angel Documentation

Release v0.1

EPFL LSI

Apr 07, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

`angel` is a modern Quantm State Preparation (QSP) library implemented in C++-17. As it is modular and header-only, it can be easily integrated with other tools and often outperforms implementation developed in high-level programming languages such as Python.

The `angel` library implements algorithms for QSP with the purpose of synthesizing an optimized quantum circuit to prepare a given quantum state. As an objective function, the algorithms focus on minimizing the quantum circuit's depth and the number of elementary quantum gates. In particular, the algorithms reduce the number of `controlled-NOT` (CNOT) gates, which are in many experimental NISQ architectures relatively expensive when compared to other elementary quantum gates. Finding the optimum quantum circuit with the minimum number of elementary gates, however, is in practice for arbitrary quantum states intractable. The `angel` library provides several different heuristics, which allow its users to trade-off runtime for quality. A good quantum circuit realization can be obtained fast and, if a user is willing to invest more runtime, the proposed algorithms are often capable to achieve substantial gate reductions. In combination with the `tweedledum`, a C++-17 header-only library for quantum circuit synthesis, `angel` can generate quantum circuits in standard quantum circuit formats, such as `Quantum Assembly` (QASM) or `Quantum Instruction Language` (QUIL).

At the present state, the `angel` library implements mainly algorithms for preparing uniform quantum states, a special class of quantum states that can be represented with Boolean functions.

CHAPTER

TWO

INSTALLATION

angel is a header-only C++-17 library. Just add the include directory of angel to your include directories, and you can integrate angel into your source files using

```
#include <angel/angel.hpp>
```

2.1 Requirements

We tested building angel on Mac OS and Linux using Clang 6.0.0, GCC 7.3.0, and GCC 8.1.0. It also compiles on Windows using the C++ compiler in Visual Studio 2017.

If you experience that the system compiler does not suffice the requirements, you can manually pass a compiler to CMake using:

```
cmake -DCMAKE_CXX_COMPILER=/path/to/c++-compiler ..
```

2.2 Building tests

In order to run the tests, you need to init the submodules and enable tests in CMake:

```
mkdir build
cd build
cmake -DANGEL_TEST=ON ..
make
./test/run_tests
```

CHAPTER

THREE

CHANGE LOG

3.1 v1.0.0 (Not yet released)

- **Algorithms:**
  - Sift variable reordering method (#1)
  - QSP using decision diagrams (#2)
  - QSP using functional dependencies (#4)
- **I/O:**
  - algorithm to extract Boolean functions from reconvergence-driven cuts. (#5)
- **Others:**
  - algorithm for exact_esop_cover_from_divisors. (#8)
  - algorithm for extracting SOP covers. (#6)

CHAPTER

FOUR

UQS PREPARATION ALGORITHMS

No efficient algorithm is known for preparing arbitrary quantum states. In the worst case, all existing algorithms require an exponential number of elementary quantum gates and runtime in the number of qubits. *Uniform quantum states* (UQS) are a subclass of arbitrary quantum states, which are superpositions over a subset of basis states, where all amplitudes are either zero or have the same value. Although uniformity is a restriction on arbitrary quantum states, uniform quantum states frequently appear as the input states of important quantum algorithms and have many practical applications.

The central idea of UQSP is that each uniform quantum state can be characterized by a Boolean function, which allows us to draw from the rich fund of Boolean approaches for analyzing and synthesizing circuit implementations.

Theorem Each n-qubit uniform quantum state $|\phi_{f(x)}\rangle$ corresponds one-to-one to an n-variable Boolean function $f(x)$, such that

$$|\phi_{f(x)}\rangle = \frac{1}{|\mathrm{Min}(f)|^{-2}} \sum_{\hat{x} \in \mathrm{Min}(f)} |\hat{x}\rangle$$

holds, where $\mathrm{Min}(f)$ denotes the minterms of f.

This theorem states that it is possible to map uniform quantum states into Boolean functions, i.e., the column vector representation $|\phi_{f(x)}\rangle$ of a uniform quantum state can be expressed as the superposition of those basis states $|\hat{x}\rangle$ for which $f(\hat{x}) = 1$ normalized by the square-root of the number of minterms of f. Using Boolean functions, we proposed two algorithms based on `functional decomposition` and `functional dependency`.

4.1 Functional decomposition

Representing uniform quantum states as Boolean functions allows us to employ the Shannon decomposition to solve the state preparation problem recursively [MSRDeMicheli20]. Our algorithm iterates over the variables of the Boolean function, which correspond to qubits, and prepares them one by one, by computing the probability of being zero for the variable depending on previously prepared variables. This computational step requires to count the number of ones for each recursive co-factor of the Boolean function. The probability is then the number of ones of the current function divided by the number of ones of the negative co-factor. We have presented an implementation of this algorithm in [MSRDeMicheli20] using *Binary Decision Diagrams* (BDDs) [Bry86] as a representation of Boolean functions and dynamic programming. BDDs are particularly suitable for our purpose because counting and co-factoring can be very efficiently implemented as BDD operations [Bry86].

Header: `angel/quantum_state_preparation/qsp_bdd.hpp`

template<class **`Network`**>
void `angel::`**`qsp_bdd`**(*Network* *&network*, std::string *str*, qsp_bdd_statistics *&stats*, create_bdd_param *param* = {})
Quantum State Preparation using Decision Diagram

Template Parameters

- `Network`: the type of generated quantum circuit

Parameters

- `network`: the extracted quantum circuit for given quantum state
- `str`: include desired quantum state for preparation in tt or pla version
- `stats`: store all desired statistics of quantum state preparation process
- `param`: specify some parameters for qsp such as creating BDD from tt or pla

4.2 Functional dependency

The construction presented in the previous section is complete and allows us to generate a quantum circuit for every uniform quantum state. In several cases, however, the recursive decomposition can be avoided in favor of more optimized constructions if a functional dependency among the current and the previously prepared qubits is recognized [MRDeMicheli20]. Such functional dependencies have been developed in the context of logic synthesis. The identified functional dependencies for a qubit q_i can be utilized in three ways: (1) to reduce the number of control qubits if q_i depends only on a subset of the previously prepared qubits, (2) to reduce the number of elementary quantum gates if the functional dependency can be well expressed with the library of hardware supported quantum gates, and (3) to reduce the number of control lines for preparing other next qubits to be prepared. We have presented two approaches to identify functional dependencies in [MRDeMicheli20] and implemented them using truth tables-based algorithms: the first approach, pattern search, identifies dependencies among variables that have a fixed and predefined structure; the second approach, ESOP synthesis, uses a SAT-based synthesis algorithm [REdOSDeMicheli20] for *Exclusive-or Sum-Of-Product* (ESOP) forms with a modified cost function. Finding dependencies in form of ESOP expressions with an XOR with many fanins and ANDs with only few fanins are particularly useful because they are the most general dependency structure that allow us to reduce the number of elementary gates. Moreover, we make use of variable reordering to ensure that no beneficial dependency is overlooked.

Header: `angel/quantum_state_preparation/qsp_deps.hpp`

template<class **`Network`**, class **`DependencyAnalysisStrategy`**, class **`ReorderingStrategy`**>
`class qsp_deps`
Quantum State Preparation using Functional Dependency

Template Parameters

- `Network`: the type of generated quantum circuit
- `DependencyAnalysisStrategy`: specify dependency analysis strategy
- `ReorderingStrategy`: specify variable reordering strategy

CHAPTER

FIVE

REFERENCES

CHAPTER

SIX

INDICES AND TABLES

- genindex
- search

BIBLIOGRAPHY

[Bry86] RE Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[MRDeMicheli20] Fereshte Mozafari, Heinz Riener, and Giovanni De Micheli. Dependency analysis for preparing uniform quantum states. In *Under Review*. 2020.

[MSRDeMicheli20] Fereshte Mozafari, Mathias Soeken, Heinz Riener, and Giovanni De Micheli. Automatic uniform quantum state preparation using decision diagrams. In *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL), Miyazaki, Japan, Nov 9-11, 2020*, To Appear. 2020.

[REdOSDeMicheli20] Heinz Riener, Rüdiger Ehlers, Bruno de O. Schmitt, and Giovanni De Micheli. *Exact Synthesis of ESOP Forms*, pages 177–194. Springer, Cham, 2020. doi:10.1007/978-3-030-20323-8_8.

INDEX

A