From a reprogrammable chip to a hard problem

Els Hoekstra
s4556852

Supervisor Radboud University: Wieb Bosma
Supervisor Nedap: Wouter Kuijper

Bachelor thesis Mathematics
Radboud University
The Netherlands
April 26, 2019

Contents

Preface

For my bachelor thesis, I wanted to do an internship at a company, because I wanted to see what it's like to work in one place for an extended period of time. Seeing how I would like to do my master thesis at the Radboud University, I decided that my bachelor thesis was a good opportunity to do said company internship.

Last year, during my board year, I visited many companies and I got a chance to find out which one suited me. Nedap unexpectedly turned out to be the one I was looking for, and I immediately felt at home there, primarily thanks to the good aubiance. I asked Loes van Hove, a mathematics alumnus of the Radboud University, whether it would be possible for me to write my thesis at Nedap, and so it started.

Nedap gave me two contacts within the company, Albert Dercksen and Wouter Kuijper. Wouter helped me come up with some topics and was my supervisor at Nedap during the project. Wieb Bosma was my supervisor at the Radboud University. I want to thank both of them a lot for the help they gave me during this thesis and for always being there when I needed them.

My thesis has two parts. The first three chapters describe a reprogrammable chip. This is the first chip that has an open source layout. The original assignment was:

For this project we are looking to develop an alternative, low-level hardware design language that does not abstract away all the details concerning placement of our design, within the available three dimensional mesh. The challenge will be to still try and keep such a (visual) language as simple, intuitive and compositional as possible.

To this end, it was important to know what a reprogrammable chip is, exactly, and what it looks like, so that there might be a way to run a program of my own in such a way that it reprograms the chip, and still satisfies the assignment.

Eventually, I went down a different road. A problem that piqued my interest crossed my path. I rewrote the problem in a more abstract form to see if we could say anything about it this way. The latter is discussed in the last two chapters.

1 Components of an FPGA

I will explain what an FPGA is in this section. *FPGA* is shorthand for Field Programmable Gate Array and this is a programmable chip. Here I only discuss an ICE40-FPGA, because there is a lot of research available for this chip. The research is on the firestorm documentation [1]. It is the first chip where we know the wiring.

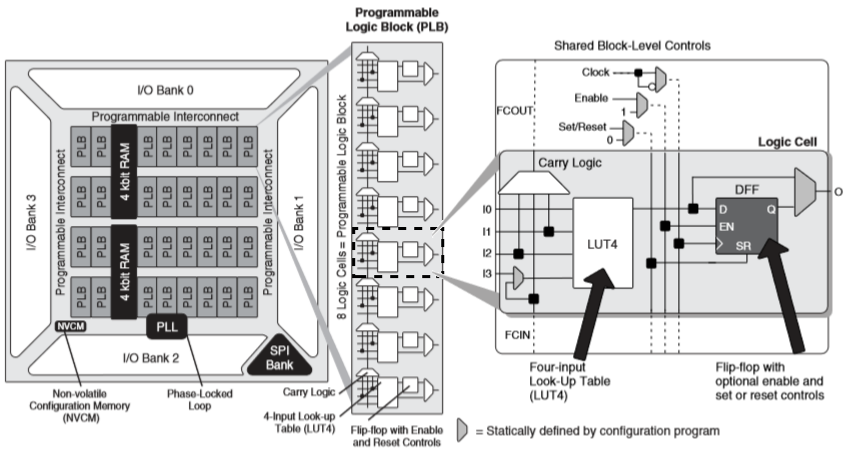

Programmable Logic Block (PLB)
I/O Bank 0
Programmable Interconnect
PLB
4 kbit RAM
I/O Bank 3
I/O Bank 1
I/O Bank 2
NVCM
PLL
SPI Bank
Non-volatile Configuration Memory (NVCM)
Phase-Locked Loop
Carry Logic
4-Input Look-up Table (LUT4)
Flip-flop with Enable and Reset Controls
8 Logic Cells = Programmable Logic Block
Shared Block-Level Controls
Clock
Enable
Set/Reset
FCOUT
Logic Cell
Carry Logic
I0
I1
I2
I3
LUT4
DFF
D
Q
EN
SR
O
FCIN
Four-input Look-Up Table (LUT4)
Flip-flop with optional enable and set or reset controls
= Statically defined by configuration program

Figure 1: FPGA, illustration retrieved from [1]

1.1 Logic Cell

In one *Programmable Logic Block* (PLB)/*Logic Tile* are eight *Logic Cells (LC)* as is shown in Figure 1. The right part is the Logic Cell in detail. A few important components are briefly explained below.

LUT4

LUT4 is short for Four-input Look-Up Table. This is a function we can program ourselves. It is possible to load the output-values in the look-up table. An example is the table below that belongs to the function $(I0 \wedge I1) \vee (I2 \wedge I3)$. The input values I0, I1, I2 and I3 are visible in Figure 1.

I0	I1	I2	I3	output
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

So there are $2^{(2^4)} = 2^{16}$ functions we can make with a LUT4.

DFF

A *DFF* (short for D Flip Flop) can be used as a register. The Flip Flop allows fetching and setting one bit. Only once every clock signal, the DFF can change the bit. In Figure 1 the DFF is shown. The following symbols are used:

- D: Input
- EN: Enable, If Enable = 1, then Q←D, else output is not changed.
- $\triangleright$: Clock-signal
- SR: Set/Reset, this can reset the DFF by changing Q←0 at every moment in time.

Carry Logic

The *Carry Logic* is responsible for more efficiency in the Logic Block. The Carry Logic can transfer one bit to the next Logic Cell through FCOUT. When adding two integers, the Carry Logic can transfer the carry.

From now on we consider one Logic Cell as a black box with input and output values. The internal structure is not relevant for our purposes. We will now zoom out to the level of one Programmable Logic Block/Logic Tile, instead of one Logic Cell. In Figure 2, you can see one tile with all the eight Logic Cells in it with the basic inputs and outputs.

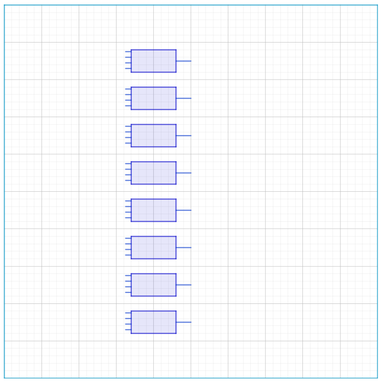


Figure 2: 1 tile, 8 Logic Cells

1.2 Main wires

Now we describe the wiring in the chip. For that we will zoom out even more to see the whole chip. There are three kinds of wiring that span four or more logic tiles.

Global

There are eight wires in the global network. These wires span the whole FPGA. In Figure 3, we see 9 tiles, one wire black highlighted in every tile and the seven other wires, only visible in one tile.

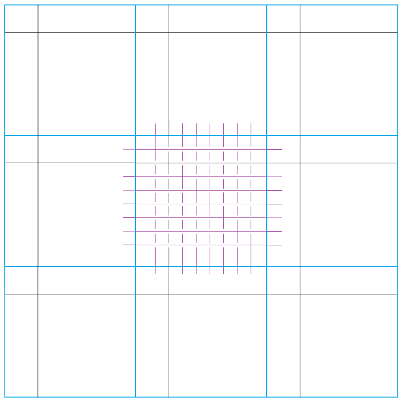
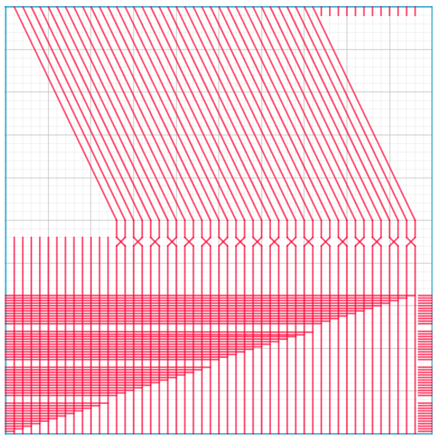


Figure 3: Global network, 9 tiles, 1 black global wire, 7 purple global wires

Span-4

The span-4 wires span four tiles. A horizontal wire that starts at one tile connects tiles with maximal horizontal distance of four. A vertical wire that starts at one tile connects tiles with maximal vertical distance of four. There are $48 + 12$ horizontal wires trough one tile and $48 + 12 + 48$ vertical wires. Below, you see how the wires are connected in one tile. In Chapters two and three, I will explain this in detail.


a. sp4-vertical

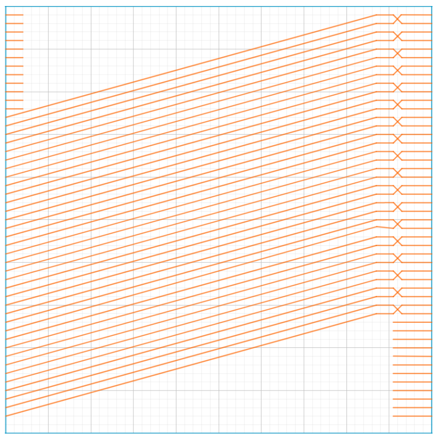
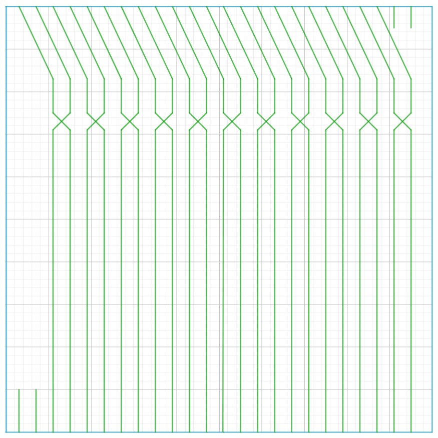

b. sp4-horizontal

Figure 4: Span 4 wires


a. sp12-vertical

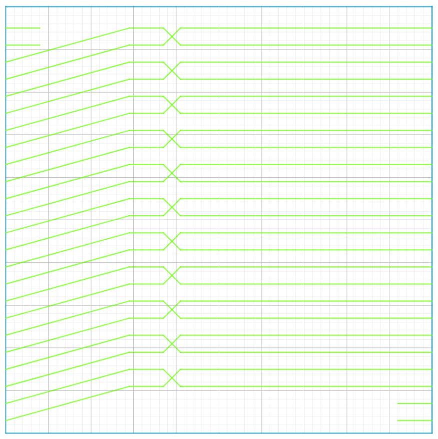

b. sp12-horizontal

Figure 5: Span 12 wires

Span-12

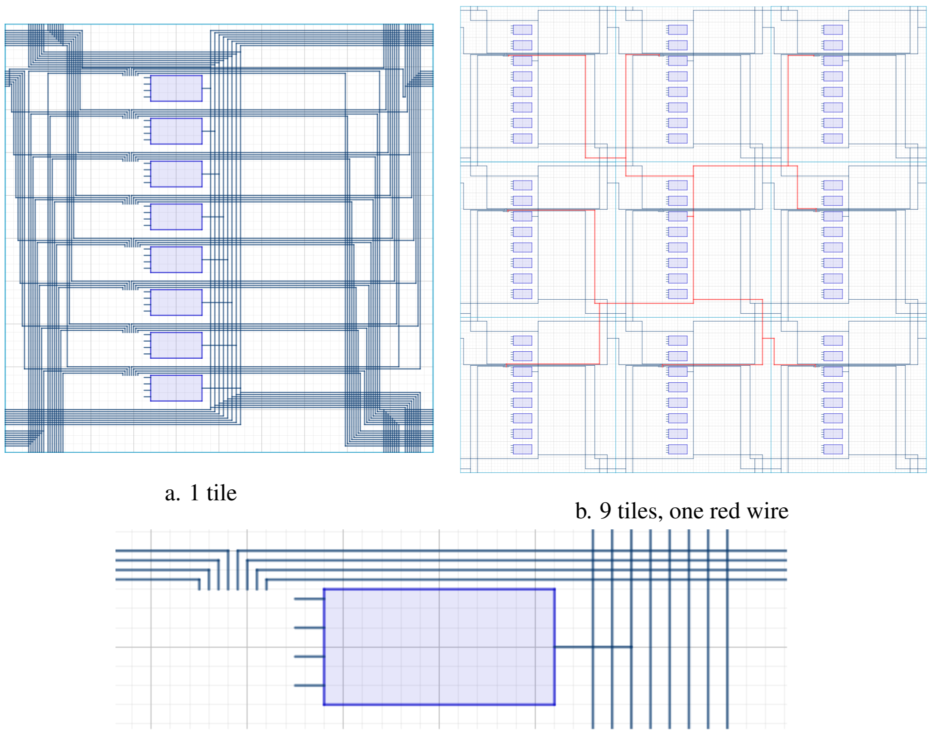
The Span-12 wiring is conceptually similar to the Span-4 wiring, except with twelve tiles instead of four. There are $24+2$ wires through one tile. In Figure 5 is a picture of the wiring.

1.3 Local wires

The other wires are discussed here.

Logic cell, LUT4-output

The Logic Cell, discussed in Section 1.1, has one output wire on the right side. This wire connects all the neighboring cells. In Figure 6 is an image for clarification. Output wires for the sixth Logic Cells are visible in Figure 6b. One wire is red, so we can follow that wire easily.


a. 1 tile

b. 9 tiles, one red wire

c. the 6th Logic Cell enlarged

Figure 6: Output LUT4

Other wiring around the Logic Cell

The three wires on the top right of a Logic Cell are for the clock, enable and the set/reset inputs of the Logic Cell, respectively. This is visible in Figure 7 and Figure 1. For every tile, the wire for the clock is shared per tile. The same for the enable and the set/reset wires input. See Figure 7 for clarification. If we want to reset one Logic Cell, we reset every Logic Cell in the same tile.

On the top left of a Logic Cell are two output-wires. These two wires can connect with the middle two input wires of the logic cell above. For convenience, these two output wires are connected with the input wires.

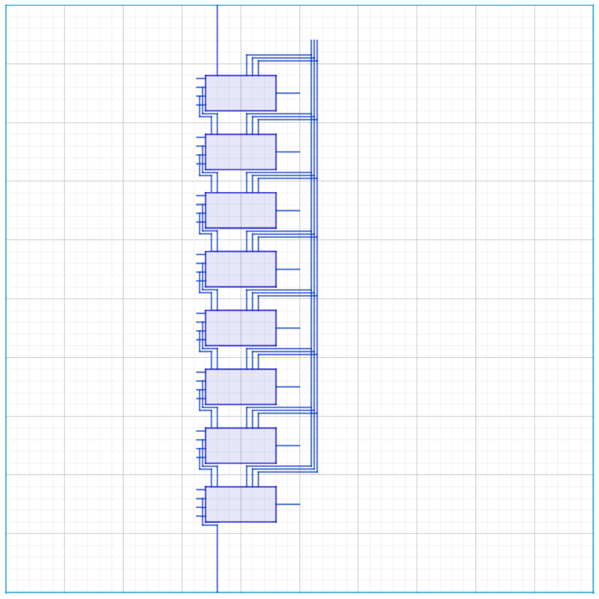


Figure 7: Logic Cell, other wiring

Local network

When we zoom in to see one Logic Tile, we can see 32 local wires, not connecting with other tiles. That is the local network and these wires make it possible to connect different parts of the circuit. There is no Figure, because it only concerns one tile. In Chapter 3 we will discuss which connections can be made.

Global to local wiring

There are four wires that can make a connection from a global wire to a local one. Just like the local network, these wires stay in one tile.

2 Naming of the FPGA wiring

The naming of the wiring has to be clear. In the firestorm documentation [1] this is not always the case, because one wire has different names in different places. We will translate the name to one unique name for clarification. This will be done by a function and its inverse image. We will start with a few definitions.

The space with the naming for the wiring in the firestorm documentation is called *Documentation Name*. NOTATION: $\mathbb{D}_N$. Every name for a part of the wire is an element of $\mathbb{D}_N$.

The space with the naming for the wiring where every wire is called uniquely is called *Unique Name*. NOTATION: $\mathbb{U}_N$ Every name for a wire is an element of $\mathbb{U}_N$

The function will be:

$$f : \mathbb{D}_N \to \mathbb{U}_N$$

We will define the function and the different names step by step, where the Documentation Name is fixed. For defining the function f, another function comes in handy:

Definition 2.0.1. Let X be a set and $A \subseteq X$ a subset. Then *the indicator function* is

$$\mathbb{1}_A : A \to \{0,1\}$$

$$x \mapsto \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

There are 32×32 tiles and you can place them in the positive x,y-plane. The tile name will be the x,y-coordinate of the left bottom corner. See Figure 8 for an example.

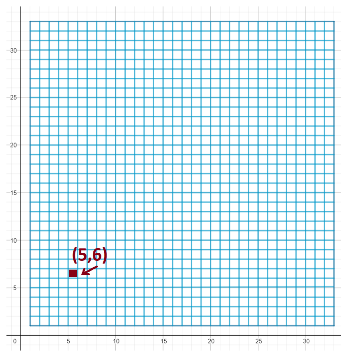


Figure 8: Tile (5,6) colored

2.1 Main wiring

See Section 1.2 to see what the wires are.

Span-4, vertical wiring

There are three sorts of naming Span-4 vertical wiring in $\mathbb{D}_N$. That is:

i) (p,q) sp4_v_b_{0,...47} for wiring on the bottom side of tile (p,q).

ii) (p,q) sp4_v_t_{36,...47} for wiring on the top side of tile (p,q).

iii) (p,q) sp4_r_v_b_{0,...47} for wiring on the right side of tile (p,q).

There is one sort of naming Span-4 vertical wiring in $\mathbb{U}_N$. That is:

i) sp4_p_q_v_$\{0,\dots,11\}$ with p and q variables, (p,q) is the tile where the wire starts. The wire continues on tiles (p',q') with $p-1 \le p' \le p, q-4 \le q' \le q$ and $(p',q') \neq (p-1,q-4)$.

Let $O = \{i \in \mathbb{Z} \mid i \text{ is odd}\}$ be a subset of the integers $\mathbb{Z}$ consisting of odd numbers. Then the function f will be:

$$\begin{aligned}
f\left((p,q)\ \text{sp4\_}v\_b\_k\right) &= \text{sp4\_}p\_(q+j)\_v\_(l+(-1)^l \cdot \mathbb{1}_O(j)) \\
f\left((p,q)\ \text{sp4\_}v\_t\_k\right) &= \text{sp4\_}p\_(q+4)\_v\_(l+(-1)^l) \\
f\left((p,q)\ \text{sp4\_}r\_v\_b\_k\right) &= \text{sp4\_}(p+1)\_(q+j)\_v\_(l+(-1)^l \cdot \mathbb{1}_O(j)) \\
\text{with } j &= \left\lfloor \frac{k}{12} \right\rfloor \text{ en } l = k \mod 12
\end{aligned}$$

Then the inverse set of the function f regarding the Span-4 vertical wires is:

$$\begin{aligned}
f^{-1}(\text{sp4\_}p\_q\_v\_i) = \{ &(p,q-j)\ \text{sp4\_}v\_b\_\left(i+12j+(-1)^i \cdot \mathbb{1}_O(j)\right), \\
&(p,q-4)\ \text{sp4\_}v\_t\_\left(i+36+(-1)^i\right), \\
&(p-1,q-j)\ \text{sp4\_}r\_v\_b\_\left(i+12j+(-1)^i \cdot \mathbb{1}_O(j)\right) \mid j \in \{0,1,2,3\}\}
\end{aligned}$$

Example 2.1.1. As an example the name sp4_17_17_v_0 $\in \mathbb{U}_N$ is equivalent to the following names in $\mathbb{D}_N$

$$\begin{aligned}
f^{-1}(\text{sp4\_17\_17\_}v\text{\_0}) = \{ &(17,17)\ \text{sp4\_}v\_b\text{\_0}, && (16,17)\ \text{sp4\_}r\_v\_b\text{\_0}, \\
&(17,16)\ \text{sp4\_}v\_b\text{\_13}, && (16,16)\ \text{sp4\_}r\_v\_b\text{\_13}, \\
&(17,15)\ \text{sp4\_}v\_b\text{\_24}, && (16,15)\ \text{sp4\_}r\_v\_b\text{\_24}, \\
&(17,14)\ \text{sp4\_}v\_b\text{\_37}, && (16,14)\ \text{sp4\_}r\_v\_b\text{\_37}, \\
&(17,13)\ \text{sp4\_}v\_t\text{\_37}\}
\end{aligned}$$

See Figure 18 in Appendix for a clarifying picture.

Proof. Let's prove that $f\left(f^{-1}(\text{sp4\_}p\_q\_v\_i)\right) = \text{sp4\_}p\_q\_v\_i$:

$$\begin{aligned}
f\left(f^{-1}(\text{sp4\_}p\_q\_v\_i)\right) = \{&f\left((p,q-j)\ \text{sp4\_}v\_b\_\left(i+12j+(-1)^i\cdot\mathbb{1}_O(j)\right)\right),\\
&f\left((p,q-4)\ \text{sp4\_}v\_t\_\left(i+36+(-1)^i\right)\right),\\
&f\left((p-1,q-j)\ \text{sp4\_}r\_v\_b\_\left(i+12j+(-1)^i\cdot\mathbb{1}_O(j)\right)\right) \mid j\in\{0,1,2,3\}\}
\end{aligned}$$

$$\begin{aligned}
= \{&\text{sp4\_}p\_(q-j+j')\_v\_\left(l+(-1)^l\cdot\mathbb{1}_O(j')\right),\\
&\text{sp4\_}p\_(q-4+4)\_v\_\left(i+(-1)^i+(-1)^{i+(-1)^i}\right),\\
&\text{sp4\_}(p-1+1)\_(q-j+j')\_v\_\left(l+(-1)^l\cdot\mathbb{1}_O(j')\right)\\
&: j' = \left\lfloor \frac{i+12j+(-1)^i\cdot\mathbb{1}_O(j)}{12}\right\rfloor, l = i+(-1)^i\cdot\mathbb{1}_O(j)\}
\end{aligned}$$

we know this:

$$j' = \left\lfloor \frac{i+12j+(-1)^i\cdot\mathbb{1}_O(j)}{12}\right\rfloor = \left\lfloor \frac{12j}{12}+\frac{i+(-1)^i\cdot\mathbb{1}_O(j)}{12}\right\rfloor = \left\lfloor j+\frac{i+(-1)^i\cdot\mathbb{1}_O(j)}{12}\right\rfloor = j$$

because $i\in\{0,...,11\}$, so $0\le i+(-1)^i\cdot\mathbb{1}_O(j) < 12$. And we know

$$\begin{aligned}
l+(-1)^l\cdot\mathbb{1}_O(j) &= i+(-1)^i\cdot\mathbb{1}_O(j)+(-1)^l\cdot\mathbb{1}_O(j')\\
&= i+\left((-1)^i+(-1)^{i+(-1)^i\cdot\mathbb{1}_O(j)}\right)\cdot\mathbb{1}_O(j)\\
&= i
\end{aligned}$$

because if j is odd, then i is odd if and only if $i+(-1)^i$ is even. Now we know:

$$\begin{aligned}
f\left(f^{-1}(\text{sp4\_}p\_q\_v\_i)\right) = \{&\text{sp4\_}p\_q\_v\_i,\\
&\text{sp4\_}p\_q\_v\_i,\\
&\text{sp4\_}p\_q\_v\_i\}\\
=\ &\text{sp4\_}p\_q\_v\_i
\end{aligned}$$

Now we need to prove that $(p,q)\ \text{sp4\_}v\_b\_k \in f^{-1}f((p,q)\text{sp4\_}v\_b\_k)$:

$$\begin{aligned}
f^{-1}f((p,q)\text{sp4\_}v\_b\_k) = &f^{-1}\left(\text{sp4\_}p\_(q+j)\_v\_\left(l+(-1)^l\cdot\mathbb{1}_O(j)\right)\right)\\
&\text{with } j = \left\lfloor\frac{k}{12}\right\rfloor, l = k \mod 12
\end{aligned}$$

$$
\begin{aligned}
=\Big\{&(p,q+j-j')\ \text{sp4\_}v\_b\_(l+(-1)^l\cdot \mathbb{1}_O(j)+12j+(-1)^{l+(-1)^l\cdot\mathbb{1}_O(j)}\cdot\mathbb{1}_O(j')),\\
&(p,q+j-4)\ \text{sp4\_}v\_t\_(l+(-1)^l\cdot \mathbb{1}_O(j)+36+(-1)^{l+(-1)^l\cdot\mathbb{1}_O(j')})\\
&(p-1,q+j-j')\ \text{sp4\_}r\_v\_b\_(l+(-1)^l\cdot \mathbb{1}_O(j)+12j+(-1)^{l+(-1)^l\cdot\mathbb{1}_O(j)}\cdot\mathbb{1}_O(j'))\\
&: j=\left\lfloor\frac{k}{12}\right\rfloor, l=k \mod 12, j'\in\{0,1,2,3\}\Big\}
\end{aligned}
$$

When you take $j'=j=\lfloor\frac{k}{12}\rfloor$, then $(p,q+j-j')=(p,q)$ and

$$
\begin{aligned}
&l+(-1)^l\cdot\mathbb{1}_O(j)+12j+(-1)^{l+(-1)^l\cdot\mathbb{1}_O(j)}\cdot\mathbb{1}_O(j)\\
&=l+12j+\Big((-1)^l+(-1)^{l+(-1)^l\cdot\mathbb{1}_O(j)}\Big)\cdot\mathbb{1}_O(j)\\
&=l+12j\\
&=(k \mod 12)+12\left\lfloor\frac{k}{12}\right\rfloor\\
&=k
\end{aligned}
$$

So (p,q) sp4_v_b_$k\in f^{-1}f((p,q)$ sp4_v_b_$k)$.

The proof for (p,q) sp4_v_t_$k\in f^{-1}f((p,q)$ sp4_v_t_$k)$ and
(p,q) sp4_r_v_b_$k\in f^{-1}f((p,q)$ sp4_v_b_$k)$ is analogous. □

Span-4, horizontal wiring

There are two sorts of naming Span-4 horizontal wiring in $\mathbb{D}_N$. That is:

i) (p,q) sp4_h_r_{0,...47} for wiring on the right side of tile (p,q).

ii) (p,q) sp4_h_l_{36,...47} for wiring on the left side of tile (p,q).

There is one sort of naming Span-4 horizontal wiring in $\mathbb{U}_N$. That is:

i) sp4_p_q_h_$\{0,\dots,11\}$ with (p,q) the tile where the wire starts. The wire continues on tiles (p',q') with $p\le p'\le p+4$ and $q'=q$.

The function f regarding the Span-4 horizontal wires will be:

$$
\begin{aligned}
f\left((p,q)\ \text{sp4\_}h\_r\_k\right)&=\text{sp4\_}(p-j)\_q\_h\_(l+(-1)^l\cdot\mathbb{1}_O(j))\\
f\left((p,q)\ \text{sp4\_}h\_l\_k\right)&=\text{sp4\_}(p-4)\_q\_h\_(l+(-1)^l)\\
&\text{with } j=\left\lfloor\frac{k}{12}\right\rfloor \text{ en } l=k \mod 12
\end{aligned}
$$

The inverse f^{-1} will be.

$$f^{-1}(\text{sp4\_}p\text{\_}q\text{\_}h\text{\_}i) = \big\{(p+j,q)\ \text{sp4\_}h\text{\_}r\text{\_}\big(i+12j+(-1)^i\cdot \mathbb{1}_O(j)\big)\,,\\ (p+4,q)\ \text{sp4\_}h\text{\_}l\text{\_}\big(i+36+(-1)^i\big)\,|\,j\in\{0,1,2,3\}\big\}$$

Span-12 wiring

There are four sorts of naming Span-12 wiring in $\mathbb{D}_N$. That is:

i) (p,q) sp12_v_b_{0,...23} for vertical wiring on the bottom side of tile (p,q).

ii) (p,q) sp12_v_t_{22,23} for vertical wiring on the top side of tile (p,q).

iii) (p,q) sp12_h_r_{0,...23} for horizontal wiring on the right side of tile (p,q).

iv) (p,q) sp12_h_l_{22,23} for horizontal wiring on the left side of tile (p,q).

There are two sorts of naming Span-12 wiring in $\mathbb{U}_N$. That is:

i) sp4_p_q_v_$\{0,1\}$ with (p,q) the tile where the vertical wire starts. The wire continues on tiles (p',q') with $p'=p$ and $q-12\le q'\le q$.

ii) sp4_p_q_h_$\{0,1\}$ with (p,q) the tile where the horizontal wire starts. The wire continues on tiles (p',q') with $p\le p'\le p+12$ and $q-12\le q'\le q$.

The function f regarding the Span-12 wires will be:

$$\begin{aligned} f\left((p,q)\ \text{sp12\_}v\text{\_}b\text{\_}k\right) &= \text{sp12\_}p\text{\_}(q+j)\text{\_}v\text{\_}(l+(-1)^l\cdot\mathbb{1}_O(j));\\ f\left((p,q)\ \text{sp12\_}v\text{\_}t\text{\_}k\right) &= \text{sp12\_}p\text{\_}(q+12)\text{\_}v\text{\_}(l+(-1)^l);\\ f\left((p,q)\ \text{sp12\_}h\text{\_}r\text{\_}k\right) &= \text{sp12\_}(p-j)\text{\_}q\text{\_}h\text{\_}(l+(-1)^l\cdot\mathbb{1}_O(j));\\ f\left((p,q)\ \text{sp12\_}v\text{\_}t\text{\_}k\right) &= \text{sp12\_}(p-12)\text{\_}q\text{\_}h\text{\_}(l+(-1)^l);\\ &\text{with } j=\left\lfloor\frac{k}{2}\right\rfloor \text{ en } l=k \mod 2. \end{aligned}$$

The inverse f^{-1} will be:

$$\begin{aligned} f^{-1}(\text{sp12\_}p\text{\_}q\text{\_}v\text{\_}i) = \big\{&(p,q-j)\ \text{sp12\_}v\text{\_}b\text{\_}\big(i+2j+(-1)^i\cdot\mathbb{1}_O(j)\big)\,,\\ &(p,q-12)\ \text{sp12\_}v\text{\_}t\text{\_}\big(i+22+(-1)^i\big)\,|\,j\in\{0,...,11\}\big\};\\ f^{-1}(\text{sp12\_}p\text{\_}q\text{\_}h\text{\_}i) = \big\{&(p+j,q)\ \text{sp12\_}h\text{\_}r\text{\_}\big(i+2j+(-1)^i\cdot\mathbb{1}_O(j)\big)\,,\\ &(p+12,q)\ \text{sp12\_}h\text{\_}l\text{\_}\big(i+22+(-1)^i\big)\,|\,j\in\{0,...,11\}\big\}. \end{aligned}$$

Global

The name for global wiring will be glb_netwk$\{0,...,7\}$. This is the same in de documentation as for the unique name

$$f(\text{glb\_netwk}i) = \text{glb\_netwk}i.$$

2.2 Local wiring

Logic Cell output wire

There are different sorts of naming the Logic Cell output wire in $\mathbb{D}_N$. There is one sort of naming the Logic Cell output wire in $\mathbb{U}_N$:

i) lutff_p_q_i/out with (p,q) the tile where the output wire of the ith Logic Cell starts. The wire continius on tiles (p',q') with $p-1 \leq p' \leq p+1$ and $q-1 \leq q' \leq q+1$.

The function f regarding the Logic Cell output wire will be:

$$\begin{aligned}
f((p,q)\ \text{lutff\_}i\text{/out}) &= \text{lutff\_}p\text{\_}q\text{\_}i \\
f((p,q)\ \text{neigh\_op\_tnr\_}i) &= \text{lutff\_}(p+1)\text{\_}(q+1)\text{\_}i \\
f((p,q)\ \text{neigh\_op\_rgt\_}i) &= \text{lutff\_}(p+1)\text{\_}q\text{\_}i \\
f((p,q)\ \text{neigh\_op\_bnr\_}i) &= \text{lutff\_}(p+1)\text{\_}(q-1)\text{\_}i \\
f((p,q)\ \text{neigh\_op\_top\_}i) &= \text{lutff\_}p\text{\_}(q+1)\text{\_}i \\
f((p,q)\ \text{neigh\_op\_bot\_}i) &= \text{lutff\_}p\text{\_}(q-1)\text{\_}i \\
f((p,q)\ \text{neigh\_op\_tnl\_}i) &= \text{lutff\_}(p-1)\text{\_}(q+1)\text{\_}i \\
f((p,q)\ \text{neigh\_op\_lft\_}i) &= \text{lutff\_}(p-1)\text{\_}q\text{\_}i \\
f((p,q)\ \text{neigh\_op\_bnl\_}i) &= \text{lutff\_}(p-1)\text{\_}(q-1\text{\_}i
\end{aligned}$$

The inverse f^{-1} will be:

$$\begin{aligned}
&f^{-1}(\text{lutff\_}p\text{\_}q\text{\_}i) \\
&= \big\{(p,q)\ \text{lutff\_}i\text{/out}, (p-1,q-1)\text{neigh\_op\_tnr\_}i, (p-1,q)\text{neigh\_op\_rgt\_}i, \\
&\quad (p-1,q+1)\text{neigh\_op\_bnr\_}i, (p,q-1)\text{neigh\_op\_top\_}i, (p,q+1)\text{neigh\_op\_bot\_}i, \\
&\quad (p+1,q-1)\text{neigh\_op\_tnl\_}i, (p+1,q)\text{neigh\_op\_lft\_}i, (p+1,q+1)\text{neigh\_op\_bnl\_}i\big\}
\end{aligned}$$

Example 2.2.1. As an example the name lutff_17_17_5 $\in \mathbb{U}_N$ is equivalent to the following names in $\mathbb{D}_N$

$$\begin{aligned}
f^{-1}(\text{lutff\_17\_17\_5}) = \big\{&(17,17)\text{lutff\_5/out}, (16,16)\text{neigh\_op\_tnr\_5}, \\
&(16,17)\text{neigh\_op\_rgt\_5}, (16,18)\text{neigh\_op\_bnr\_5}, \\
&(17,16)\text{neigh\_op\_top\_5}, (17,18)\text{neigh\_op\_bot\_5}, \\
&(18,16)\text{neigh\_op\_tnl\_5}, (18,17)\text{neigh\_op\_lft\_5}, \\
&(18,18)\text{neigh\_op\_bnl\_5}\big\}
\end{aligned}$$

Rest of the wiring

The rest of the wiring is mainly simple, so we only give the function itself and not the inverse.

$f((p,q)$ carry_in_mux $=$ carry_in_mux_p_q
A wire we did not draw.

$f((p,q)$ carry_in $=$ lutff_p_$(q-1)$_7$/$cout
$f((p,q)$ lutff_$i/$cout) $=$ lutff_p_q_$i/$cout
on the top left, the right output wire of the ith Logic Cell, see Figure 7. And $i \in \{0,..,7\}$

$f((p,q)$ lutff_$i/$in_j $=$ lutff_p_q_$i/$in_j
the right jth input wire of the ith Logic Cell, see Figures 1, 2 and 6c. And $i \in \{0,..,7\}$, $j \in \{0,..,3\}$

$f((p,q)$ lutff_$i/$lout $=$ lutff_p_q_$i/$lout
on the top left, the left output wire of the ith Logic Cell, see Figures 7. And $i \in \{0,..,6\}$

$f((p,q)$ lutff_global$/$cen $=$ lutff_p_q_global$/$cen
on the top right, the middle output wire on every Logic Cell, see Figures 1 and 7

$f((p,q)$ lutff_global$/$clk $=$ lutff_p_q_global$/$clk
on the top right, the right output wire on every Logic Cell, see Figures 1 and 7

$f((p,q)$ lutff_global$/$s_r $=$ lutff_p_q_global$/$s_r
on the top right, the left output wire on every Logic Cell, see Figures 1 and 7

$f((p,q)$ glb2local_i $=$ glb2local_p_q_i
Wires we did not draw that make connections from global to local wires possible. And $i \in \{0,..,3\}$

$f((p,q)$ local_gi_j $=$ local_p_q_gi_j
Wires we did not draw that make connections between different kinds of wires possible. And $i \in \{0,..,3\}$, $j \in \{0,...,7\}$

3 Routing

Some wires can connect with other wires. In this chapter we discuss which connections are possible. Connections will always be from a Source net to a Destination net.

3.1 Global

All possible connections regarding global wires or global2local wires are:

Source net		Destination net
glb_netwk$\{0,1,2,3,4,5,6,7\}$	$\mapsto$	glb2local_p_q_$\{0,1,2,3\}$
glb_netwk$\{1,3,5,7\}$	$\mapsto$	lutff_p_q/global/cen
glb_netwk$\{0,1,2,3,4,5,6,7\}$	$\mapsto$	lutff_p_q_global/clk
glb_netwk$\{0,2,4,6\}$	$\mapsto$	lutff_p_q_global/s_r
glb2local_p_q_i with $i \in \{0,...,3\}$	$\mapsto$	local_p_q_g0_$(i+4)$

This table tells us that global2local wires exists only for connecting a global network wire to a local wire.

3.2 Local Cell wires

Logic Cell output wire

First I give you the output wire of a Logic Cell. Again it is a table from the Source net to the Destination net.

Source net		Destination net
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	local_$(p-1)$_$(q+1)$_g$\{0,1\}$_i
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	local_(p)_$(q-1)$_g$\{0,1\}$_i
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	local_(p)_$(q+1)$_g$\{0,1\}$_i
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	local_$(p+1)$_(q)_g$\{0,1\}$_i
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	local_$(p-1)$_$(q-1)$_g$\{2,3\}$_i
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	local_$(p-1)$_(q)_g$\{2,3\}$_i
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	local_$(p+1)$_$(q-1)$_g$\{2,3\}$_i
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	local_$(p+1)$_$(q+1)$_g$\{2,3\}$_i
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	local_(p)_(q)_g$\{0,1,2,3\}$_i
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	(p,q) sp4_v_$b$$\{2(i+8k)\}$ with $k \in \{0,1,2\}$
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	(p,q) sp4_r_v_$b$$\{2(i+8k)+1\}$ with $k \in \{0,1,2\}$
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	(p,q) sp4_h_$r$$\{2(i+8k)\}$ with $k \in \{0,1,2\}$
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	(p,q) sp12_v_$b$$\{2(i+8k)\}$ with $k \in \{0,1\}$. If $i \geq 4$, then $k=0$
lutff_p_q_i/out with $i \in \{0,...,7\}$	$\mapsto$	(p,q) sp12_h_$r$$\{2(i+8k)-8\}$ with $k \in \{0,1\}$. If $i \leq 3$, then $k=1$

Logic Cell input wires

Every connection regarding lutff_p_q_$\{0,1,2,3,4,5,6,7\}$/in_$\{0,..,3\}$ is in the table below. In the table we have the Destination net on the left and the Source net on the right. The reason is clarity. Connections on the FPGA will be from the Source net to the Destination net.

Destination net		Source net
lutff_p_q_k/in_i	$\hookleftarrow$	local_p_q_gm_$(2n+\mathbb{1}_{k+i+n \text{ is oneven}})$ with $m \in \{0,..,3\}$
with $k \in \{0,..,7\}$ and $i \in \{0,..,3\}$		and $n \in \{0,..,4\}$ if $i = 3$, then $(m,n) \notin \{(0,0),(0,1)\}$
lutff_p_q_0/in_3	$\hookleftarrow$	carry_in_mux_p_q
lutff_p_q_k/in_3 with $k \in \{1,..,7\}$	$\hookleftarrow$	lutff_p_q_$(k-1)$/cout
lutff_p_q_k/in_2 with $k \in \{1,..,7\}$	$\hookleftarrow$	lutff_p_q_$(k-1)$/lout

Logic Cell, rest of the wiring

All possible connections with lutff_p_q_$\{0,...7\}$/cout, lutff_p_q_$\{0,...6\}$/lout, carry_in_mux, lutff_p_q/global/cen, lutff_p_q/global/clk and lutff_p_q/global/s_r are shown in the table below.

Source net		Destination net
lutff_p_q_i/cout with $i \in \{0,..,6\}$	$\mapsto$	lutff_p_q_$(i+1)$/in_3
lutff_p_q_7/cout	$\mapsto$	carry_in_mux_p_$(q+1)$
carry_in_mux_p_q	$\mapsto$	lutff_p_q_0/in_3
lutff_p_q_i/lout with $i \in \{0,..,6\}$	$\mapsto$	lutff_p_q_$(i+1)$/in_2

Destination net		Source net
lutff_p_q/global/cen	$\hookleftarrow$	glb_netwk$\{1,3,5,7\}$
lutff_p_q/global/cen	$\hookleftarrow$	local_p_q_g$(i-2k)$_i with $i \in \{2,3\}$ and $k \in \{0,1\}$
lutff_p_q/global/clk	$\hookleftarrow$	glb_netwk$\{1,2,3,4,5,6,7\}$
lutff_p_q/global/clk	$\hookleftarrow$	local_p_q_g$(i+2k)$_i with $i \in \{0,1\}$ and $k \in \{0,1\}$
lutff_p_q_global/s_r	$\hookleftarrow$	glb_netwk$\{0,2,4,6\}$
lutff_p_q/global/s_r	$\hookleftarrow$	local_p_q_g$(i-2(k+1))$_i with $i \in \{4,5\}$ and $k \in \{0,1\}$

3.3 Local wiring

In the following table are the connections of the local wires which are not yet discussed. That are the connections between the local network and the Span-4 and Span-12 wires.

Destination net		Source net
local_p_q_g$\{0,1\}$_i with $i \in \{0,...,7\}$	$\hookleftarrow$	(p,q) sp4_v_$b\{i,8+i,16+i\}$
local_p_q_g$\{0,1\}$_i with $i \in \{0,...,7\}$	$\hookleftarrow$	(p,q) sp4_h_$r\{i,8+i,16+i\}$
local_p_q_g$\{0,1\}$_i with $i \in \{0,...,7\}$	$\hookleftarrow$	(p,q) sp12_h_$r\{i,8+i,16+i\}$
local_p_q_g$\{2,3\}$_i with $i \in \{0,...,7\}$	$\hookleftarrow$	(p,q) sp4_v_$b\{28+i,32+i,40+i\}$
local_p_q_g$\{2,3\}$_i with $i \in \{0,...,7\}$	$\hookleftarrow$	(p,q) sp4_h_$r\{28+i,32+i,40+i\}$
local_p_q_g$\{2,3\}$_i with $i \in \{0,...,7\}$	$\hookleftarrow$	(p,q) sp12_v_$b\{i,8+i,16+i\}$
local_p_q_g0_i with $i \in \{0,...,7\}$	$\hookleftarrow$	(p,q) sp4_r_v_$b(24-i)$
local_p_q_g0_i with $i \in \{0,...,3\}$	$\hookleftarrow$	(p,q) sp4_r_v_$b(35-i)$
local_p_q_gk_i with $k \in \{1,2,3\}$ and $i \in \{0,...,7\}$	$\hookleftarrow$	(p,q) sp4_r_v_$b\{8k+i,24+8k+i\}$

4 Independent path problem

A lot of connections have to be made in an FPGA. From different switches through multiple Logic Cells to the led lights. And we want to know the fastest route. A connection between two wires slows down the route. It is like there is a traffic light between the two different wires. Just like driving a car, the shortest route is not always the fastest. To conquer this problem, we need to make a mathematical model. A model that can handle simple examples and a model we can make more complex. We will use graph theory. The definitions are in line with [2].

Definition 4.0.1. A *graph* G is an ordered pair (V,E) with V a finite set and $E \subseteq \wp(V)$ with $\forall B \in E, \#B = 2$. The elements in V are called the *points* of G or the *vertices* of G. The elements of E are called the *edges* of G.

Definition 4.0.2. A *directed graph* G is an ordered pair (V,E) with V a finite set and $E \subseteq V \times V$ with if $(v,w) \in E$, then $v \neq w$. The elements in V are called the *points* of G or the *vertices* of G. The elements of E are called the *arrows* of G.

Below is an example of a directed graph.

$$V = \{a,b,c,d,e,f,g,h,i,j\} \text{ and}$$
$$E = \{(a,d),(b,d),(b,e),(c,f),(d,c),(d,e),(d,g),(e,h),(f,i),(h,g),(h,j),(i,j)\}$$

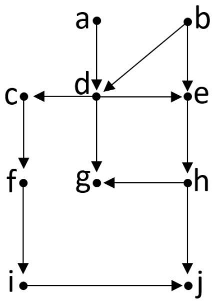


Figure 9: Directed graph

When we translate this back to an FPGA, the elements of V are the wires in an FPGA and an element of E is a connection from a source wire to a destination wire. We want to know the fastest route from wire a to wire b. This corresponds to finding the shortest path from point a to point b in the model.

Definition 4.0.3. A *directed path* in a directed graph $G = (V,E)$ is a sequence of points $(a_0, a_1, ..., a_k)$ with $(a_{i-1}, a_i) \in E \ \ \forall i \in \{1,..,k\}$ and $a_j \neq a_i \ \ \forall i \neq j$. The *length* of this path is k. The *starting point* is a_0 and a_k is the *end point*.

Still we are not satisfied. Alongside a path from a to b, we want to find a path from point c to d. And to translate this back correctly, we want independence between the power in the path from a to b and the power running from path c to d. We make the following definition:

Definition 4.0.4. Let $G = (V,E)$ be a graph and $\alpha = (a_0,...,a_k)$, $\beta = (b_0,..,b_l)$ be paths in G. Then α and β are *vertex-disjoint* if $a_i \neq b_j \quad \forall i \in \{0,...,k\}, j \in \{0,...,l\}$. The *length* of paths a and b combined is $k+l$.

Definition 4.0.5. Let $G = (V,E)$ be a graph and $\alpha_1,...,\alpha_n$ be paths in G with length l_i for path α_i. This set of paths are *vertex-disjoint* if α_i and α_j are vertex-disjoint for all $i \neq j$. The *length* is $\sum_{i=1}^{n} l_i$.

Now we can formulate our problem correctly.

Short vertex disjoint paths-problem Let $G = (V,E)$ be a graph and $a_0,....,a_n$ be starting points and $b_0,...,b_n$ be end points. What are the shortest vertex-disjoint paths from beginpoints a_i to endpoints b_i?

There are multiple efficient algorithms to find a solution with one path. Unfortunately we can not extend that algorithm to find two or more vertex independent paths. The example below will clarify this.

Example 4.0.6. Take the graph in Figure 9 and assume we want to find a path α from a to j that is independent to a path β from b to g. First we find the shortest path from a to j and then we try to find β. Finding a path from b to g that is independent from α is not possible. Second we find the shortest path from b to g and then we try to find α. This too will fail. But there do exist independent paths α and β, see Figure 10.

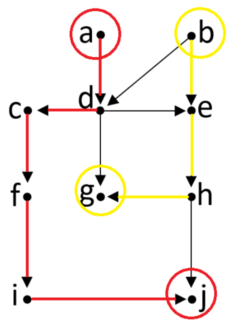


Figure 10: Shortest paths

It seems like, we need to find the paths α and β at the same time. And we think it is not as simple as this looks. For that we have Chapter 5.

5 NP Hardness

We want to know if there exists an algorithm that will find a solution to the short vertex disjoint path-problem discussed in Chapter 4 with the condition that the algorithm will still be fast when the graph becomes bigger. One way to prove this algorithm exists is to find the algorithm. The algorithm discussed in Chapter 4 failed. That could be bad luck or an indication that there is no fast algorithm. To prove there is no fast algorithm, we need to introduce some definitions first. The definitions and are in line with [3]

5.1 Introduction to complexity

The definition of an elementary step we will leave vague, because we need the definition of Turing machines for this. A computer is an example of a Turing machine. And we do not want to go that far for the definitions. In most cases adding one bit is an example of an elementary step. Elementary steps do not depend on the size of the input variables. Hopefully the feeling of an elementary step will become clear after an example.

Definition 5.1.1. The *Time complexity* of an algorithm is a function. The input size of the function is the number of bits to write down the input of the algorithm. The outcome of the fuction is the maximal number of elementary steps it takes to complete the algorithm.

Example 5.1.2. Let's make an algorithm that will check if an n times n sudoku S has filled the rows correctly.

```
(1)   for (r = 1; r <= n; r++){
(2)       for (i = 1; i <= n; i++){
(3)           number = 0;
(4)           for (c = 1; number == 1; c++){
(5)               if (S[r][c] == i){
(6)                   number = 1; }
(7)               if (i == n && c == n){
(8)                   return false; }
(9)           }
(10)      }
(11)  }
(12)  return true;
```

We check every row and every number if it is somewhere in the row. Line (5) checks if the number i is the same as in the sudoku on row r and column c. Steps (3), (5) to (8) and (12) have elementary step 1.

- In (4) we see that the algorithm computes maximal n times steps (5) to (9). This comes down to $4n$ elementary steps.

- In (2) we see that the algorithm computes maximal n times steps (3) to (9). This comes down to $n \cdot 4n$ elementary steps.

- In (1) we see that the algorithm computes maximal n times steps (2) to (9). This comes down to $n \cdot 4n^2$ elementary steps.

The maximal total of elementary steps is: $4n^3 + 1$. The time complexity is at order n^3.

Definition 5.1.3. A problem is in *Polynomial time/the class* P if there exists an algorithm to solve the problem with Time complexity a polynomial.

Definition 5.1.4. A problem A is in *Non deterministic polynomial time/the class* NP if the decision problem B to check if a given certificate/input is a solution to the problem A is in the class P.

Definition 5.1.5. A decision problem of kind A *reduces to* a decision problem of kind B, NOTATION: $A \leq_P B$ if there exists a polynomial time algorithm f that can translate any problem of kind A to a problem of kind B: a is the input for a problem of kind A is translated to $b = f(a)$, input value to a problem of kind B. The algorithm must have the following property: a is true *iff* b is true. We say B is *at least as hard* as A.

Problem B in the previous defnition *is at least as hard* as problem A, because when we have an algorithm to solve problem B, we also have an algorithm to solve problem A: First translate A to B through algorithm f and follow the algorithm for solving B.

There are problems which are believed to be hard to solve:

Definition 5.1.6. A problem A is *NP-Hard* if the problem is at least as hard as any problem in NP.

To prove that a problem is NP-hard the following theorem will help:

Theorem 5.1.7. [3] If A is NP-hard and $A \leq_P B$ then B is NP-hard.

Proof. Let A be NP-hard with the property that $A \leq_P B$. Thus there exist an algorithm f with polynomial time complexity and with the property that a-input value for problem A is true *iff* $f(a)$-input value for problem B is true.

Let C be a given decision problem. Because A is NP-hard, there exist an algorithm g with polynomial time complexity and with the property c-input value for problem C is true *iff* $g(c)$-input value for problem A is true.

This means there exist an algorithm h with polynomial time complexity and with the property c-input value for problem C is true *iff* $h(c)$-input value for problem B is true. Namely first follow algorithm g and then algorithm f. Then $h(c) = f(g(c))$. And this has the desired property.

B is at least as hard as any problem in NP. This means that B is NP-hard. □

Definition 5.1.8. A problem A is *NP-complete* if the problem is NP-hard and in the class of NP.

We will formulate the short vertex disjoint paths-problem a little bit differently.

Vertex disjoint paths-problem Let $G = (V,E)$ be a graph and $a_0,....,a_n$ be starting points and $b_0,...,b_n$ be end points. Do vertex-disjoint paths from starting points a_i to end points b_i exist?

We want to prove that this problem is NP-complete. For that we need to prove that the vertex disjoint paths-problem is in NP and is NP-hard.

Proposition 5.1.9. The vertex disjoint paths-problem is in the class of NP.

Proof. Let $G = (V,E)$ be a directed graph and $a_1,..,a_n$ be begin points and $b_1,..,b_n$ the end points. When we have the paths from a_i to b_i for $i = 1,...,n$, then we need to check two things for verifying a solution:

1. Is a given path $(a_i,...,b_i)$ a path in the graph $G = (V,E)$?
   Let $(c_1,...,c_k)$ be a given path. We need to check if $\forall 1 \leq i \leq k-1 \ \ (c_i,c_{i+1}) \in E$.
2. Are the paths disjoint?
   For every path $(a_i,...,b_i)$, make this as a set: $S_i = \{c :$ the path $(a_i,...,b_i)$ runs trough vertex $c\}$. And make $S = \bigcup_{i=1,...n} S_i$. Check for $i = 1,...,n$ and for all elements $c \in S_i$ if $c \in S$. If that is true, remove that element from S, so: $S = S \backslash \{c\}$. If it is not true, then the paths are not vertex disjoint.

Both steps have time complexity a polynomial, so the vertex disjoint paths problem is in the class of NP. ☐

We need more information before we can prove that the vertex disjoint paths problem is NP-Hard. We need to find a problem A we know is NP-Hard. And we need a construction/algorithm as in Definition 5.1 where B is the vertex disjoint paths-problem. We know that the following decision problem is NP-Hard:

3 colorability problem[4] Let $G = (V,E)$ be a graph. Is it possible to color the vertices with three colors such that if $(v_i,v_j) \in E$, the color of v_i is not the same as the color of v_j?

We will make a construction that will help reducing the 3 colorability problem to the vertex disjoint path problem.

5.2 Construction

Let $G = (V,E)$ be a graph. We want to look if graph G is 3-colorable with the colours 'yellow', 'red' and 'blue'. To do so, we will construct a new graph $G' = (V',E')$. This we will do in two parts, with the following example:

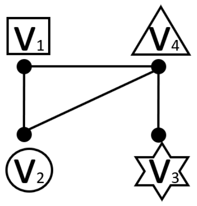


Figure 11: Example graph G

For Part one, we will make $\#V$ directed subgraphs, one for every vertex $v_i \in V$. One subgraph will contain 'copies' of a point $v_i \in V$. We will generate a 'copy' for every edge in E with point v_i, consisting of three 'color points', representing the three colors. Vertex $v_2 \in V$ of example in Figure 11 has two edges with that point, namely $\{v_1, v_2\}, \{v_2, v_4\} \in E$. Then the outcome is in Figure 12. The color points are $c_{2,1,b}$, $c_{2,1,r}$, $c_{2,1,y}$, $c_{2,4,b}$, $c_{2,4,r}$, $c_{2,4,y}$

In Part two, we will make one graph out of the subgraphs generated in part one. We will connect the color points with new points, such that it can help us reducing the 3-colorability problem to the simple vertex disjoint paths-problem. For every $e \in E$ we will add 5 points and 12 arrows.

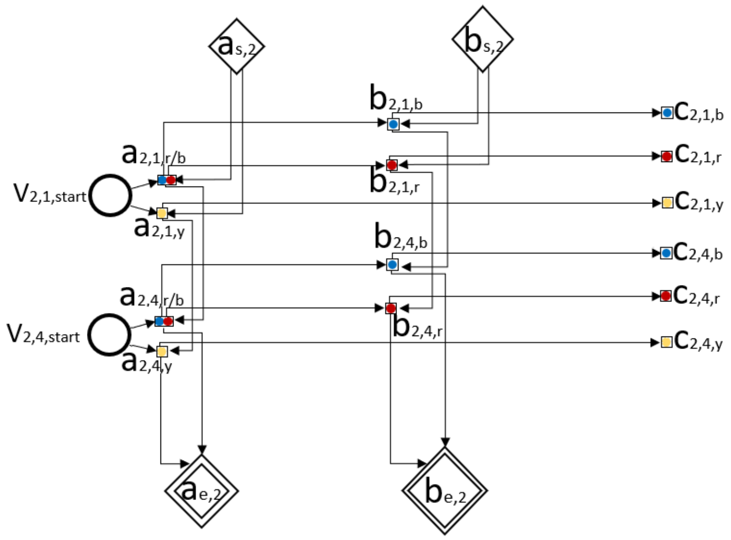


Figure 12: Part one of example with $i = 2$

Algorithm for Part one:

Input: Undirected graph $G = (V, E)$, with $V = v_1, ..., v_n$ and $E \subseteq \{\{v_i, v_j\} | 1 \leq i < j \leq n\}$

Output: n directed graphs, one for every $v_i \in V$, $G_i = (V_i, E_i)$

Step 0. For every $i \in \{1,2,...,n\}$ let $V_i = \{a_{s,i}, a_{e,i}, b_{s,i}, b_{e,i}\}$ and put $E_i = \emptyset$

Step 1. For every pair $\{i, j\}$ such that $\{v_i, v_j\} \in E$:

i. Add 8 points $v_{i,j,start}$, $a_{i,j,y}$, $a_{i,j,r/b}$, $b_{i,j,r}$, $b_{i,j,b}$, $c_{i,j,y}$, $c_{i,j,r}$, $c_{i,j,b}$ to V_i

ii. Add 7 arrows $(v_{i,j,start}, a_{i,j,r/b})$, $(v_{i,j,start}, a_{i,j,y})$, $(a_{i,j,r/b}, b_{i,j,r})$, $(a_{i,j,r/b}, b_{i,j,b})$, $(a_{i,j,y}, c_{i,j,y})$, $(b_{i,j,r}, c_{i,j,r})$, $(b_{i,j,b}, c_{i,j,b})$ to E_i

Step 2. For $i \in \{1,2,...,n\}$: let $J_i \subseteq \{1,2,...,n\}$ such that $j \in J_i \iff \{v_i, v_j\} \in E$ or $(v_j, v_i) \in E$. Write $J_i = \{j_1, j_2, ..., j_k\}$ with $1 \leq j_1 < j_2 < .. < j_k \leq n$ and k depends on i. Add the following arrows to E_i

i. $(a_{s,i}, a_{i,j_1,y}), (a_{i,j_1,y}, a_{i,j_2,y}), ..., (a_{i,j_{k-1},y}, a_{i,j_k,y}), (a_{i,j_k,y}, a_{e,i})$

ii. $(a_{s,i}, a_{i,j_1,r/b}), (a_{i,j_1,r/b}, a_{i,j_2,r/b}), ..., (a_{i,j_{k-1},r/b}, a_{i,j_k,r/b}), (a_{i,j_k,r/b}, a_{e,i})$

iii. $(b_{s,i}, b_{i,j_1,r}), (b_{i,j_1,r}, b_{i,j_2,r}), ..., (b_{i,j_{k-1},r}, b_{i,j_k,r}), (b_{i,j_k,r}, b_{e,i})$

iv. $(b_{s,i}, b_{i,j_1,b}), (b_{i,j_1,b}, b_{i,j_2,b}), ..., (b_{i,j_{k-1},b}, b_{i,j_k,b}), (b_{i,j_k,b}, b_{e,i})$

End of algorithm for Part one.

In Figure 13, we have graph $G' = (V', E')$ for graph $G = (V, E)$ as in Figure 11. This is after part two.

Algorithm for Part two:

Input: Undirected graph $G = (V, E)$ with $V = v_1, ..., v_n$ and $E \subseteq \{\{v_i, v_j\} | 1 \leq i < j \leq n\}$. For every $i \in \{1,2,...,n\}$ a directed graph $G_i = (V_i, E_i)$ as we have after part one.

Output: Directed graph $G' = (V', E')$

Step 0. Let $V' = \bigcup_{1 \leq i \leq n} V_n$ and $E' = \bigcup_{1 \leq i \leq n} E_n$

Step 1. For every pair (i, j) such that $(v_i, v_j) \in E$:

i. Add five points $d_{i,j,b}$, $d_{i,j,r}$, $d_{i,j,y}$, $v_{i,j,e}$, $v_{j,i,e}$ to V'.

ii. Add twelve arrows $(c_{i,j,b}, d_{i,j,b})$, $(c_{j,i,b}, d_{i,j,b})$, $(c_{i,j,r}, d_{i,j,r})$, $(c_{j,i,r}, d_{i,j,r})$, $(c_{i,j,y}, d_{i,j,y})$, $(c_{j,i,y}, d_{i,j,y})$, $(d_{i,j,b}, v_{i,j,e})$, $(d_{i,j,r}, v_{i,j,e})$, $(d_{i,j,y}, v_{i,j,e})$, $(d_{i,j,b}, v_{j,i,e})$, $(d_{i,j,r}, v_{j,i,e})$, $(d_{i,j,y}, v_{j,i,e})$ to E'.

End of algorithm for Part two.

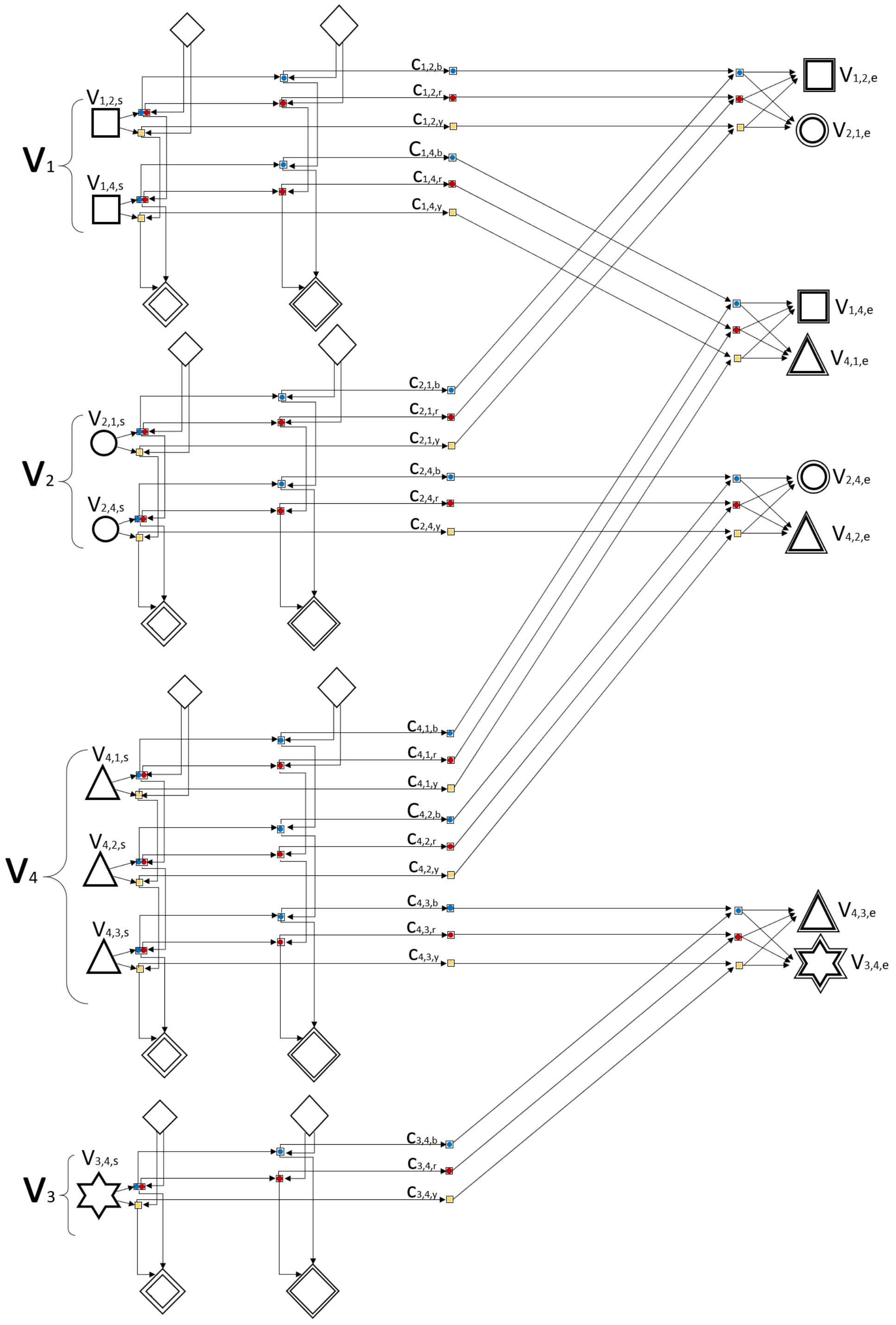


Figure 13: Graph G' for example graph G

5.3 End of proof

Lemma 5.3.1. The algorithm of the construction discussed in Subsection 5.2 is in polynomial time.

Proof. Let $G = (V,E)$ with $\#V = n$. Adding an element to a set is one elementary step. The complexity of the algorithm for part one:

Step 0. For every $i \in \{1,2,...,n\}$ it has 5 elementary steps. This brings a total of $5n$ steps.

Step 1. For every element in E, we add 16 points to a set vertices and we add 14 elements to a set arrows. This is $(16+14)\cdot\#E$. The worst case scenario is when $G = (V,E)$ is a complete graph. Then $\#E = \frac{n\cdot(n-1)}{2}$. This leads to a total of at most $15n(n-1)$ elementary steps.

Step 2. Vertex $i \in \{1,2,...,n\}$ has k connections. For every step i. to iv. we have k elementary steps. In the worst case, when G is a complete graph, $k = n-1$. For every vertex $4(n-1)$ is the maximal number of elementary steps. The total is $4n(n-1)$.

The complexity of the algorithm for part two:

Step 0. We skip this one, because by doing part one, we already have V' and E'.

Step 1. For every element in E, we add 5 points to V' and twelve arrows to E'. This is $17\cdot\#E$. In the worst case, we need $\frac{17n(n-1)}{2}$ elementary steps.

In the worst case scenario, part one has $5n+15n(n-1)+4n(n-1) = 19n^2-14n$ elementary steps and part two has $\frac{17n(n-1)}{2}$. Only the highest power matters and a constant before that is irrelevant, so the time complexity is n^2. The algorithm is in polynomial time. □

Lemma 5.3.2. Let $G = (V,E)$ a graph and $G' = (V',E')$ the directed graph constructed by the algorithm. If for all $\{i,j\} \in E$ exist vertex disjoint paths from $v_{i,j,s}$ to $v_{i,j,e}$, for all $i \in V$ from $a_{s,i}$ to $a_{e,i}$ and for all $i \in V$ from $b_{s,i}$ to $b_{e,i}$, then for every $\{v_i,v_j\} \in E$ one of the following is true:

The path from $v_{i,j,s}$ to $v_{i,j,e}$ is:

i. $(v_{i,j,s},\ a_{i,j,r/b},\ b_{i,j,r},\ c_{i,j,r},\ d_{\min\{i,j\},\max\{i,j\},r},\ v_{i,j,e})$: The 'red' path.

ii. $(v_{i,j,s},\ a_{i,j,r/b},\ b_{i,j,b},\ c_{i,j,b},\ d_{\min\{i,j\},\max\{i,j\},b},\ v_{i,j,e})$: The 'blue' path.

iii. $(v_{i,j,s},\ a_{i,j,y},\ c_{i,j,y},\ d_{\min\{i,j\},\max\{i,j\},y},\ v_{i,j,e})$:1 The 'yellow' path.

Proof. We want to make a path $a_{s,i}$ to $a_{e,i}$. From starting point $a_{s,i}$ there are two possibilities for the next vertex in the path, namely $a_{i,1,y}$ and $a_{i,1,r/b}$. From the second vertex, the path is fixed. Otherwise the path will never meet the endpoint $a_{e,i}$. So the two possibilities for the paths are:

1. $(a_{s,i},\ a_{i,1,y},...,\ a_{i,k-1,y},\ a_{i,k,y},\ a_{e,i})$

2. $(a_{s,i},\ a_{i,1,r/b}, \ldots,\ a_{i,k-1,r/b},\ a_{i,k,r/b},\ a_{e,i})$

We want to make a path $b_{s,i}$ to $b_{e,i}$. Analogous arguments will lead to the two possibilities:

3. $(b_{s,i},\ b_{i,1,b}, \ldots,\ b_{i,k-1,b},\ b_{i,k,b},\ b_{e,i})$
4. $(b_{s,i},\ b_{i,1,r}, \ldots,\ b_{i,k-1,r},\ b_{i,k,r},\ b_{e,i})$

When we make a vertex disjoint path from $v_{i,j,s}$ to $v_{i,j,e}$, it depends on the paths from $a_{s,i}$ to $a_{e,i}$ and from $b_{s,i}$ to $b_{e,i}$.

i. If we choose the possibilities 1. and 2., then the path will be the red path: From $v_{i,j,s}$ the path can only go to $a_{i,j,r/b}$, because the vertex $a_{i,j,y}$ is part of another path. From $a_{i,j,r/b}$ the path can only go to $b_{i,j,r}$, because the vertex $b_{i,j,b}$ is part of another path and when adding vertex $a_{i,j+1,y}$ to the path, the path will never meet the endpoint $v_{i,j,e}$. The rest is fixed.

ii. If we choose the possibilities 1. and 3., then the path will be the blue path. The arguments are analogous.

iii. If we choose the possibilities 2. and (3. or 4.), then the path will be the yellow path. The arguments are analogous.

Because for every $v_i \in V$, the choice for path $a_{s,i}$ to $a_{e,i}$ and $b_{s,i}$ to $b_{e,i}$ are the same. This concludes the proof. $\square$

Proposition 5.3.3. The vertex disjoint paths-problem is NP-Hard.

Proof. We know that the 3 colorability problem is NP-Hard. Let $G = (V,E)$ a graph. Construct a graph $G' = (V',E')$. We know from Lemma 5.3.1 that the construction is in polynomial time, so we only need to prove this:

$$\begin{aligned} \text{Graph } G=(V,E) \text{ is 3-colorable} \iff\ & \text{there exist vertex disjoint paths in } G' \\ & \text{from } v_{i,j,s} \text{ to } v_{i,j,e}\ \forall\{v_i,v_j\} \in E \\ & \text{from } a_{s,i} \text{ to } a_{e,i}\ \forall v_i \in V \\ & \text{from } b_{s,i} \text{ to } b_{e,i}\ \forall v_i \in V \end{aligned}$$

$\Rightarrow$:
Assume graph $G = (V,E)$ is 3-colorable and that a proper 3-coloring is given with the colors red, blue and yellow, then we know the paths:

i. If v_i has color 'red', make the following paths:

$$(v_{i,j,s},\ a_{i,j,r/b},\ b_{i,j,r},\ c_{i,j,r},\ d_{\min\{i,j\},\max\{i,j\},r},\ v_{i,j,e})\ \forall j \text{ with } \{v_i,w_j\} \in E$$

$$(a_{s,i},\ a_{i,1,y}, \ldots,\ a_{i,k-1,y},\ a_{i,k,y},\ a_{e,i})$$
$$(b_{s,i},\ b_{i,1,b}, \ldots,\ b_{i,k-1,b},\ b_{i,k,b},\ b_{e,i})$$

As you can see these paths are vertex disjoint.

ii. If v_i has color 'blue', make the following paths:

$$(v_{i,j,s},\ a_{i,j,r/b},\ b_{i,j,b},\ c_{i,j,b},\ d_{\min\{i,j\},\max\{i,j\},b},\ v_{i,j,e})\ \forall j \text{ with } \{v_i, w_j\} \in E$$

$$(a_{s,i},\ a_{i,1,y}, \ldots,\ a_{i,k-1,y},\ a_{i,k,y},\ a_{e,i})$$
$$(b_{s,i},\ b_{i,1,r}, \ldots,\ b_{i,k-1,r},\ b_{i,k,r},\ b_{e,i})$$

As you can see these paths are vertex disjoint.

iii. If v_i has color 'yellow', make the following paths:

$$(v_{i,j,s},\ a_{i,j,y},\ b_{i,j,y},\ c_{i,j,y},\ d_{\min\{i,j\},\max\{i,j\},y},\ v_{i,j,e})\ \forall j \text{ with } \{v_i, w_j\} \in E$$

$$(a_{s,i},\ a_{i,1,r/b}, \ldots,\ a_{i,k-1,r/b},\ a_{i,k,r/b},\ a_{e,i})$$
$$(b_{s,i},\ b_{i,1,b}, \ldots,\ b_{i,k-1,b},\ b_{i,k,b},\ b_{e,i})$$

As you can see these paths are vertex disjoint.

Now the only way that the paths are not vertex disjoint is when $d_{\min\{i,j\},\max\{i,j\},\text{colour } c}$ is in the two paths from $v_{i,j,s}$ to $v_{i,j,e}$ and $v_{j,i,s}$ to $v_{j,i,e}$. That means that $\{v_i, v_j\} \in V$ and the color c for v_i is the same as for v_j, but graph G is properly 3-colored. Contradiction!
$\Leftarrow$:
Assume graph G' has the vertex disjoint paths. Lemma 5.3.2 says that for every $v_i \in V$ there exists a path of one color. Give the color of that path to the vertex v_i. When $\{v_i, v_j\} \in E$ are colored with the same color c, then both paths $v_{i,j,s}$ to $v_{i,j,e}$ and $v_{j,i,s}$ to $v_{j,i,e}$ has the vertex $d_{\min\{i,j\},\max\{i,j\},c}$ in the path, this is in contradiction with that the paths being disjoint, so this is a well-defined coloring for the graph G. □

Theorem 5.3.4. The vertex disjoint paths-problem is NP-complete

Proof. From Proposition 5.3.3 we see that the vertex disjoint paths-problem is NP-Hard. And from Proposition 5.1.9 we know that the vertex disjoint paths-problem is in the class of NP. □

References

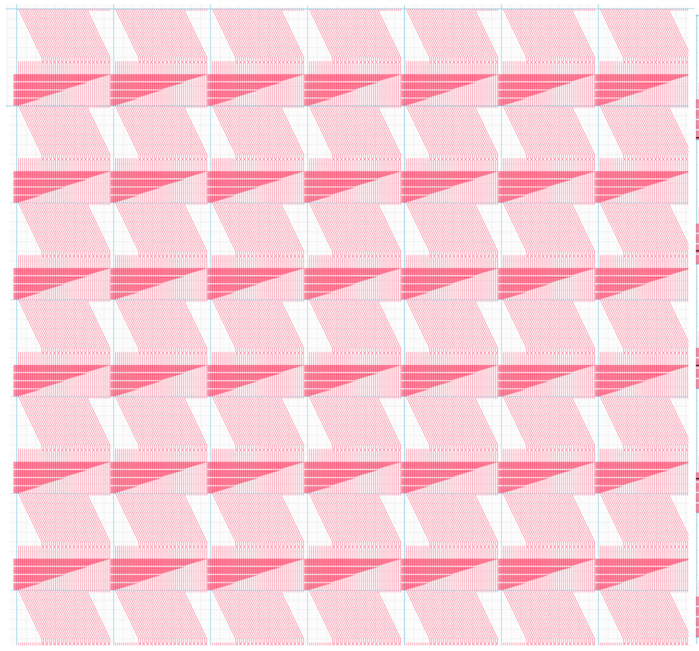
[1] Wolf, C. & Lasser, M., Project IceStorm (2015)
Retrieved from: http://www.clifford.at/icestorm/

[2] Schrijver, A., Grafen: Kleuren en Routeren
Retrieved from: https://homepages.cwi.nl/ lex/files/graphs1_3.pdf

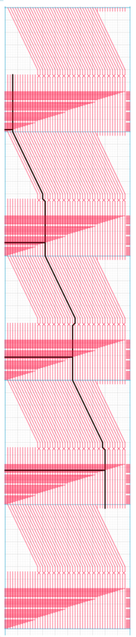
[3] Zantema, H., Lecture notes for the course Complexity IBC028 (March 28, 2019)
Retrieved from: https://www.win.tue.nl/ hzantema/cpln.pdf

[4] Mouatadid, L., Introduction to Complexity Theory: 3-Colouring is NP-complete (summer 2014)
Retrieved from: http://cs.bme.hu/thalg/3sat-to-3col.pdf

Appendix 1

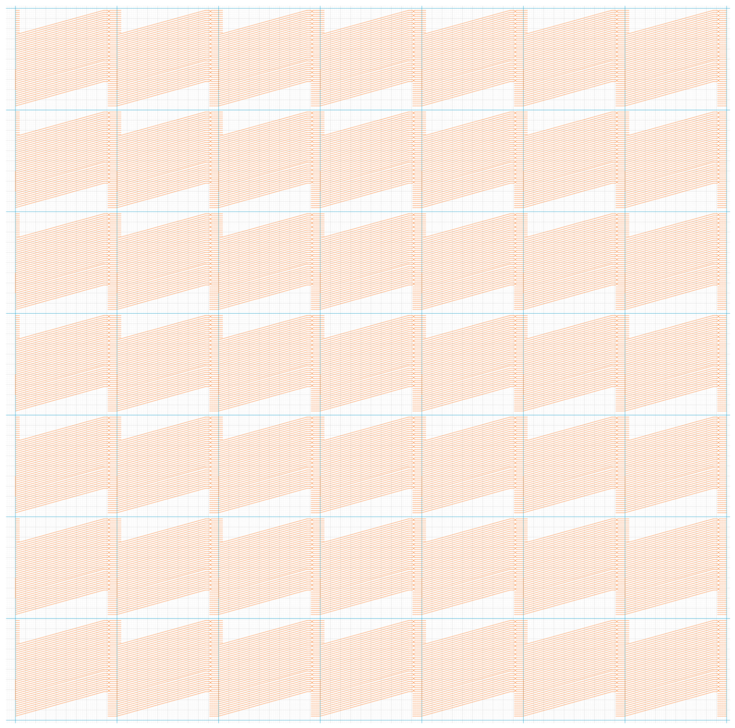


a. 49 tiles

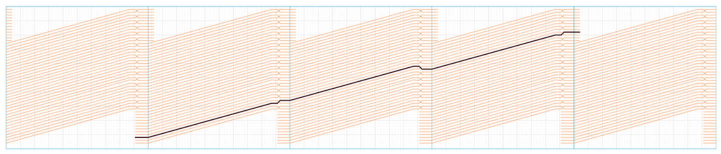


b. 1 wire

Figure 14: sp4-vertical

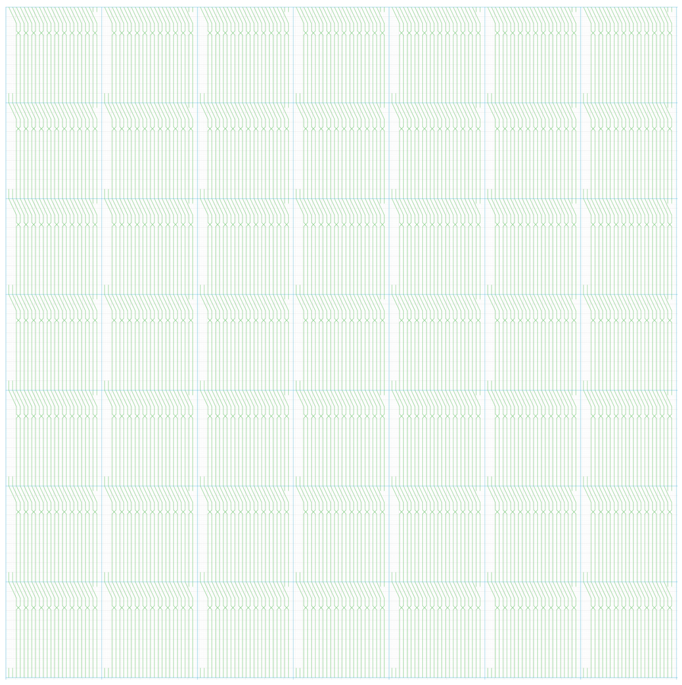


a. 49 tiles

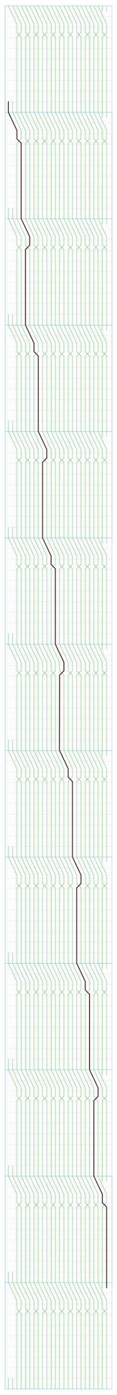


b. 1 wire

Figure 15: sp4-horizontal

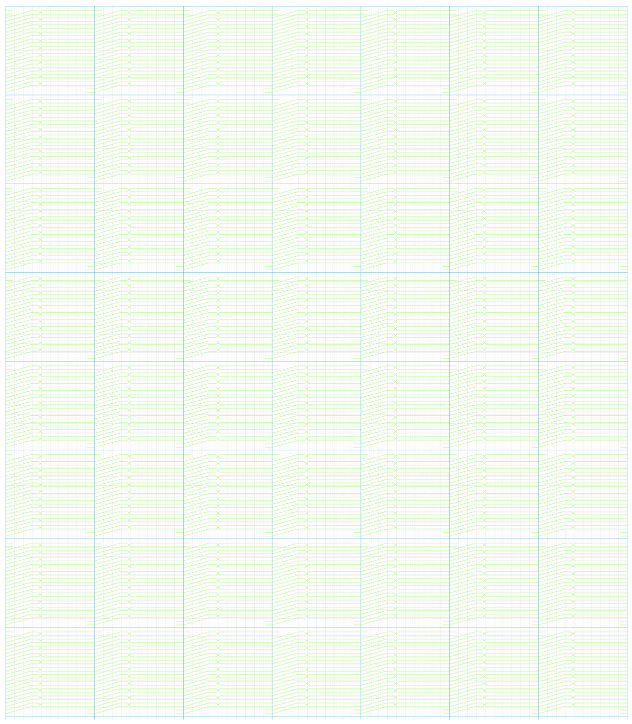


a. 49 tiles



b. 1 wire

Figure 16: sp12-vertical


a. 49 tiles

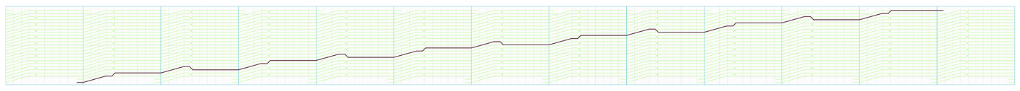

b. 1 wire

Figure 17: sp4-horizontal

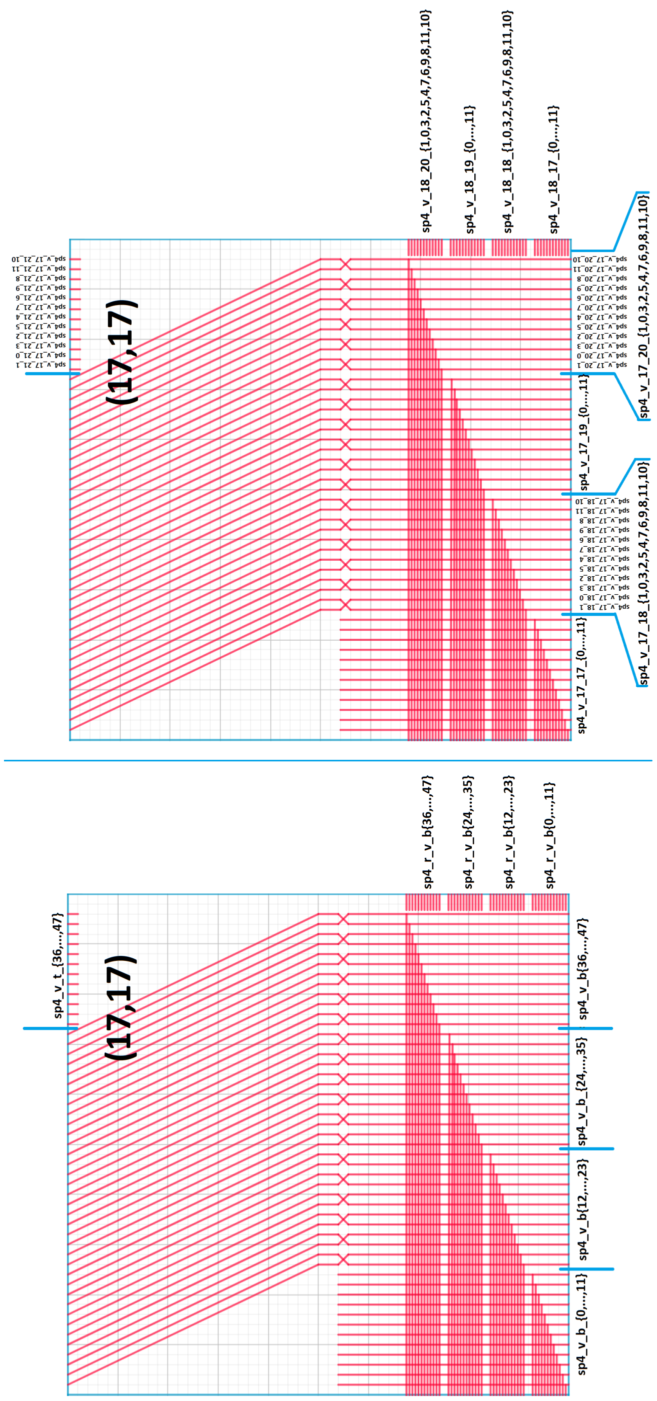


Figure 18: sp4-vertical, naming

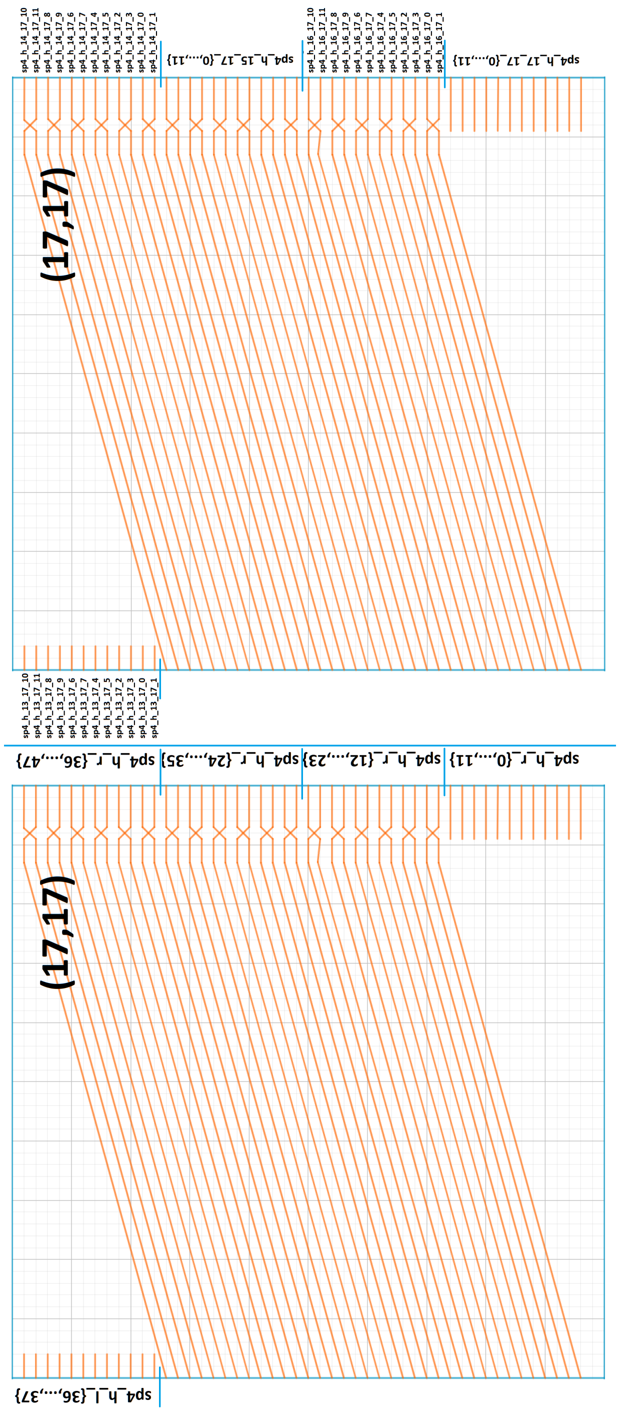


Figure 19: sp4-horizontal, naming