Synthesis of Sorting Networks using SAT Solvers

Andreas Morgenstern und Klaus Schneider

AG Eingebettete Systeme, TU Kaiserslautern

MBMV 2011

Motivation

Projekt Autosyn

- Automatische Synthese von Algorithmen aus Spezifikationen
- Werkzeuge: BDDs, SAT Solver, ...

Anwendungsbeispiel: Sortiernetzwerke

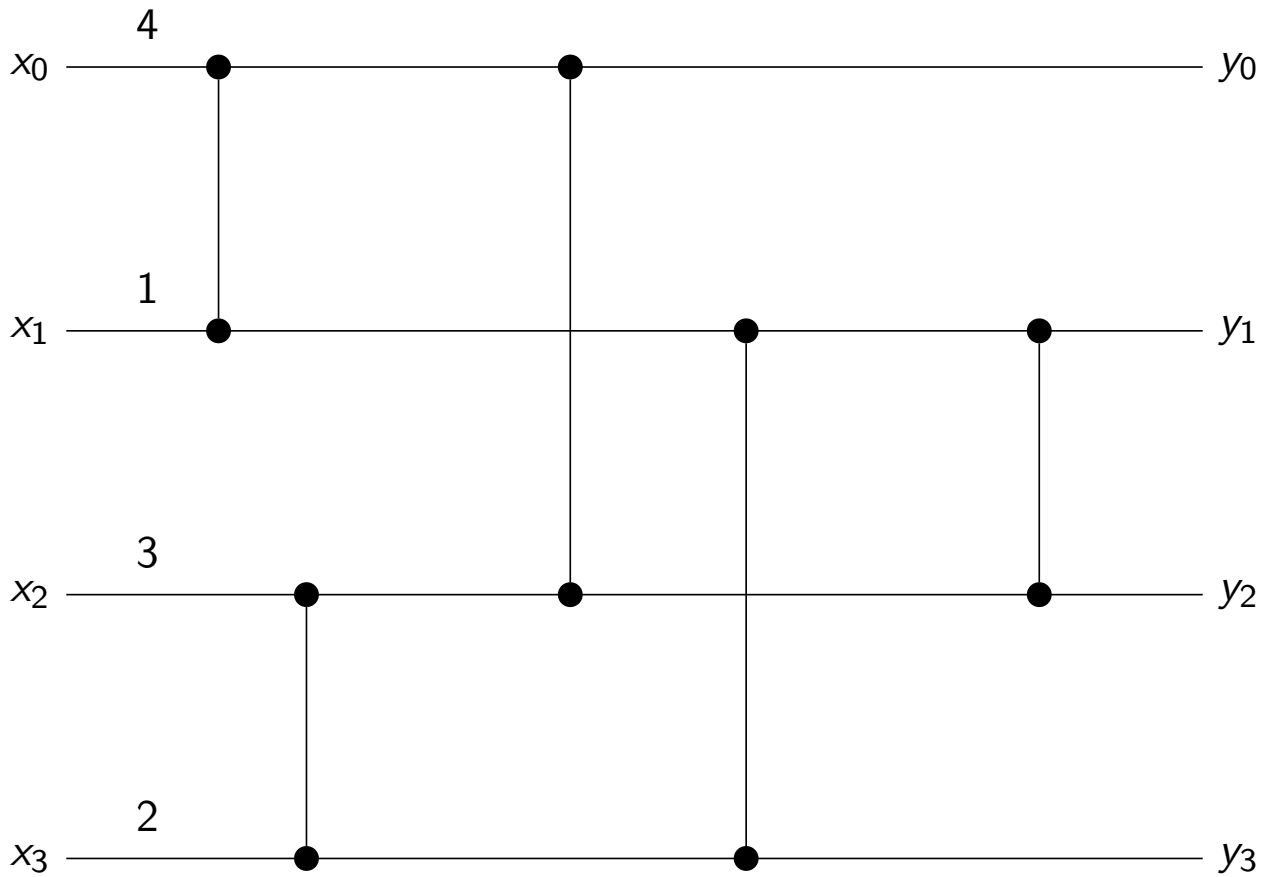
- Sortiernetzwerke repräsentieren Sortierverfahren
- Lassen sich diese automatisch mit SAT Solvern erzeugen ?

Averest Framework

- System für HW/SW Design
- Synchrone Sprache Quartz
- Model-Checking

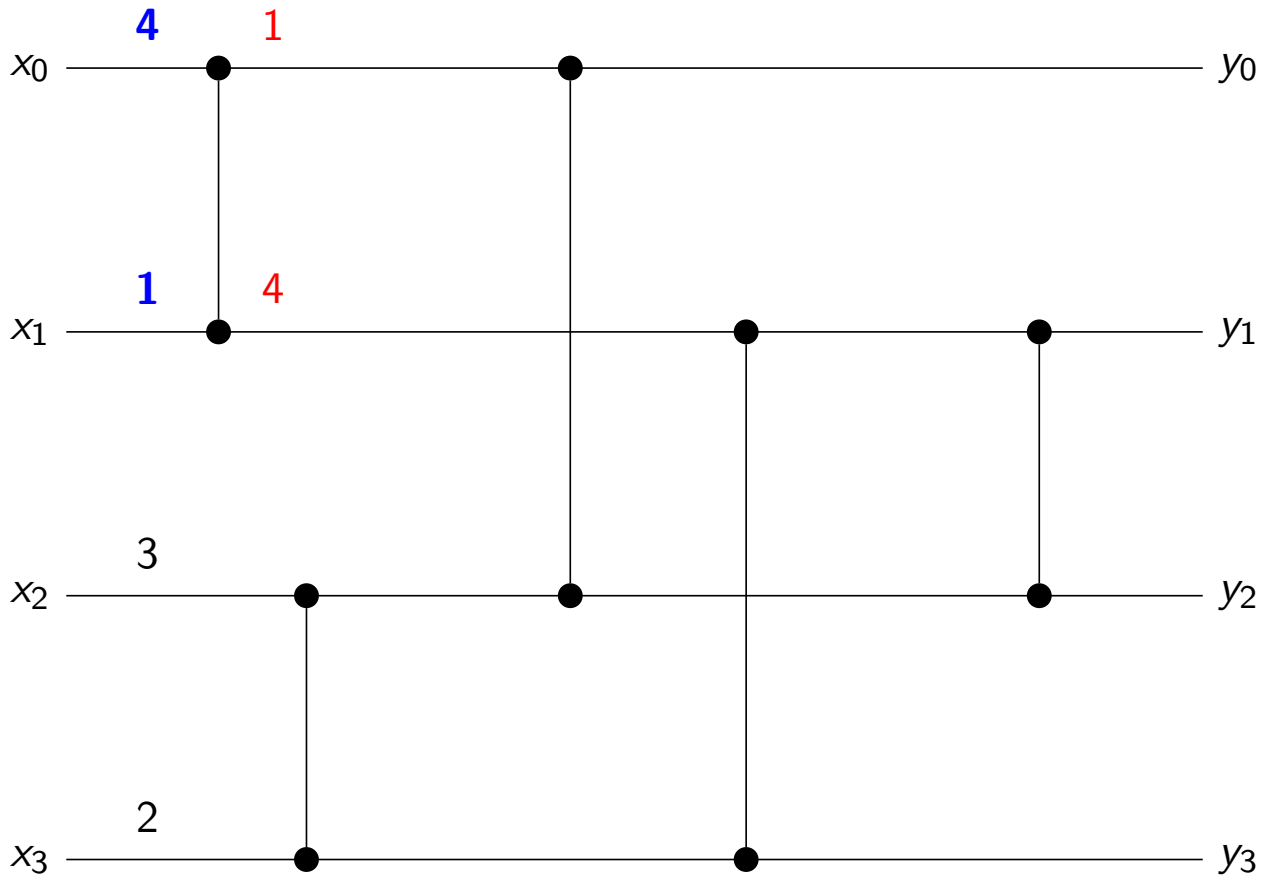
Übersicht

Grundlagen Sortiernetzwerke



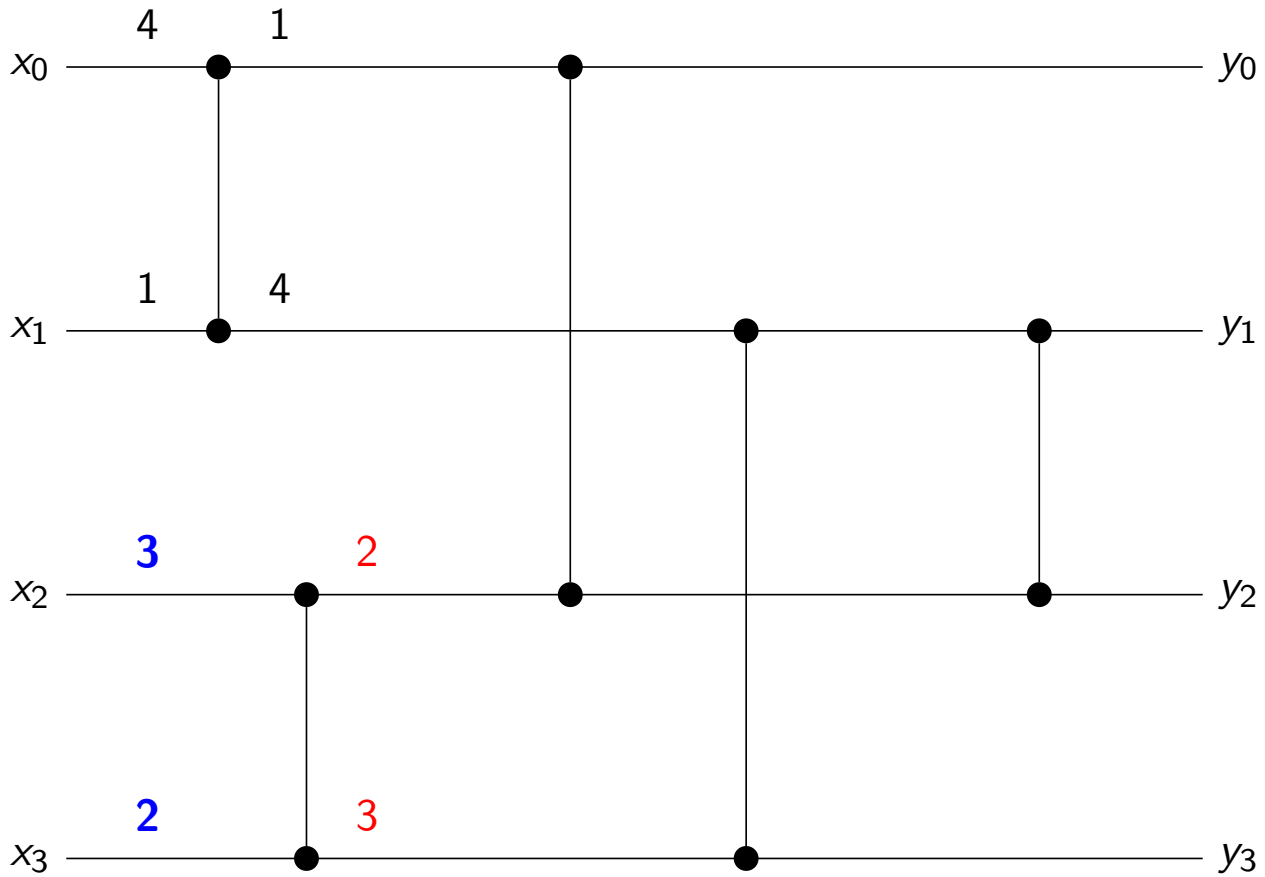
Vergleichsknoten : größerer Wert “sinkt“ nach unten

Grundlagen Sortiernetzwerke



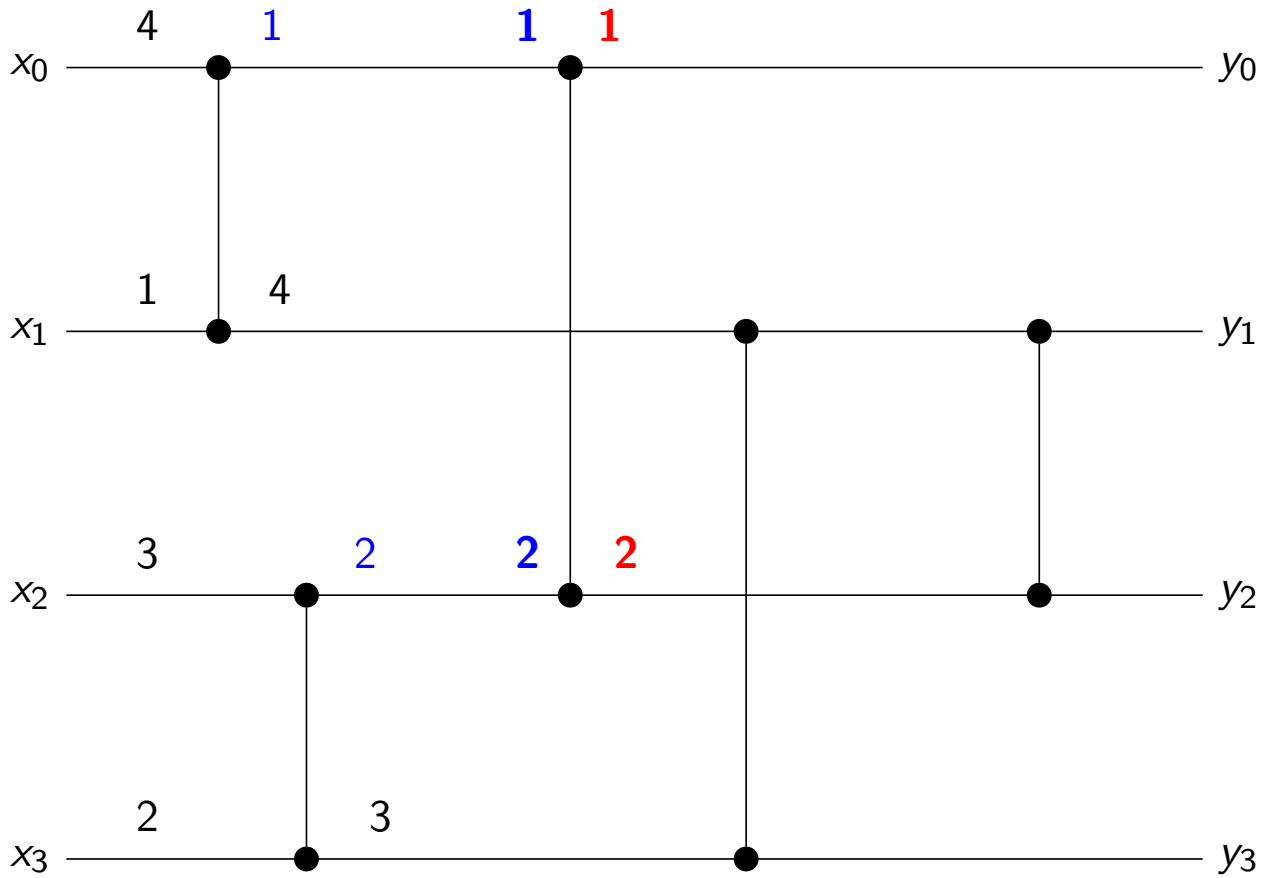
Vergleichsknoten : größerer Wert “sinkt“ nach unten

Grundlagen Sortiernetzwerke



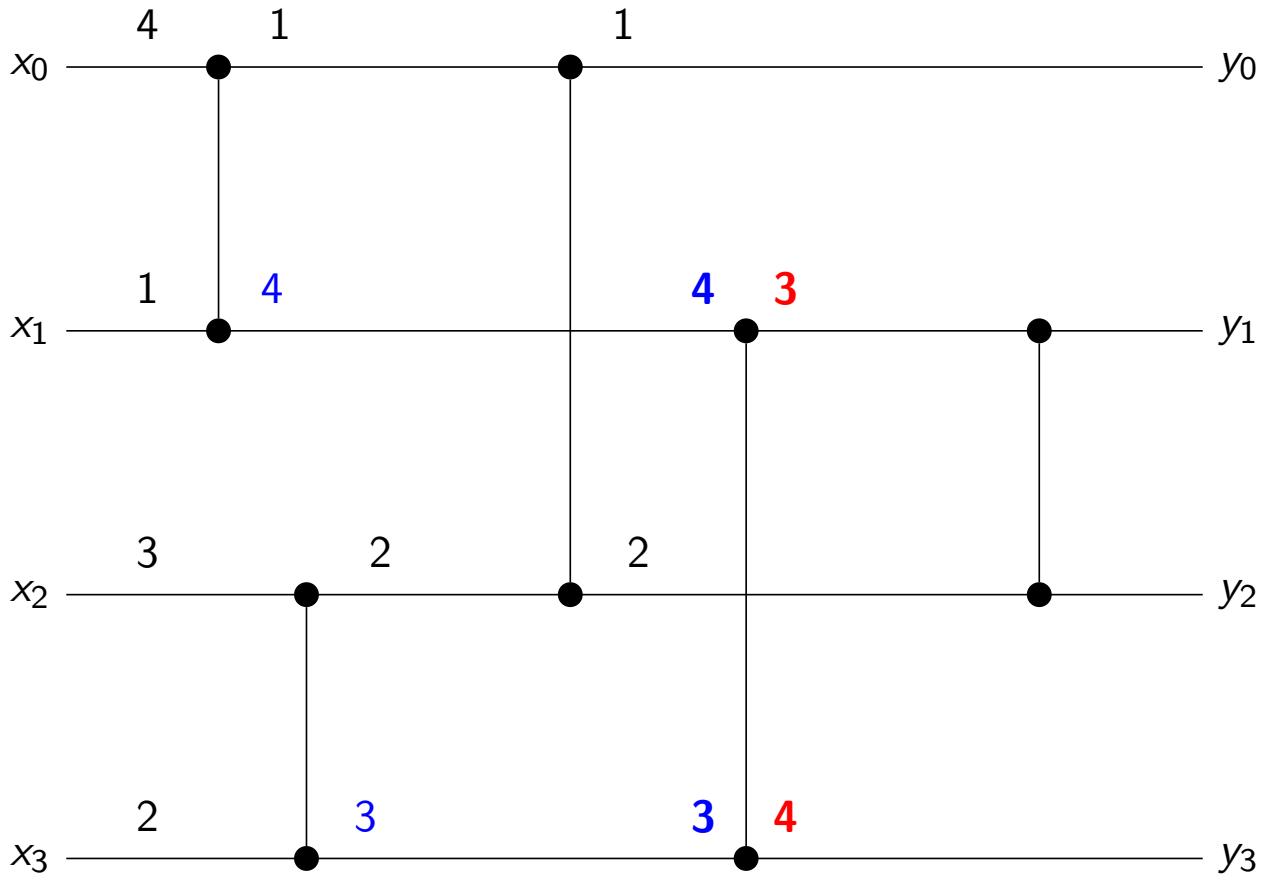
Vergleichsknoten : größerer Wert “sinkt“ nach unten

Grundlagen Sortiernetzwerke



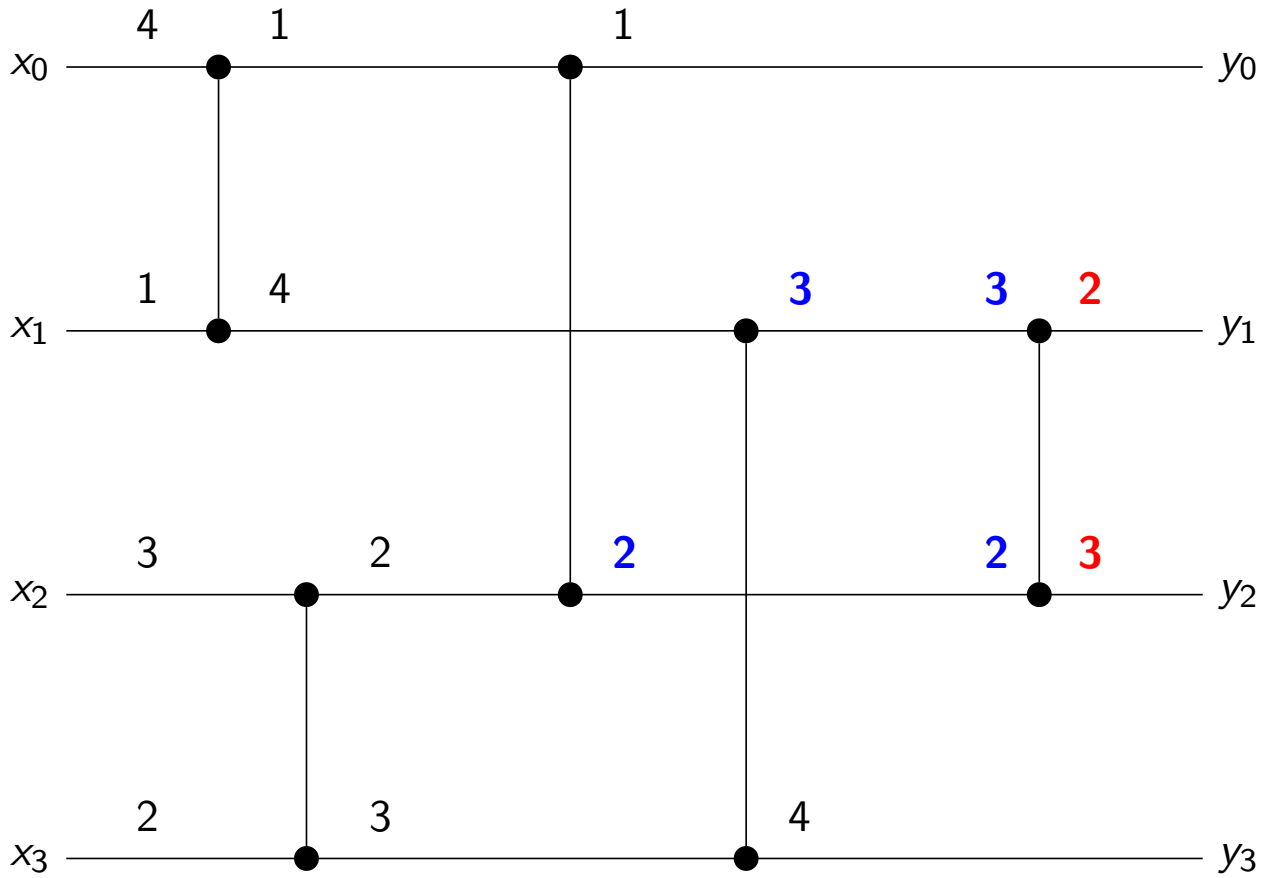
Vergleichsknoten : größerer Wert “sinkt“ nach unten

Grundlagen Sortiernetzwerke



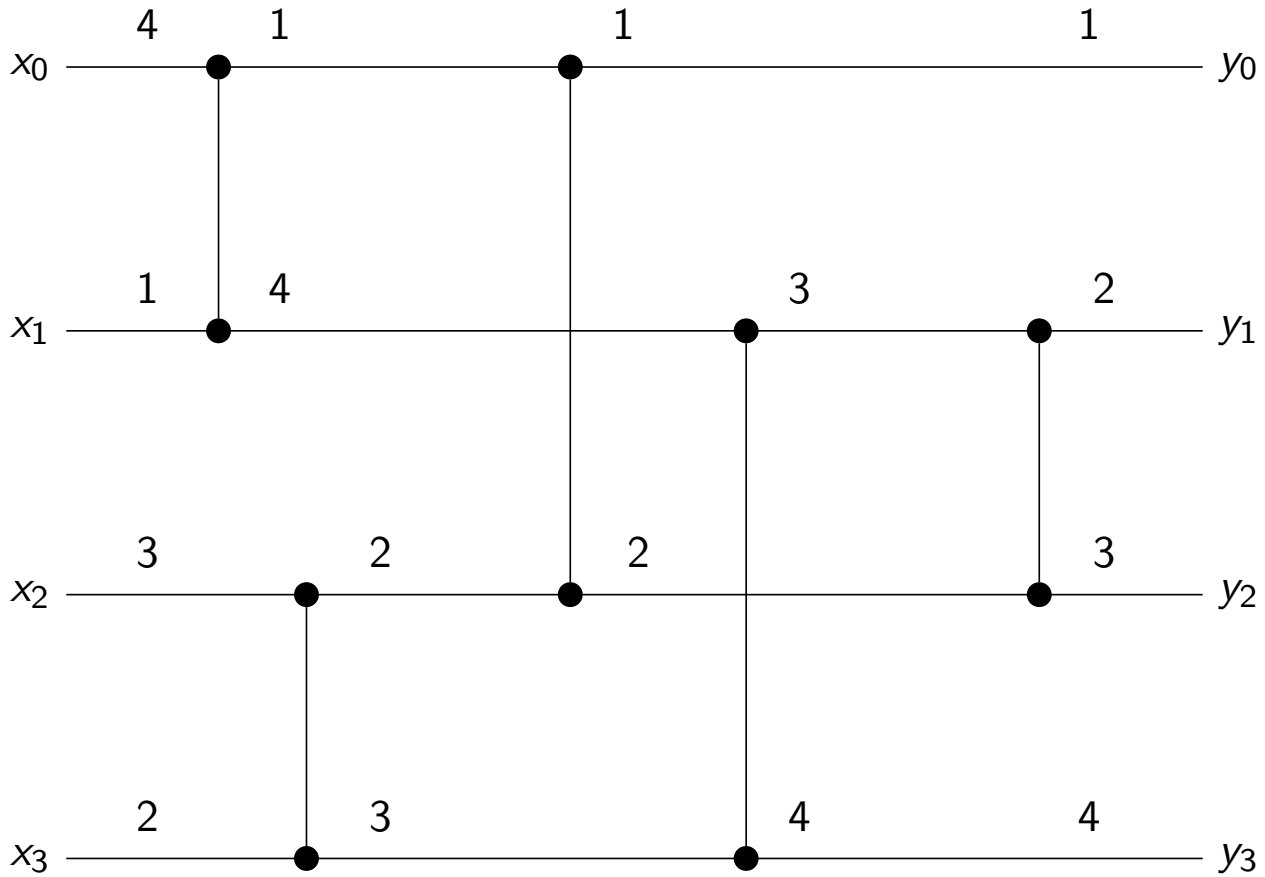
Vergleichsknoten : größerer Wert “sinkt“ nach unten

Grundlagen Sortiernetzwerke



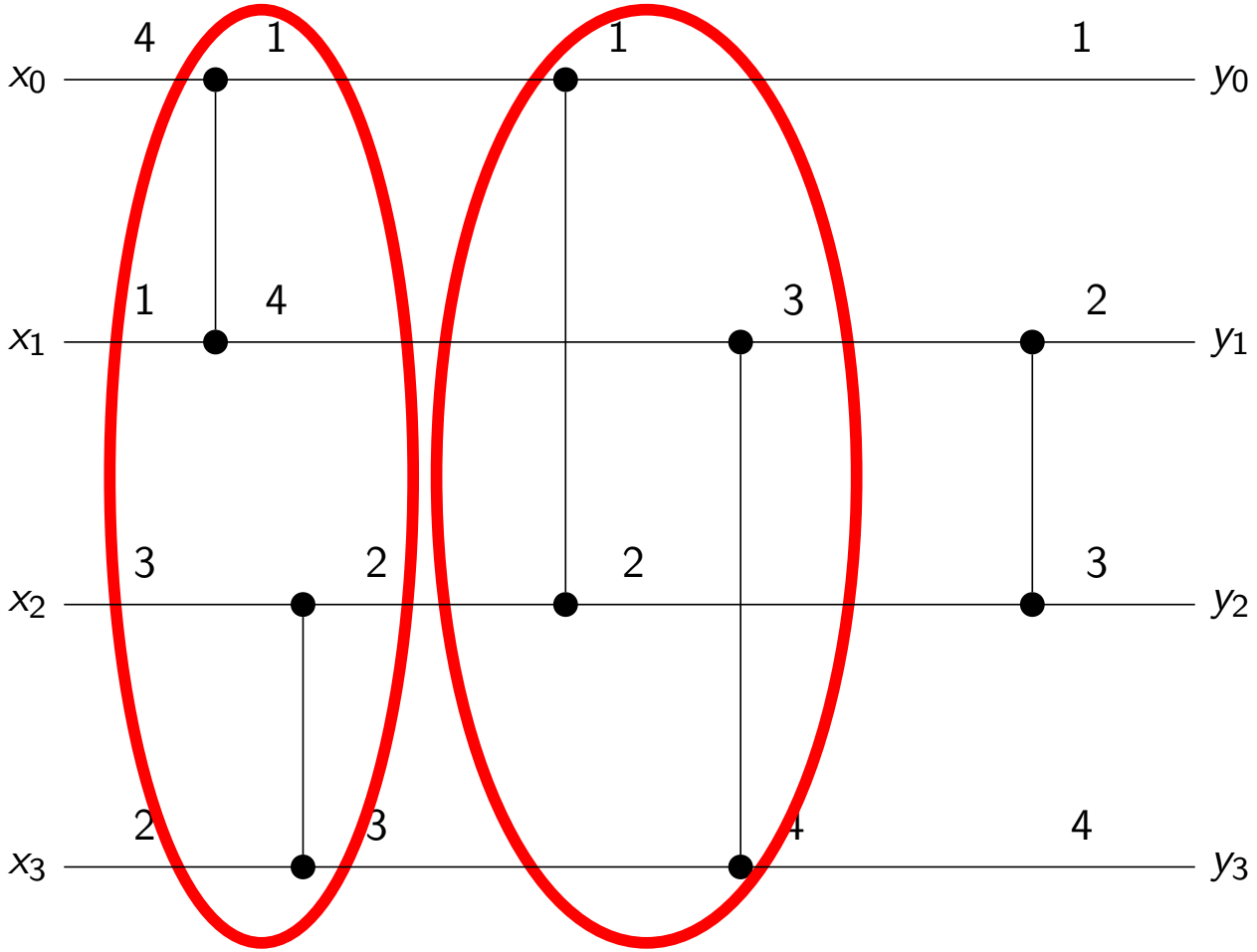
Vergleichsknoten : größerer Wert “sinkt“ nach unten

Grundlagen Sortiernetzwerke



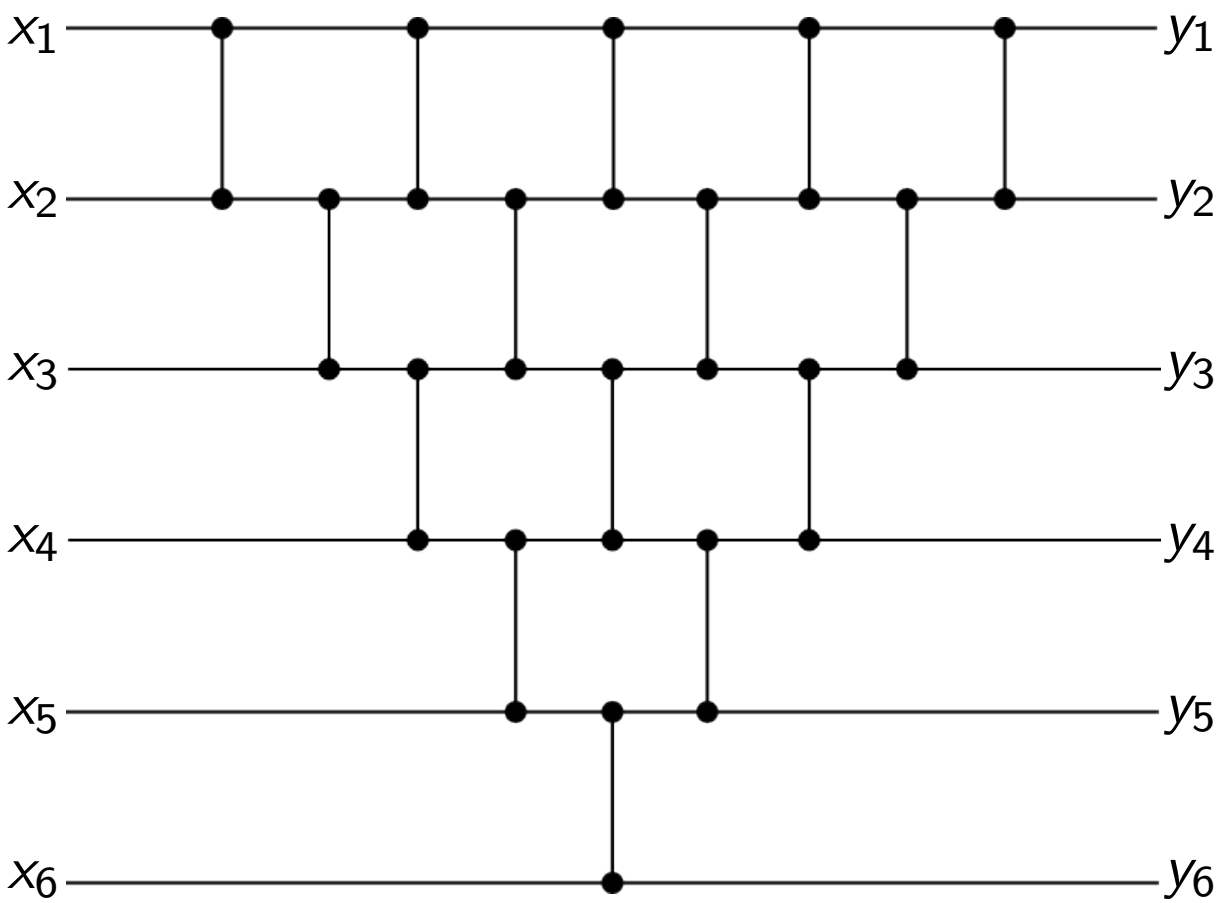
Vergleichsknoten : größerer Wert “sinkt“ nach unten

Grundlagen Sortiernetzwerke



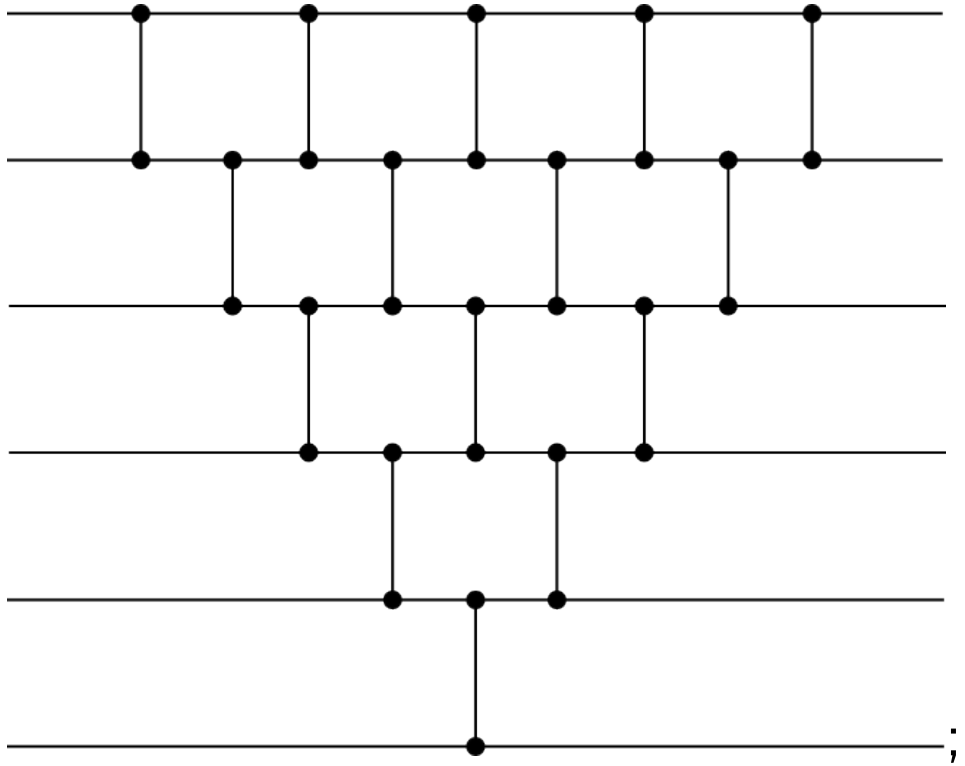
- **Parallele** Implementierung von Sortierverfahren
- Minimierungsziel: Tiefe (kritischer Pfad) und somit Laufzeit

Paralleles Bubblesort



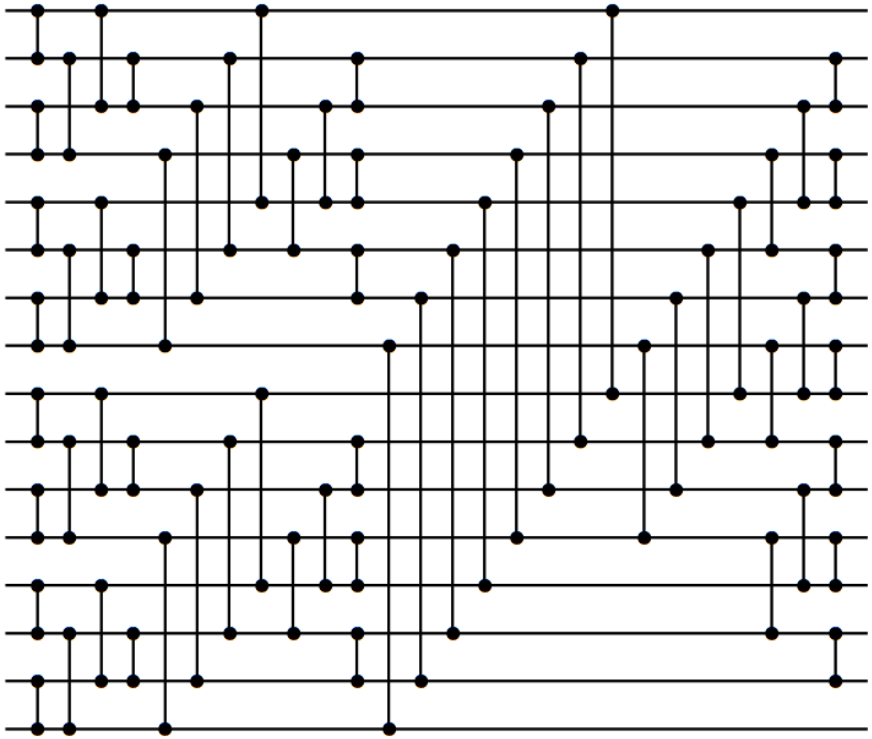
Übersicht

Geschichte der Sortiernetzwerke



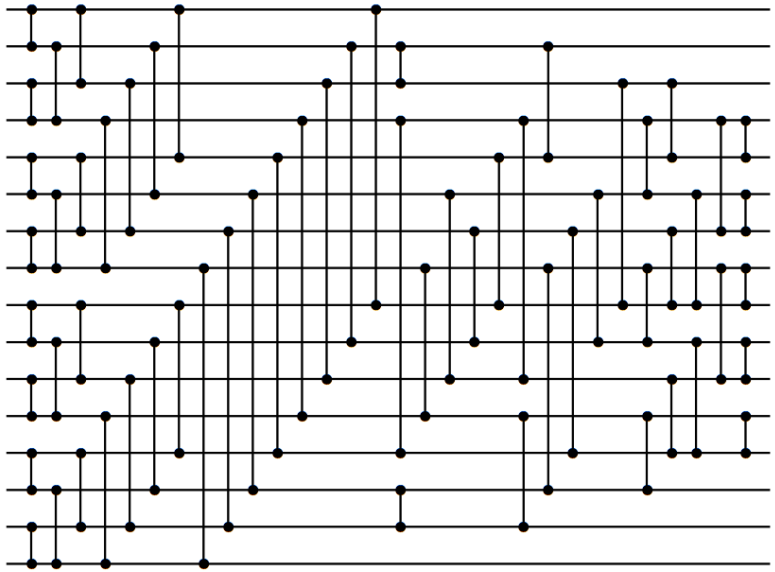
- Armstrong/ Nelson/ Connor Patent[1954]: Netzwerke $4 \ldots 8$
- Bose / Nelson 1962 : $O(n^{1.586})$ Knoten
- Floyd / Knuth 1964 : $O(n^{\left(1+\frac{c}{\sqrt{\log n}}\right)})$ Knoten

Geschichte der Sortiernetzwerke



- Batcher Odd-Even Merge 1964:
  $O(n(\log n)^2)$ Knoten, Tiefe $O((\log n)^2)$
- AKS-Netzwerk 1983: $O(n \log n)$ Knoten, Tiefe $O(\log n)$

Geschichte der Sortiernetzwerke



Spezialisierte Netzwerke für feste Größen

Co-Prozessor für bspw. Merge-Sort

- Green 1969: input 16, 60 Knoten, Tiefe 10
- Vorhois 1971: input 16, 61 Knoten, Tiefe 9

Computerunterstützte Suche

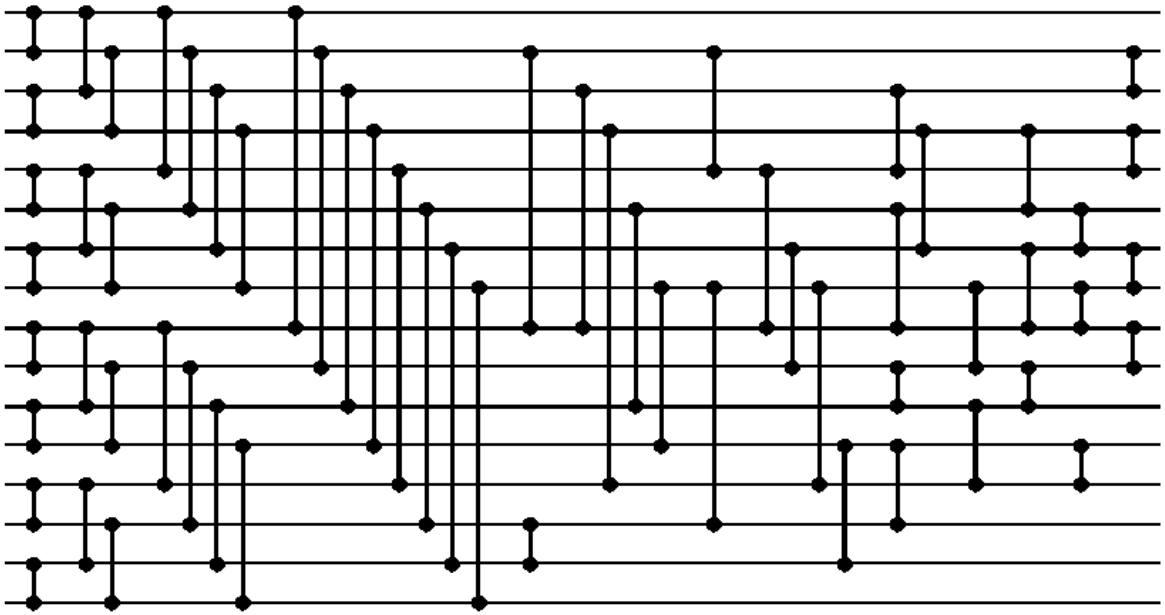


Abbildung: Hillis: input 16, 60 Vergleiche, Tiefe 11

- Parberry 1989: Symmetrien, ...
- Hillis 1992: Genetische Algorithmen
- Schwiebert 2001 : Genetische Algorithmen

Zusammenfassung Einleitung

Sortiernetzwerke: Repräsentieren parallele Sortieralgorithmen

Ziel: Sortiernetzwerke mit minimaler Laufzeit

Idee: SAT-Solver durchsuchen große Zustandsräumeräume

Frage: Ist die Suche nach optimalen Sortiernetzwerken mit SAT-Solvern möglich?

Obere(O) und Untere(U) Schranken für optimale Sortiernetzwerke

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
O	0	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9
U	0	1	3	3	5	5	6	6	7	7	7	7	7	7	7	7

Übersicht

Umsetzung

Grundlegende Idee

- Formulierung als Model-Checking Problem
- Nutze Averest-Compiler zur Erzeugung der SAT-Formel
- Löse mit SAT-Solver

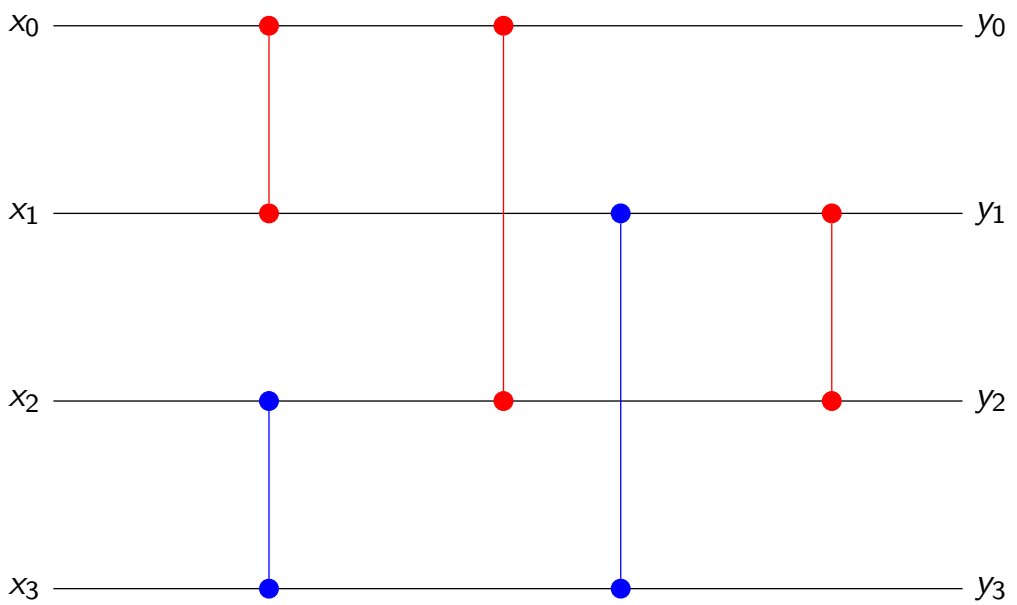
Idee Verfeinerung

- Sortiernetzwerk als Input
- Model Checker “errät“ korrekte Lösung (bestimmt inputs)
- Quartz-Programm überprüft, ob korrekt

Technik

- Model Checker kann nur Gegenbeispiel erzeugen
- Spezifikation also Negation

Sortiernetzwerke als Permutationsmatrix

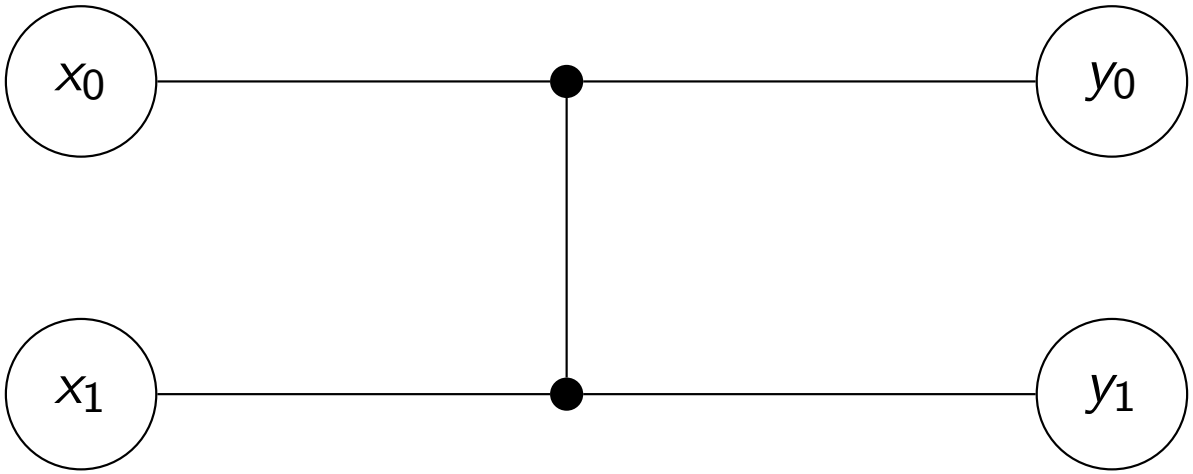


i	$P_{0,i}$	$P_{1,i}$	$P_{2,i}$
0	1	2	0
1	0	3	1
2	3	0	2
3	2	1	3

Matrix ist gültige Repräsentation , falls gilt :

$$\forall k.\ P_k, i = j \Rightarrow P_k, j = i$$

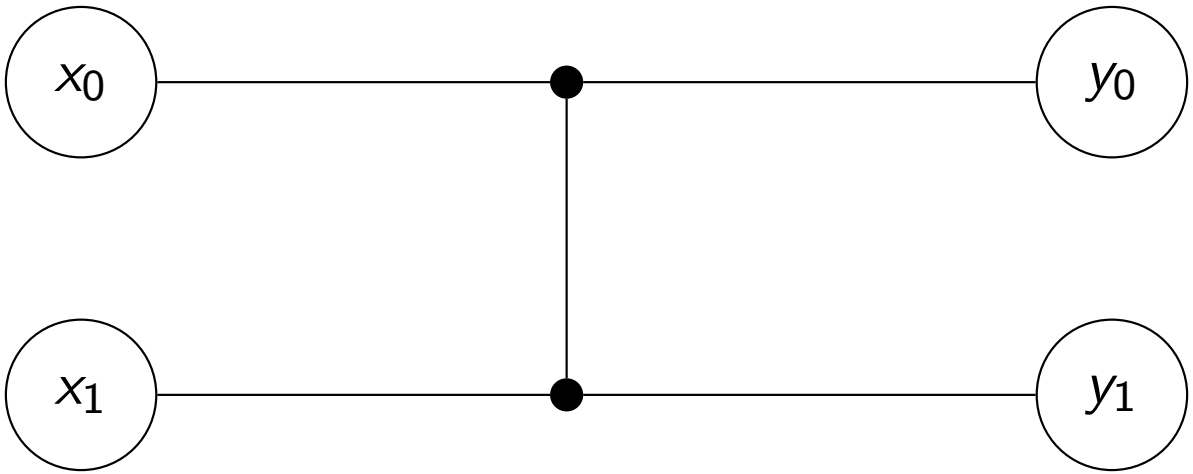
SAT-Kodierung



Theorem (Bouricius,Knuth)

Falls ein Netzwerk mit n Eingängen alle 2^n Sequenzen von 0 und 1 sortiert, so sortiert es beliebige Sequenzen von n Zahlen.

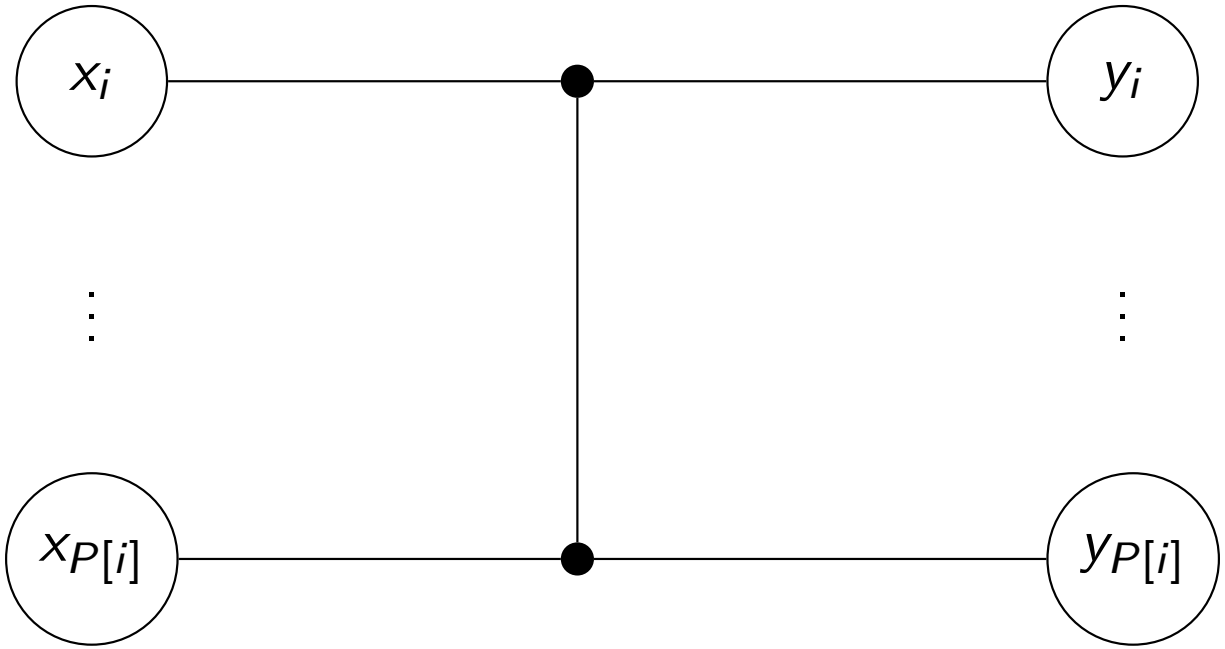
SAT-Kodierung



$$y_0 = \left\{ \begin{array}{l} x_0 \text{ if } x_0 \leq x_1 \\ x_1 \text{ else} \end{array} \right\} = x_0 \wedge x_1$$

$$y_1 = \left\{ \begin{array}{l} x_1 \text{ if } x_0 \leq x_1 \\ x_0 \text{ else} \end{array} \right\} = x_0 \vee x_1$$

SAT-Kodierung mit Permutationsmatrix



$$y_i = x_i \wedge x_{P[i]}$$
$$y_{P[i]} = x_i \vee x_{P[i]}$$

$$y_j = \begin{cases} y_j \wedge x_{P[j]} & \text{if } j < P[j] \\ y_j \vee x_{P[j]} & \text{else} \end{cases}$$

Quartz Program

```
module SortingNetwork([d][l]nat{l} ?P,bool noperm,unsorted) {
    for(x=0..exp(2,l)-1) {          // betrachte alle möglichen
        event [d+1][l]bool C;       // inputs x des Netzwerks
        for(i=0..l-1)
            C[0][i] = nat2bv(x,l){i};
        for(k=0..d-1)               // sortiere dieses input
            for(j=0..l-1)           // mit dem Netzwerk P
                if(P[k][P[k][j]]==j)
                    if(i<=P[k][j])
                        C[k+1][j] = C[k][j] & C[k][P[k][j]];
                    else
                        C[k+1][j] = C[k][j] | C[k][P[k][j]];
                else emit(noperm);
        if(!forall(i=0..l-2) (C[d][j] -> C[d][j+1]))
            emit(unsorted);         // teste, ob korrekt sortiert
    }
    assert(!noperm -> unsorted);
}
```

Gegenbeispiel : $\neg((\neg\text{noperm} \rightarrow \text{unsorted})) = (\neg\text{noperm} \wedge \neg\text{unsorted})$

Zusammenfassung

Workflow

- Übersetze Quartz-Program in SAT-Formel
- Nutze SAT-Engine des SMT-Solvers Yices
- Vergrößere Permutationsmatrix, um optimale Tiefe zu finden

Beispiel

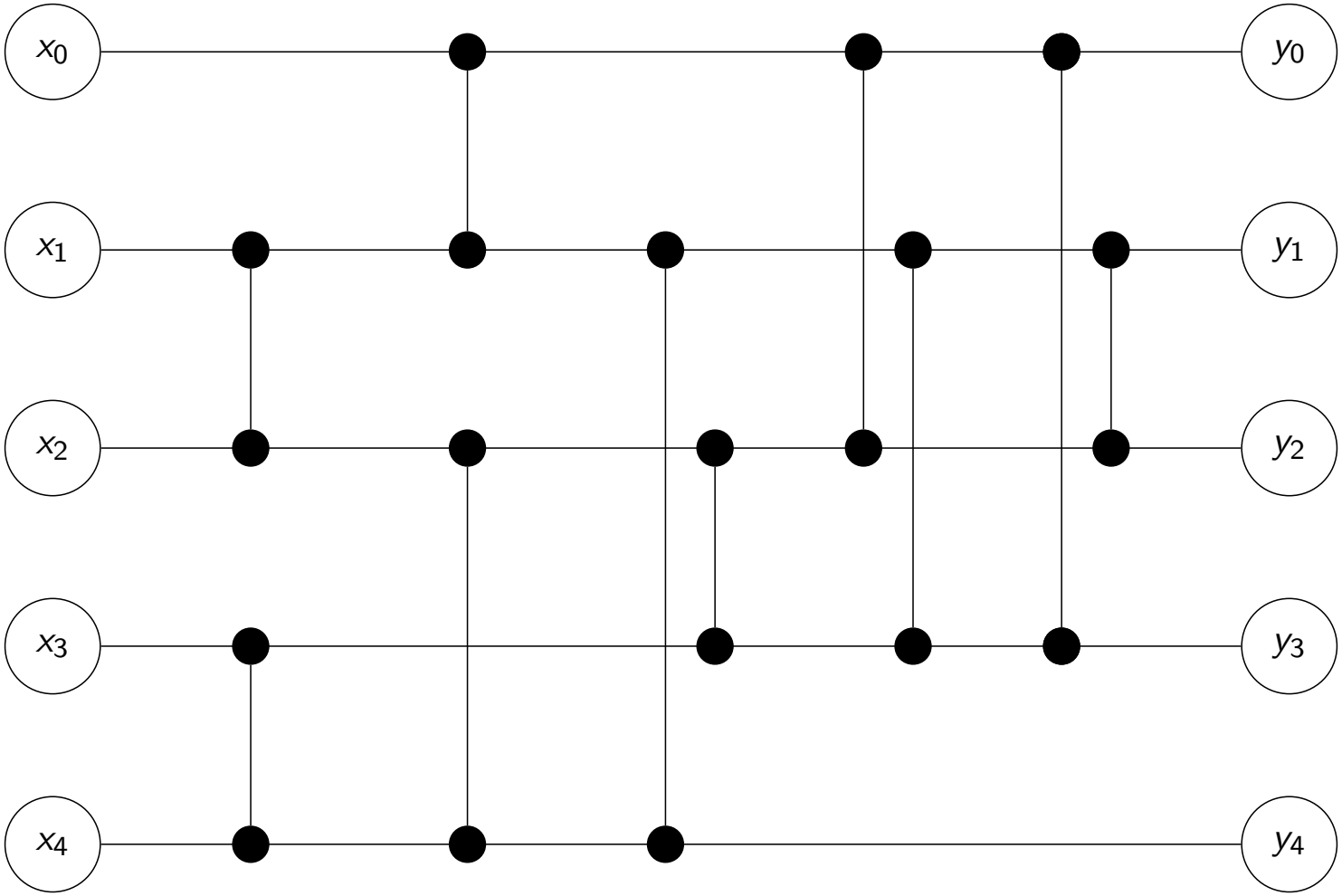


Abbildung: Synthetisiertes Netzwerk für n=5 und Tiefe 5

Zusammenfassung

Experimente

Länge	2	3	4	5	6	7	8	9	10
Tiefe	1	3	3	5	5	6	6	7	7
Zeit	$<$ 1s	$<$1s	$<$1s	1s	4s	1m20s	14m32s	15h50m	21d20h

Zusammenfassung

- Leider: Für n=11 : Laufzeit $>$ 2 Monate
- Reproduktion der bekannten Ergebnisse
- Dennoch : SAT-Solver können Algorithmen synthetisieren !

Ausblick

- Generiertes Netzwerk= Permutationsmatrix als CNF-Formel
- Wie erzeugt man daraus Programmausdrücke ?

Danke !