the constraint generation excluding the area connectivity substantially. Surprisingly the runtime for the area connectivity constraint generation stays fairly low, even for large topologies. In large topologies there exist countless paths connecting two routers, which would lead to a drastic increase of the runtime, making it near impossible to run the synthesis. However, since we only consider the shortest paths in our connectivity constraint generation, the number of considered paths and thus the runtime both stay at a manageable level. As expected when working with the default number of areas, the measurement times out after a total runtime of 12 hours. While the smaller topology `rocketfuel_1221_r0_cch` manages to reach the synthesis stage, the bigger topologies did not even manage to generate the first half of their constraints for the general area limitations.

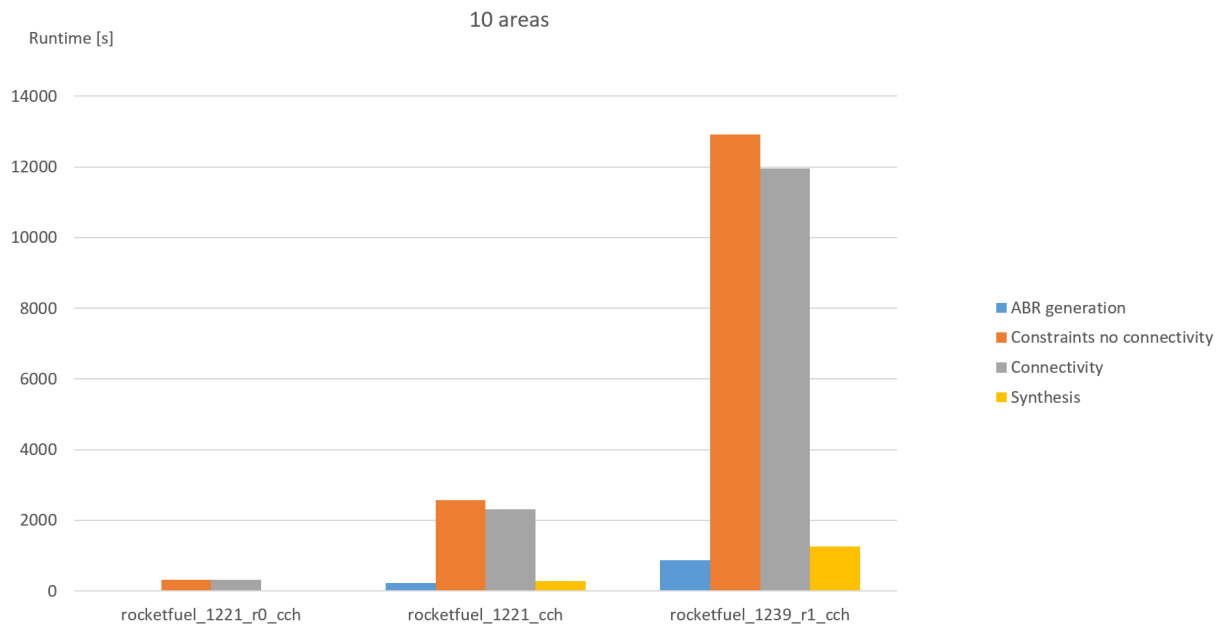


Figure 5.27: Runtime for big topologies limited by 10 areas

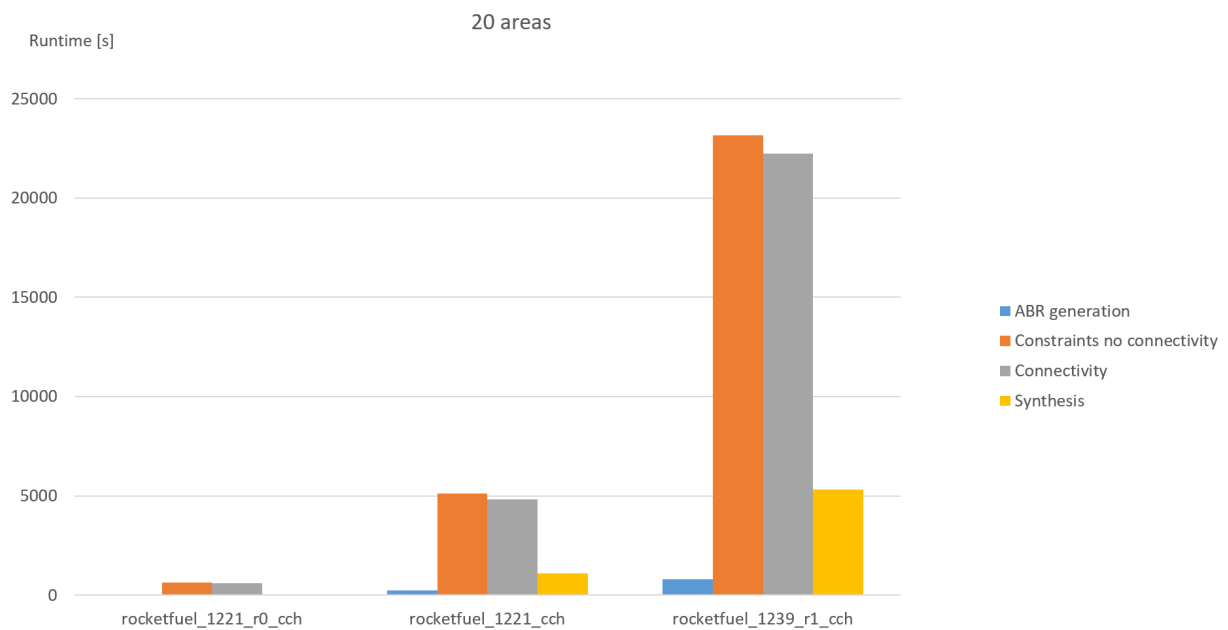


Figure 5.28: Runtime for big topologies limited by 20 areas

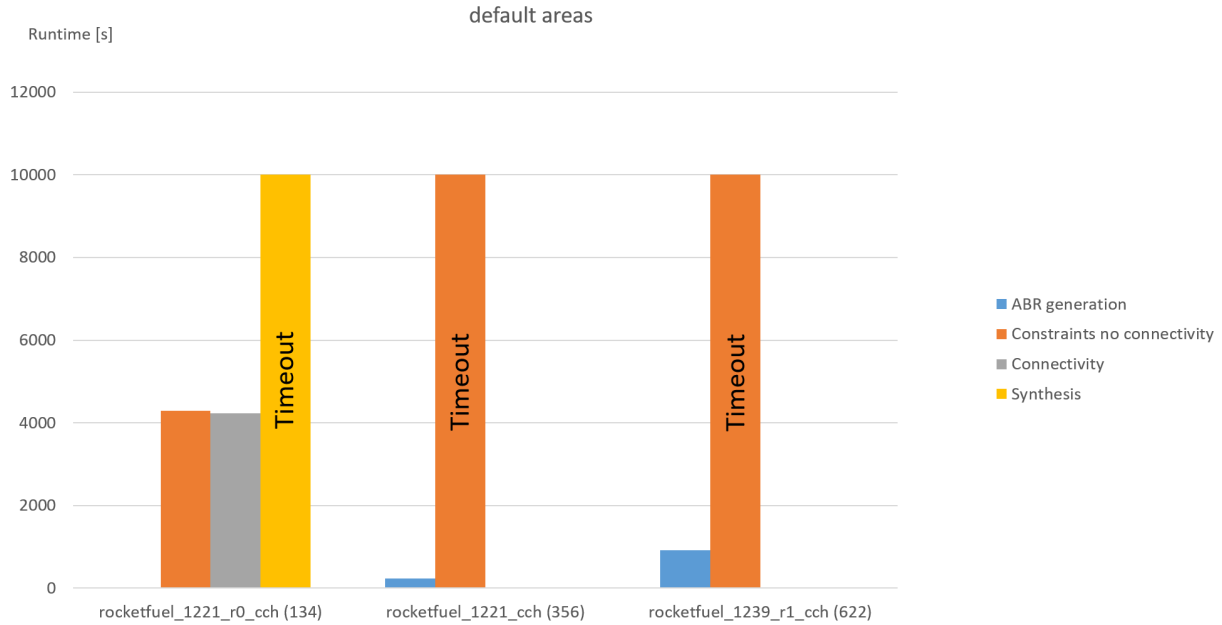


Figure 5.29: Runtime for big topologies limited by the default amount of areas given in brackets

### 5.5.3 Threshold For ABR Generation

In this section we evaluate the impact on the runtime when adjusting the threshold for the ABR generation. Different thresholds result in a different ABR selection which implies different constraints.

#### Setup

We limit the amount of areas available to the synthesis to ten to avoid timeouts even for larger topologies.

#### Results

Two fairly clear trends stick out when looking at the measurements for all topologies, shown in Figure 5.30 to Figure 5.37.

The first trend is the increase of the area connectivity constraint generation runtime for an increasing threshold. This comes from the fact that a higher threshold means more ABRs, whereas every ABR needs to generate its own set of constraints to guarantee the area connectivity.

The second trend is a generally low synthesis runtime for very low and very high thresholds, as we either have not enough constraints to separate areas or too many constraints which limit the search space too much. Exceptions to this second trend is on the one hand `rocketfuel_small`, which, as already discussed, has a low synthesis time since for even a low threshold we select a lot of ABRs, limiting the SMT solver greatly and counteracting the increase in runtime for higher thresholds. On the other hand `tutorial_2` and `rocketfuel_big` do not follow the rule either, which we explain by looking at the output of the area synthesis. For `rocketfuel_big` an input threshold of 0.1 produces an assignment of five areas, 0.2 produces seven areas and 0.3 produces nine areas. Threshold 0.4 or



larger produces an assignment of ten areas which is optimal and therefore the SMT solver does not have to prove that there is no better solution, significantly decreasing the synthesis time. For topology tutorial\_2 the behavior is more difficult to explain since none of thresholds actually produces ten areas. Low thresholds produce three or four areas, while the threshold of 0.9 produces five areas. The increase in runtime is most likely due to the fact that the larger amount of constraints (we can see this by the runtime of the connectivity constraint generation) makes it harder for the SMT solver to prove that five areas is the optimal solution, while not reducing the search space enough to counteract this increase in runtime.

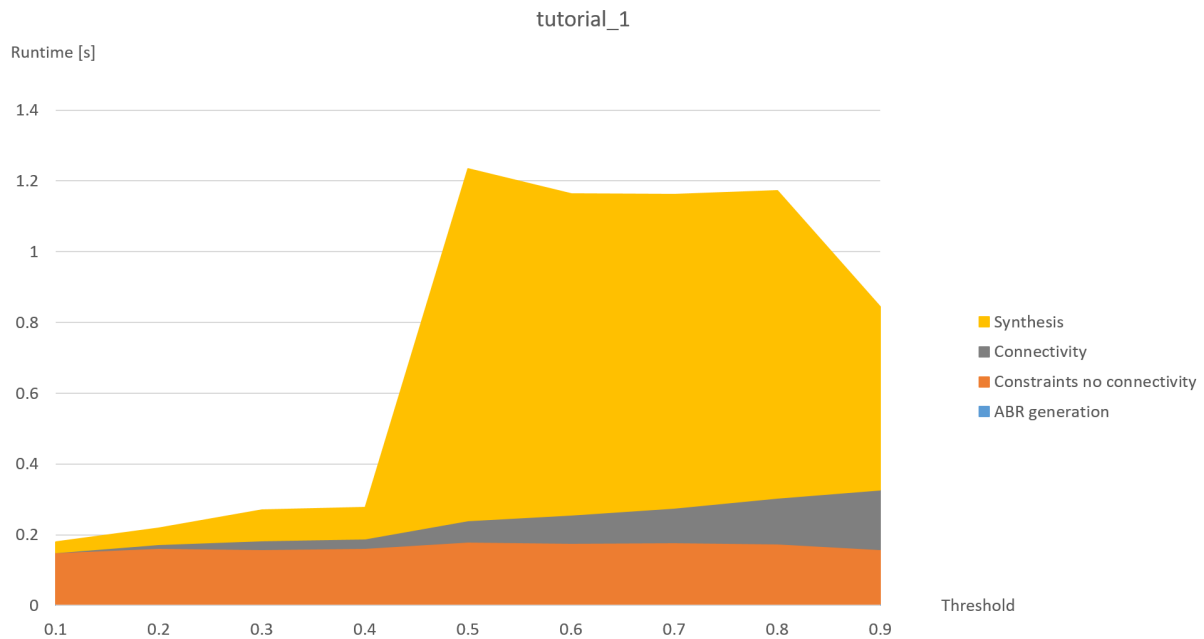


Figure 5.30: Runtime of tutorial\_1 with different thresholds for ABR generation

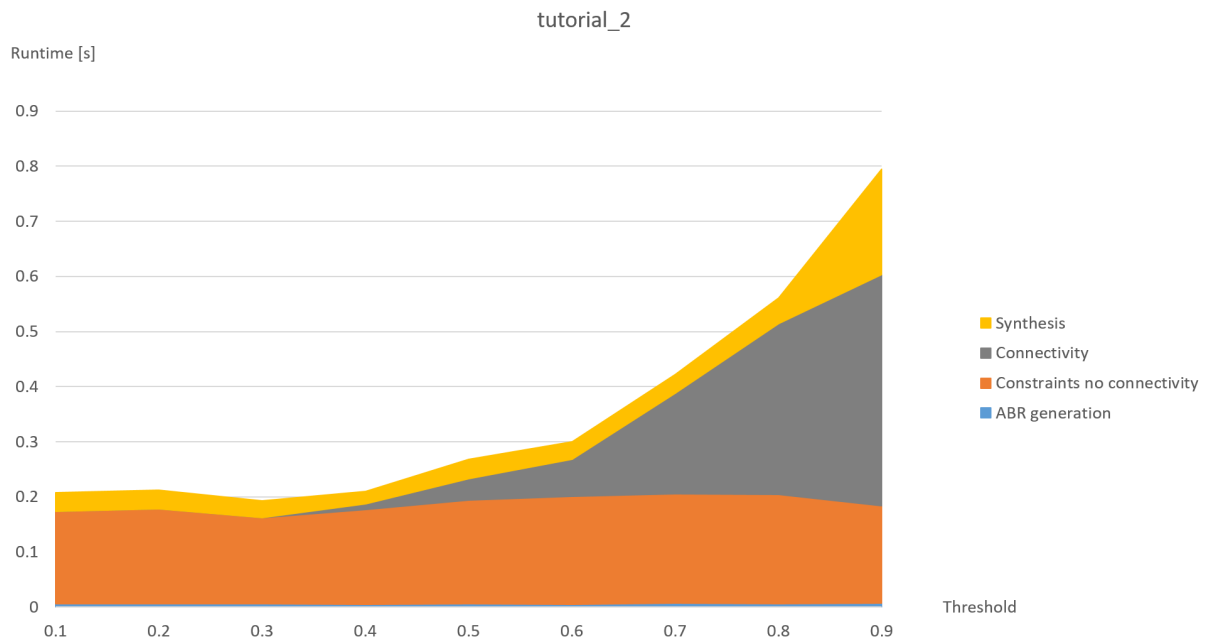


Figure 5.31: Runtime of tutorial\_2 with different thresholds for ABR generation

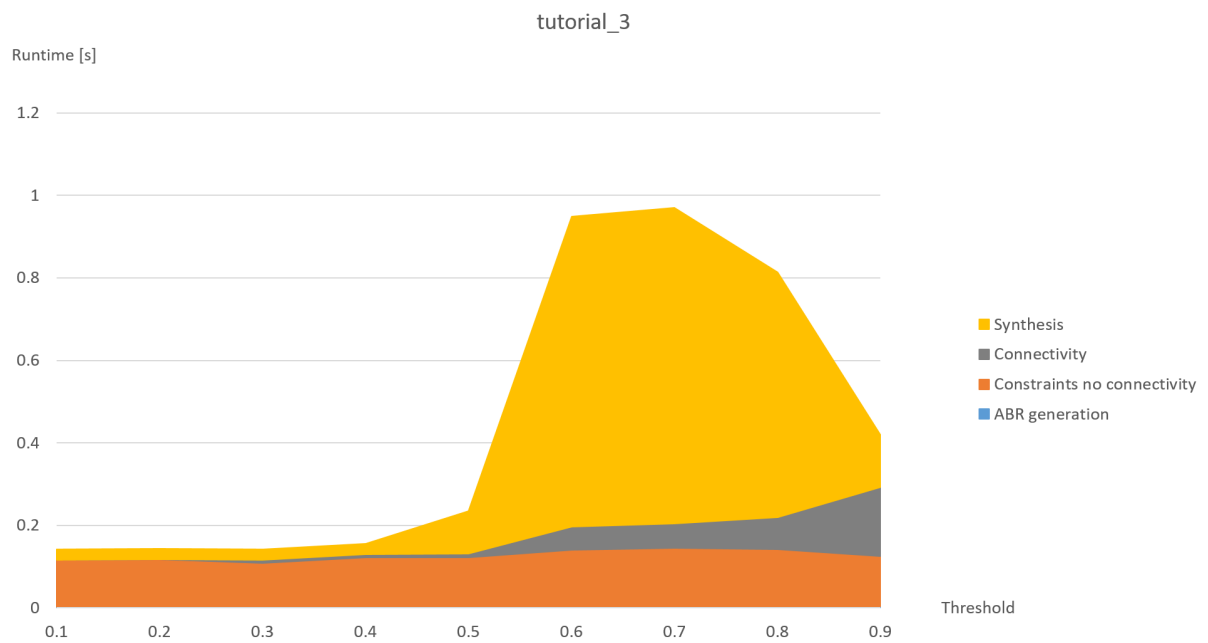


Figure 5.32: Runtime of tutorial\_3 with different thresholds for ABR generation

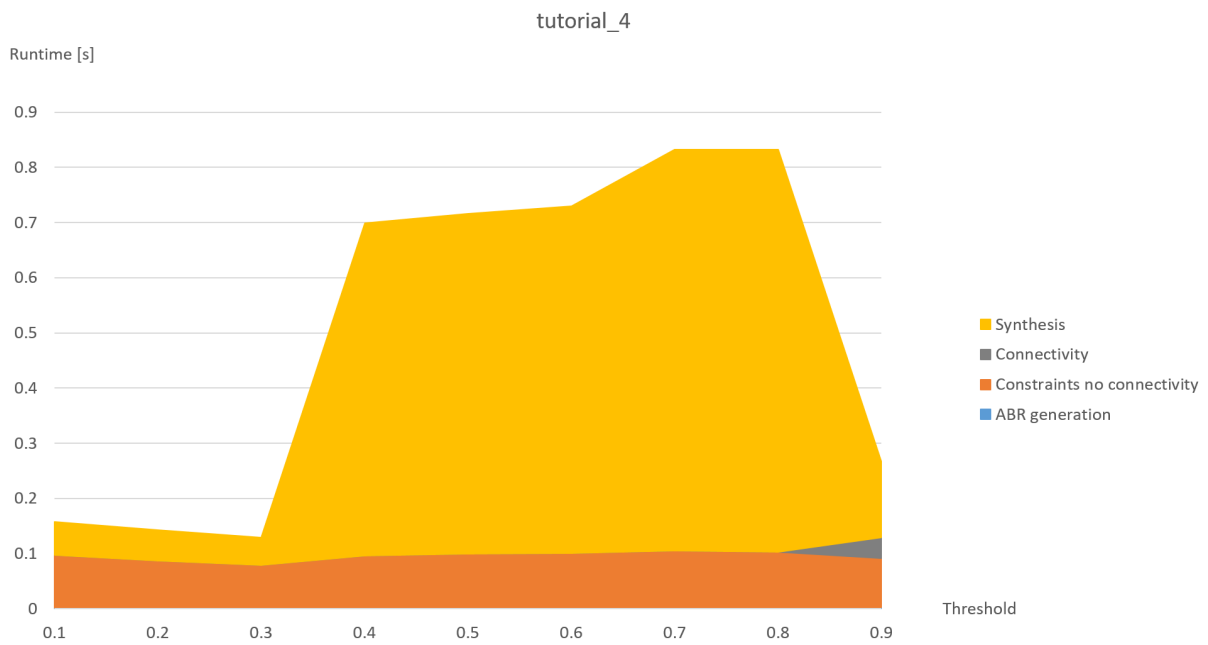


Figure 5.33: Runtime of tutorial\_4 with different thresholds for ABR generation

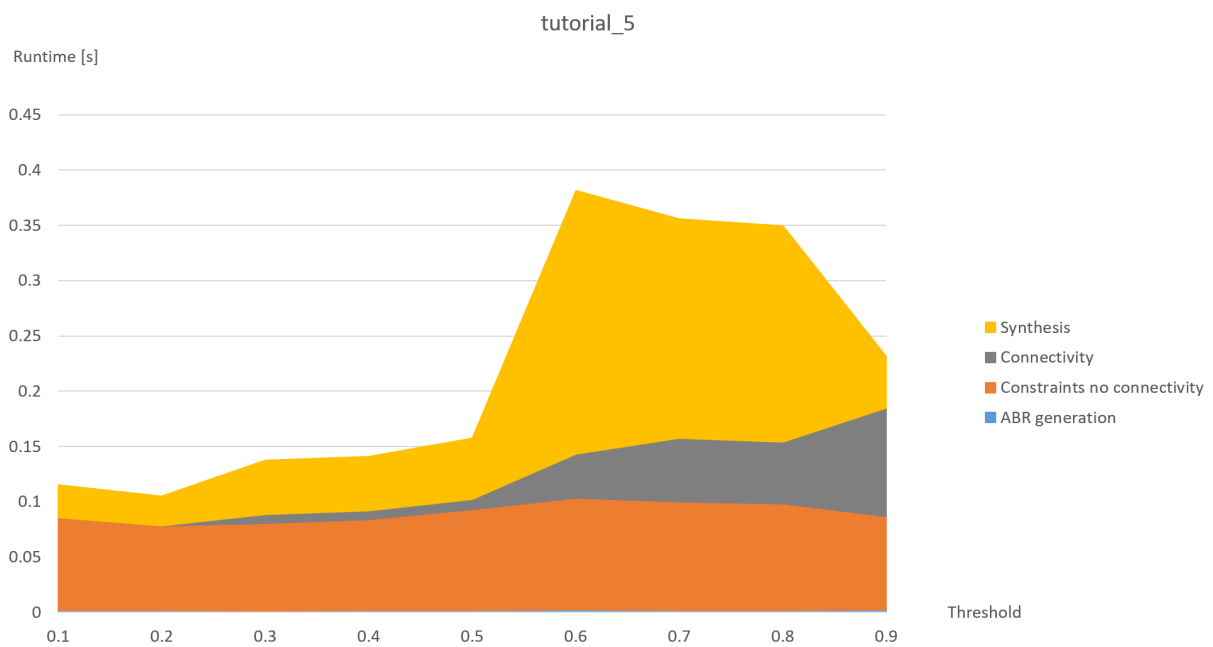


Figure 5.34: Runtime of tutorial\_5 with different thresholds for ABR generation

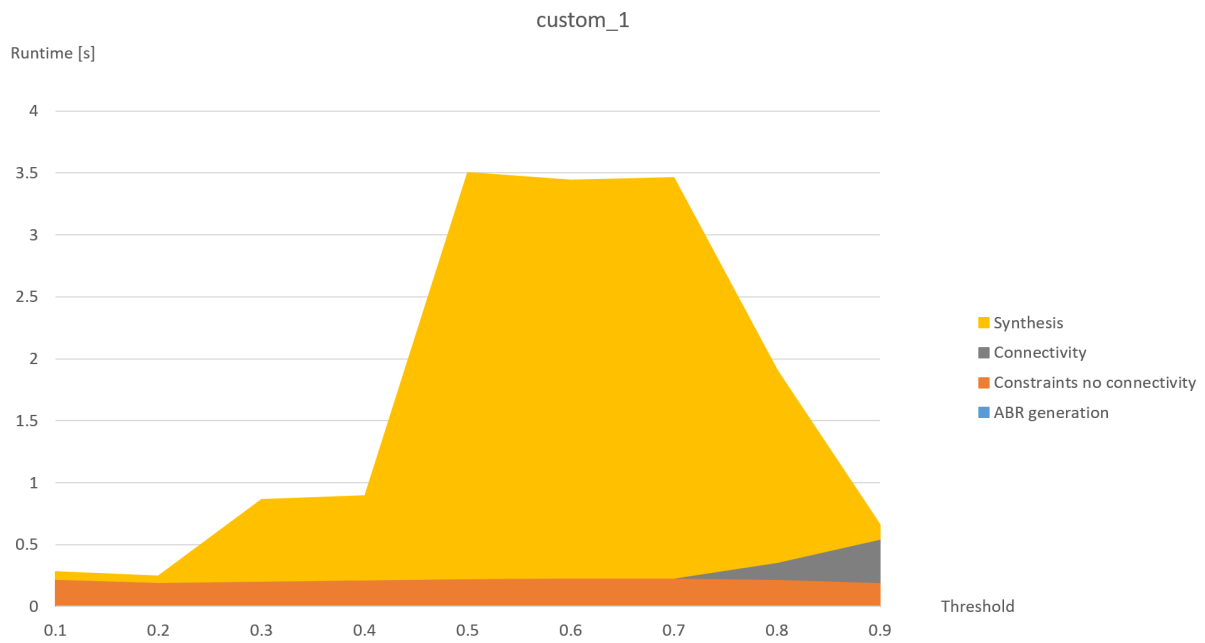


Figure 5.35: Runtime of custom\_1 with different thresholds for ABR generation

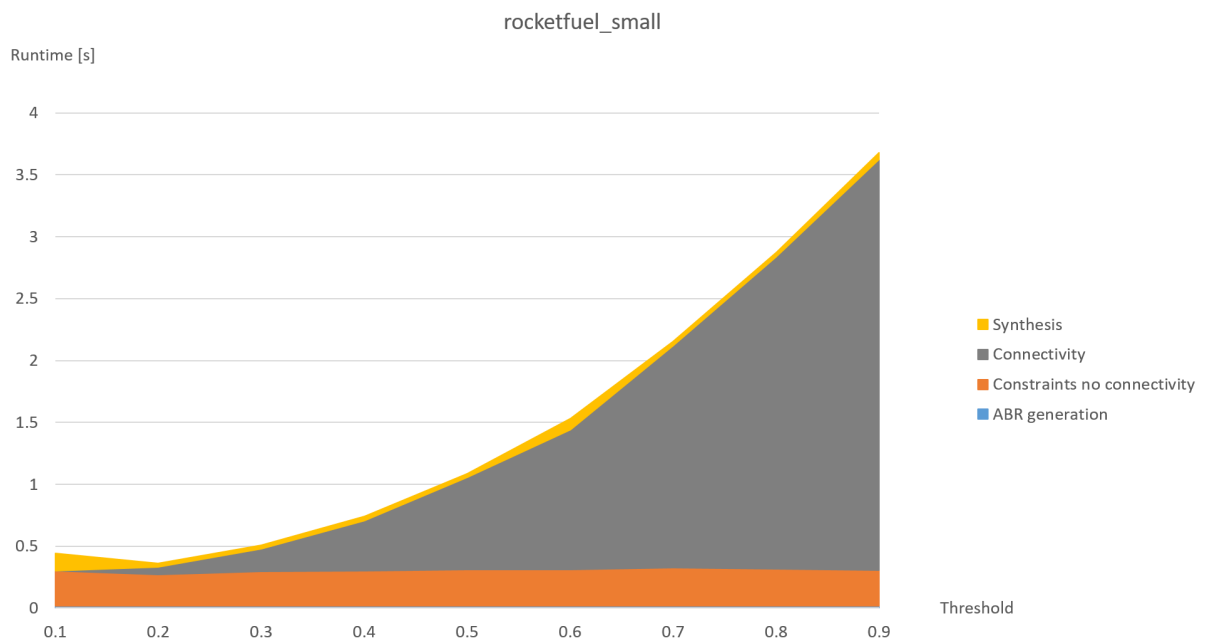


Figure 5.36: Runtime of rocketfuel\_small with different thresholds for ABR generation

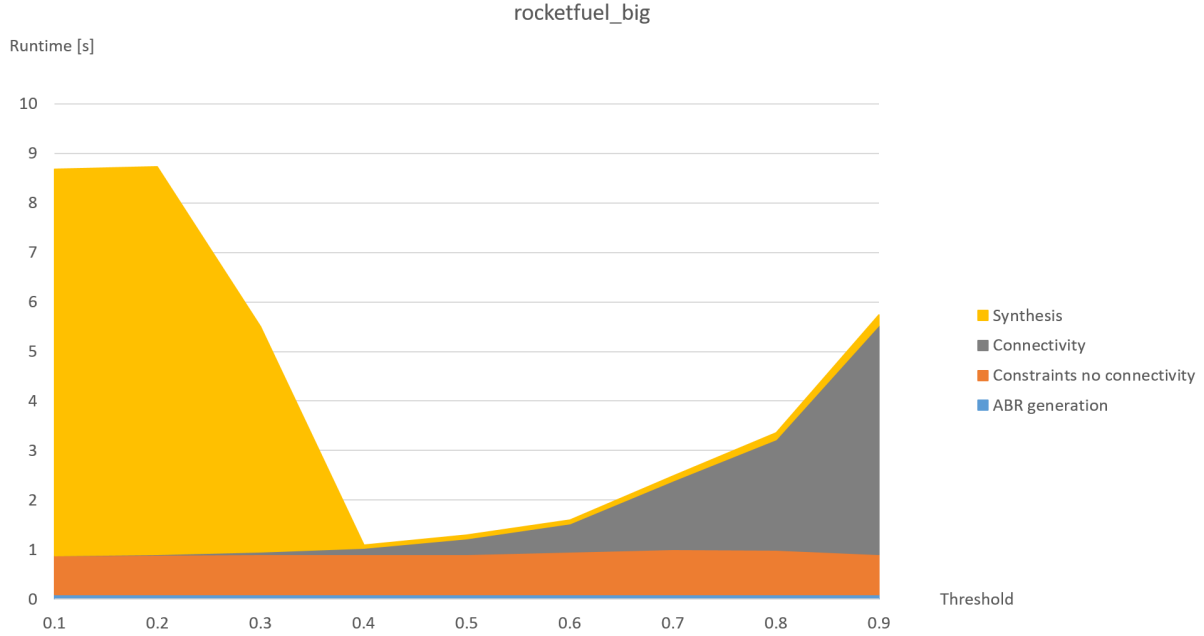


Figure 5.37: Runtime of `rocketfuel_big` with different thresholds for ABR generation

#### 5.5.4 Topology Shape

In this section we will take a look at five different topology types and evaluate what impact they have on the runtime. The difference between those topologies is the number of neighbors and the number of paths between two routers.

##### Setup

The topologies we will look at are a line topology, ring topology, full mesh topology, a full mesh topology that has 50% of its edges removed and a full mesh topology that has 80% of its edges removed (Figure 5.39). The line graph has a minimum amount of neighboring relationships for a connected topology and only offers one single path between any two routers. The ring graph adds one neighboring relationship and increases the number of paths to two. The different versions of the full mesh have way more neighboring relationships and paths, reaching the maximum for the full mesh with no edges removed. All of them have three fixed ABRs, *R5*, *R15* and *R25*. The ABR generation is only measured and not used as input.

##### Results

Figure 5.38 shows the results of the measurement and the trend that better connected topologies generally need more time to generate their constraints. Heavily connected topologies will push the runtime of the constraint generation not including the area connectivity as there are a lot of neighboring relationships and for each we need to generate constraints. We can see this best when looking at the three versions of the full mesh, whereas removing some of the edges lowers the number of neighboring relationships.

The runtime of the area connectivity constraints in a full mesh is especially low since all ABRs are

directly connected to each other, which means that we do not need to generate a constraint for multiple paths.

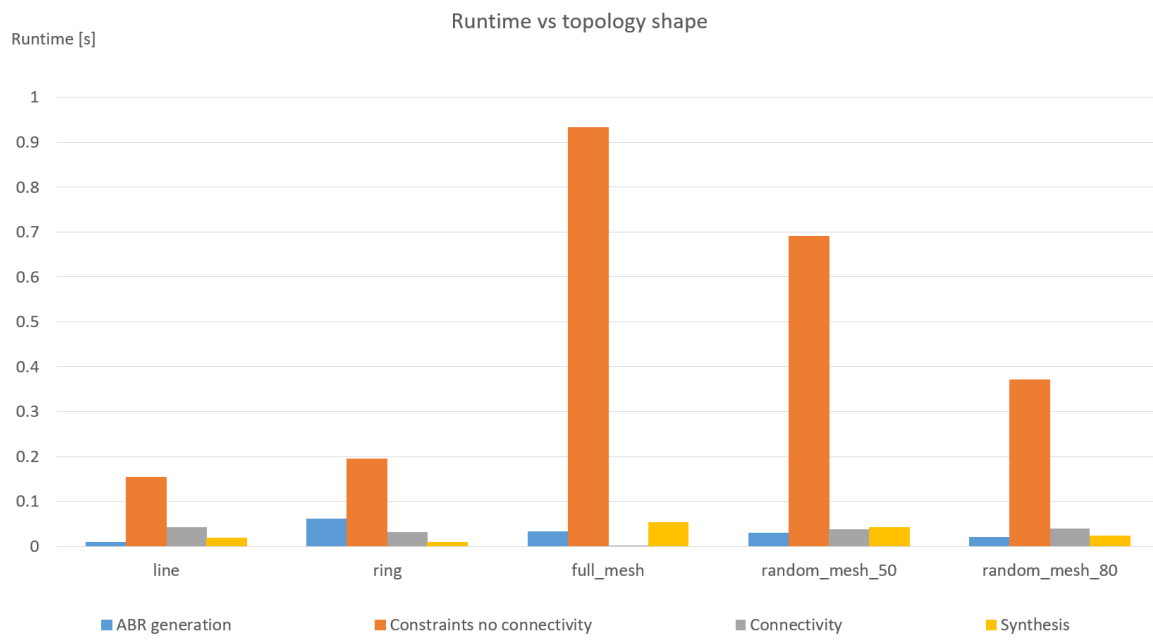


Figure 5.38: Runtime comparison of topology types

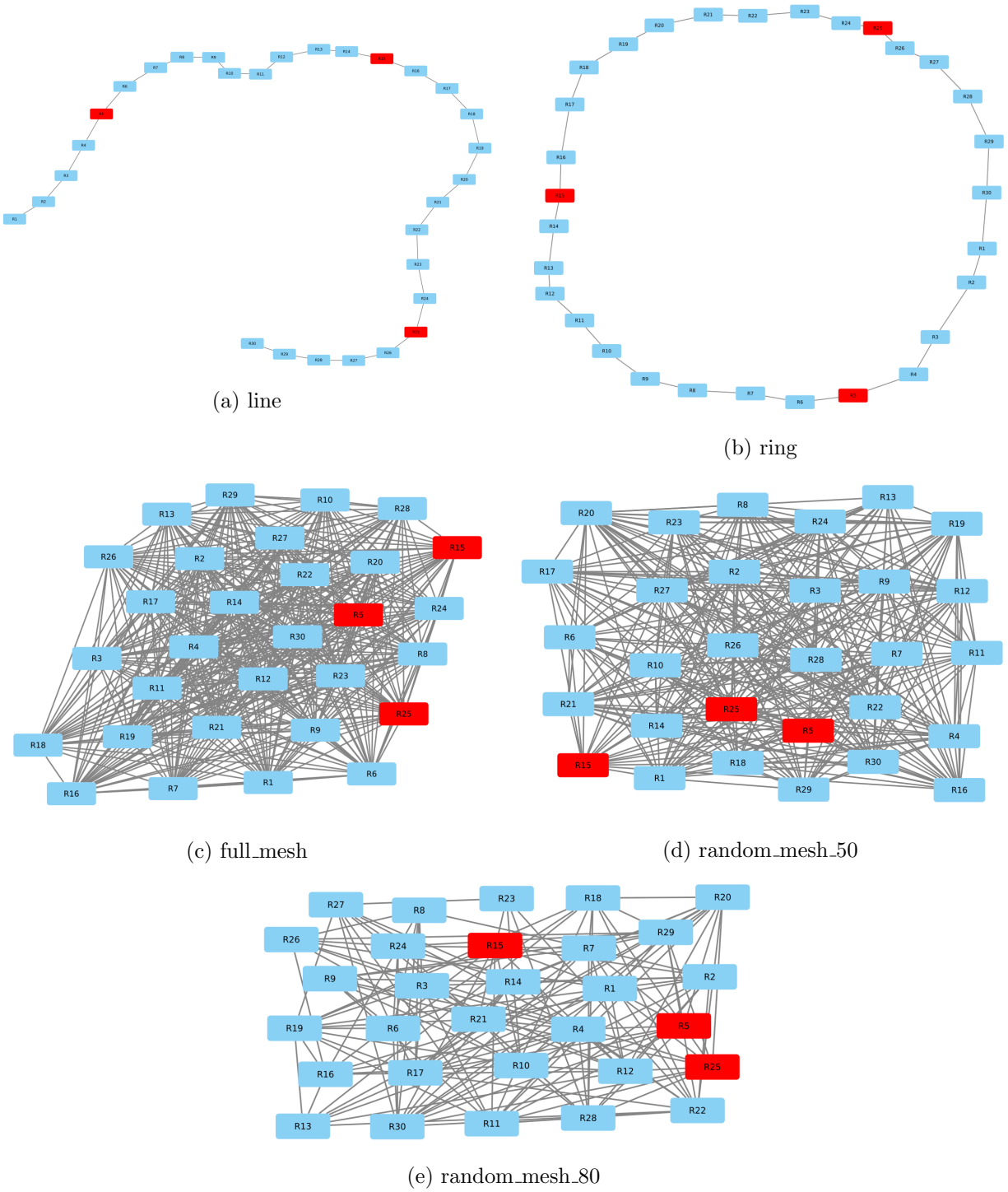


Figure 5.39: Five different topology shapes

### 5.5.5 Manual Area Input

Last but not least we want to evaluate the impact of providing some area input manually. Assigning some routers to an area before the synthesis starts can potentially decrease the search space and therefore lower the runtime.

#### Setup

To do this we consider topology `rocketfuel_big` with the ground truth ABR input as shown in Figure 5.40. We will evaluate the base case with no input provided, four different versions, each with a different marked area (green, dark blue, purple and yellow) as input, and one final version where all marked areas combined are given as input. We only provide one single router as input for the whole area since the neighboring relationships take care of the rest. To avoid the possibility of producing an unsatisfiable problem, we assign the marked areas to the backbone.

#### Results

Figure 5.40 shows that the runtime for the ABR generation and both constraint generations is close to constant for all inputs, which is expected as all constraints are generated regardless of the manual area input. If we take a closer look at the synthesis in Figure 5.42, we can see that it shows no great differences either. Only a slight decrease shows when all areas are provided as input.

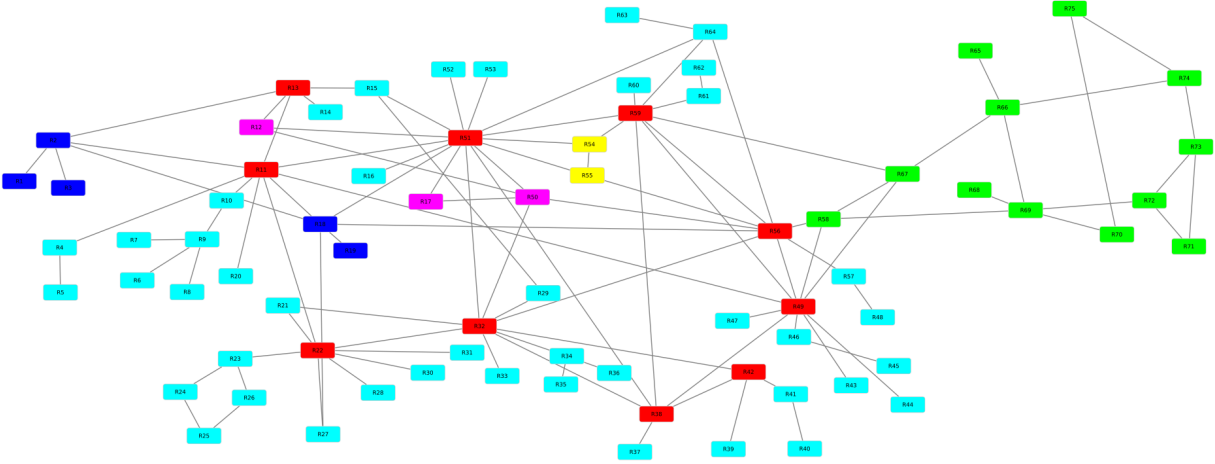


Figure 5.40: topology `rocketfuel_big`, light blue routers have no areas assigned



## 5.6 Discussion

The ABR generation provides a fairly fast way of generating a set of ABRs that enables the assignment of a large amount of areas. However it prefers to split of areas with a single ABR. Areas that can be split off with multiple ABRs are not recognized as such. This can lead to a bad generation of ABRs for topologies such as `rocketfuel_small`. Since nearly every single router in this topology has at least two neighbors, we can not split off any areas with the choice of a single ABR. Generally we recommend the use of a large threshold to give the synthesis a larger search space, which increases the chance for an assignment of more areas. Since the synthesis also eliminates redundant ABRs we get a relatively low amount of ABRs even for large thresholds.

The area synthesis produces close to expected results for a given topology and ABR input, which makes it a great tool to generate the area assignment and ensuring its correctness compared to writing it down by hand. When utilizing the area synthesis together with the ABR generation, the user has to consider the aforementioned points that limit the ABR generation.

In terms of performance, the most important factor is by far the number of available areas. For small topologies in the order of tens of routers the number of generated constraints stays low even for a large number of areas. This makes it possible to run the synthesis even when using the default number of areas. For larger topologies however we do not recommend to run the synthesis without restricting the number of areas. The number of constraints generated gets too big and not only increases the the generation runtime but also the synthesis time. The SMT formulas simply get too big and the search space is too large to handle.

Keeping the amount of connections between the routers low mainly benefits the runtime of the constraint generation due to less neighboring relationships, but also keeps the synthesis time lower since we generate fewer constraints.

Manually inputting areas does not yield a great benefit towards the runtime, unless the user assigns a large part of the topology. This decreases the search space for solutions which decreases the synthesis time.

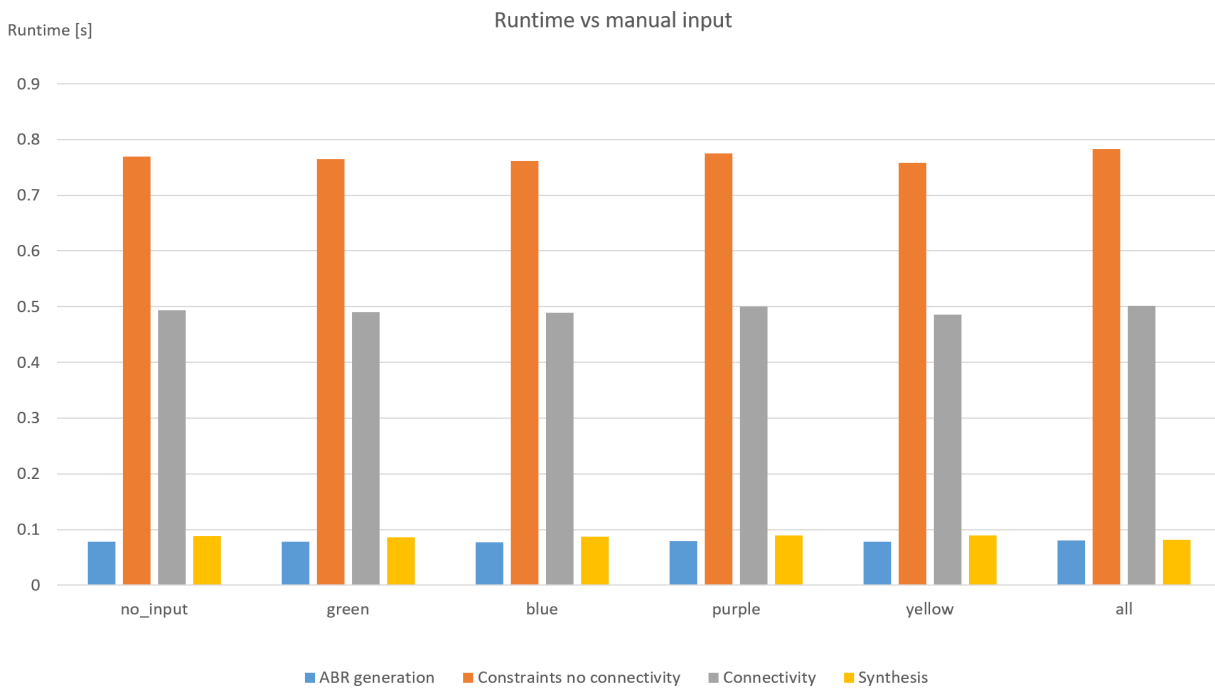


Figure 5.41: Runtime for different manual area inputs

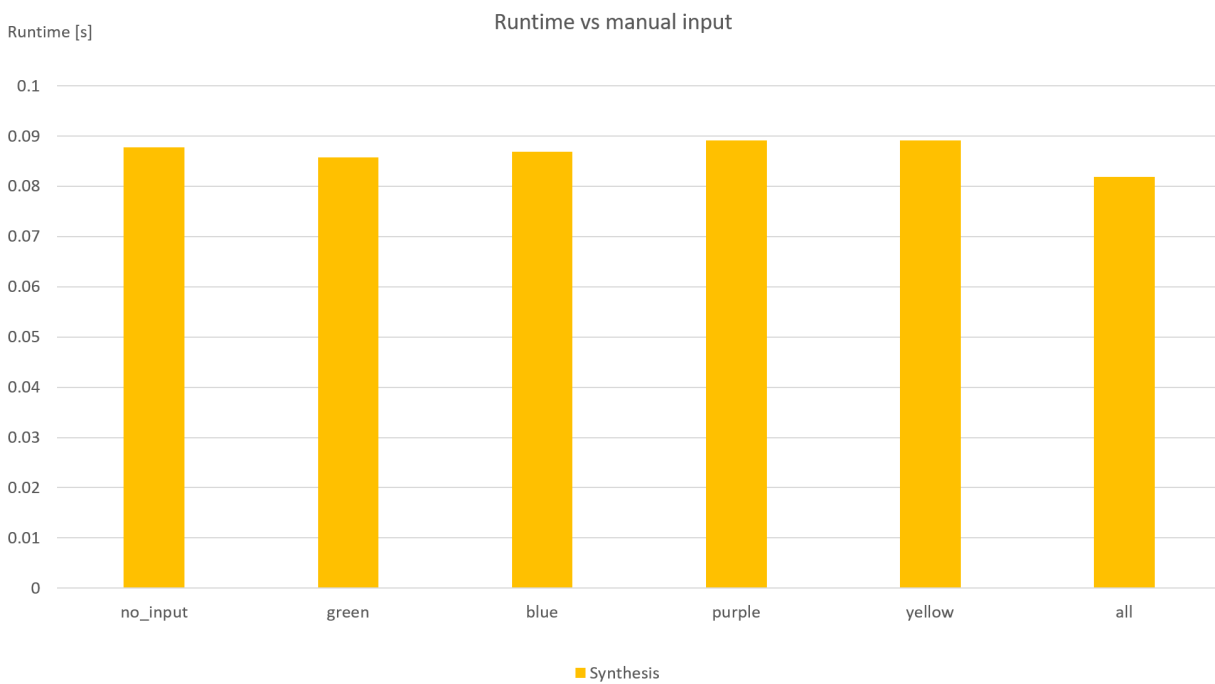


Figure 5.42: Synthesis runtime fir different manual area inputs

## Chapter 6

# Conclusion And Outlook

This chapter summarizes our contributions (Section 6.1) and gives an outlook on improvements that could be done in the future and what problems can be tackled next (Section 6.2).

### 6.1 Conclusion

In the first part of the thesis we added support for two new IGP protocols, RIP and IS-IS, which gives NetComplete a larger set of supported protocols and makes it more interesting for a wider range of users. Based on the addition of those two IGP protocols we realized that they share a lot of functionality which lead to the implementation of the common core. On one hand this allows us to reuse a lot of the code. On the other hand it provides a quick way to implement similar IGPs in a quick fashion, by extending the functionality of only a few functions.

The second part of the thesis focused on the support of hierarchical OSPF. Utilizing this feature can have a big impact on the size of routing tables, the flooding in the network and the CPU usage. Furthermore we enable the network operator to configure stubby areas by providing information about which routers should be in a stubby area. The addition of an area synthesis tool to generate area assignments gives the network operator the possibility to utilize the advantages of hierarchical OSPF with no additional input required. Our evaluation shows that the area synthesis is able to produce valid area assignments that utilize multiple areas in a reasonable time. Furthermore we also show that the design can scale to bigger topologies of multiple hundred routers if we limit the number of areas properly.

### 6.2 Outlook

We see potential to continue and expand upon this work in four categories.

**IGP** We extended the support for IGPs by RIP and IS-IS. Future work can extend the number of supported protocols even further with the addition of protocols like Enhanced Interior Gateway Routing Protocol (EIGRP), Multiprotocol Label Switching (MPLS) or Virtual Local Area Network (VLAN). Some of the widely used protocols today are not yet supported in NetComplete and their addition would greatly increase the utility of it.

**Hierarchical OSPF features** We implemented the support for hierarchical OSPF which offers a multitude of new features, of which we only explored the area stubbiness. A key feature we have not looked at in this thesis is route summarization. Besides stubby areas, route summarization offers an additional tool to keep the routing tables and LSAs in a network to a minimum, reducing memory and bandwidth usage even further. Although the usage of virtual links is not recommended, it

is another feature of hierarchical OSPF. The support for virtual links would open new ways of generating area assignments and increase the search space with new constraints. Additionally virtual links can replace the need for the connectivity of the backbone, which would decrease the number of generated constraints in this regard.

**Area Synthesis** Lastly we can improve the area synthesis in several ways. To begin with, we can extend the ABR generation with an algorithm that is able to detect pairs of routers as ABRs. Not only is it recommended to have multiple ABRs per area, the ABR generation would also perform better in cases where we cannot find single routers that split the network. In addition to the ABR generation, we can improve the area assignment to respect the number of routers in non-backbone areas. As of now, the synthesis only ensures that they are not empty. While this is not an issue for small topologies, it can lead to solutions in bigger topologies where only a small fraction of routers is in a non-backbone area while still utilizing a large amount of areas.

**Hierarchical IS-IS** Utilizing the insights we gained for hierarchical OSPF we can expand IS-IS to support a hierarchical structure as well. Additionally the area assignment synthesis can be modified to support IS-IS as well as OSPF.

# Bibliography

- [1] Cisco - stub area configuration of ospf. [https://www.cisco.com/c/m/en\\_us/techdoc/dc/reference/cli/nxos/commands/ospf/area-stub-ospf.html](https://www.cisco.com/c/m/en_us/techdoc/dc/reference/cli/nxos/commands/ospf/area-stub-ospf.html). Accessed: 2020-09-22.
- [2] Cisco recommendations for hierarchical ospf. <https://ccnacompletecouse.blogspot.com/2019/10/ospf-area-and-lsa-types-link-state.html>. Accessed: 2020-09-13.
- [3] Cytoscape. <https://cytoscape.org/>. Accessed: 2020-09-17.
- [4] Designing cisco network service architectures (arch): Developing an optimum design for layer 3 (ccdp). <https://www.ciscopress.com/articles/article.asp?p=1763921&seqNum=6>. Accessed: 2020-09-22.
- [5] Isis tutorial. <https://www.menog.org/presentations/menog-4/MENOG4-ISIS-Tutorial.pdf>. Accessed: 2020-09-13.
- [6] Ospf implementation. <https://www.ciscopress.com/articles/article.asp?p=2294214>. Accessed: 2020-09-22.
- [7] Ospf lsa types explained. <https://networklessons.com/ospf/ospf-lsa-types-explained>. Accessed: 2020-09-22.
- [8] Rip metric offset. [https://www.cisco.com/c/m/en\\_us/techdoc/dc/reference/cli/nxos/commands/rip/ip-rip-metric-offset.html](https://www.cisco.com/c/m/en_us/techdoc/dc/reference/cli/nxos/commands/rip/ip-rip-metric-offset.html). Accessed: 2020-09-22.
- [9] Rip vs ospf: What is the difference? <https://community.fs.com/blog/rip-vs-ospf-what-is-the-difference.html>. Accessed: 2020-09-22.
- [10] Rocketfuel: An isp topology mapping engine. <https://research.cs.washington.edu/networking/rocketfuel/>. Accessed: 2020-09-22.
- [11] Setting best practice parameters for is-is fast convergence. [https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute\\_isis/configuration/15-s/irs-15-s-book/irs-fscbp.pdf](https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_isis/configuration/15-s/irs-15-s-book/irs-fscbp.pdf). Accessed: 2020-09-22.
- [12] BECKETT, R., MAHAJAN, R., MILLSTEIN, T., PADHYE, J., AND WALKER, D. Don't mind the gap: Bridging network-wide objectives and device-level configurations: Brief reflections on abstractions for network programming. *SIGCOMM Comput. Commun. Rev.* 49, 5 (Nov. 2019), 104–106.
- [13] DE MOURA, L., AND BJØRNER, N. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems* (Berlin, Heidelberg, 2008), C. R. Ramakrishnan and J. Rehof, Eds., Springer Berlin Heidelberg, pp. 337–340.

- [14] EL-HASSANY, A., TSANKOV, P., VANBEVER, L., AND VECHEV, M. Netcomplete: Practical network-wide configuration synthesis with autocompletion. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (Renton, WA, Apr. 2018), USENIX Association, pp. 579–594.
- [15] FOGEL, A., FUNG, S., PEDROSA, L., WALRAED-SULLIVAN, M., GOVINDAN, R., MAHAJAN, R., AND MILLSTEIN, T. A general approach to network configuration analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, May 2015), USENIX Association, pp. 469–483.
- [16] PERLMAN, R. A comparison between two routing protocols: Ospf and is-is. *Network, IEEE* 5 (10 1991), 18 – 24.
- [17] SPRING, N., MAHAJAN, R., AND WETHERALL, D. Measuring isp topologies with rocketfuel. *SIGCOMM Comput. Commun. Rev.* 32, 4 (Aug. 2002), 133–145.
- [18] SUBRAMANIAN, K., D’ANTONI, L., AND AKELLA, A. Genesis: Synthesizing forwarding tables in multi-tenant networks. *SIGPLAN Not.* 52, 1 (Jan. 2017), 572–585.
- [19] WEITZ, K., WOOS, D., TORLAK, E., ERNST, M. D., KRISHNAMURTHY, A., AND TATLOCK, Z. Scalable verification of border gateway protocol configurations with an smt solver. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (New York, NY, USA, 2016), OOPSLA 2016, Association for Computing Machinery, p. 765–780.