Reversible Algorithms
Topics in Reversible Algorithms

- **Software and Programming Languages** [Glück et al '20]
  - Compilers and Interpreters [Yokoyama, Glück '07] [Yokoyama, Axelson, Glück '08]
  - Memory management [Cservenka, Haulund, Mogensen, Glück '18]
  - Object oriented [Axelson, Shultz '16] [Haulund, Mogensen, Glück '17] and functional programming [Yokoyama, Axelson, Glück '11] [Kawabe, Glück '03]

- **Algorithms and Complexity Theory**
  - Universal transformations for Turing Machines and Circuit Models [later slides]
  - Pebbling lower bounds [Li, Vitányi '92] [Li, Vitányi '96]
  - Oracle separation [Frank, Ammer '17]
  - Classification of reversible circuits [Aaronson, Grier, Schaeffer '15]

- **Efficient Reversible Algorithms** [later slides]
  - Small constant factor overhead in time and space complexity
  - Sorting Algorithms
  - Graph Algorithms
  - Linear Algebra
Universal Reversible Computing

- **History Recording** [Lecerf ‘63, Bennett ‘73]
  - Make functions bijective by storing inputs
  - Time: $T' (n) = O(T(n))$; Space: $S' (n) = O(S(n) T(n))$

- **Recursive Recomputing** [Bennett ‘79]
  - Compute to the midpoint, store it and uncompute to the last checkpoint.
  - Recurse.
  - Time: $T' (n) = O(T(n) \lg(T(n)))$; Space: $S' (n) = O(S(n) \lg(T(n)))$

- **Configuration Space Enumeration** [Lange, McKenzie, Tapp ‘00]
  - Walk the entire computation tree. Reminiscent of Savitch’s Algorithm.
  - Time: $T' (n) = O(2^{1T(n)})$; Space: $S' (n) = O(S(n))$

- **Time-Space Tradeoff** [Williams ‘00][Buhrman, Tromp, Vitanyi ‘01]
  - Embed Configuration Space Enumeration at the bottom of a Bennett recursion.
  - Time: $T' (n) = O(S(n)k2^{1kT(n)/2^{1k} })$; Space: $S' (n) = O(kS(n))$

- **Computing with Dirty Ancilla Bits** [Xu ‘15]
  - Space can be temporarily used without knowing it’s prior state.
  - Time: $T' (n) = O(2^{\lg T(n)})$; Space: $S' (n) = S(n)+1$
Energy, Entropy, and Conditional Reversibility

- Tradeoff between space-usage and bit-erasure as a resource [Li, Vitányi ‘92]
- Define word RAM and transdichotomous RAM models with energy cost based on function injectivity [Demaine, Lynch, Mirano, Tyagi ’16]
- Formally defines conditional reversibility, allowing reversibility on input-restricted domains [Frank ‘17]
- Analysis of conditional reversibility in a machine learning test case. [DeBenedictis, Frank, Anderson ‘16]
What is Efficient?

- Asymptotically equivalent time and space usage
- Small constant factors in overhead
- Axelsen and Yokoyama define the following:
  - \( g(n) \) faithful simulation has time \( T'(n) = \Theta(T(n)) \) and space \( S'(n) = O(S(n) + g(n)) \)
  - A hygienic simulation is \( g(n) \) faithful for minimum possible \( g(n) \).
Efficient Reversible Algorithms

- **Sorting Algorithms** [Axelsen, Yokoyama ’15] [Masuda, Yokoyama ’19]
  - Constant factor overhead for several algorithms
  - Quadratic algorithms which preserves best case running time with additive space overhead
- **Graph Algorithms** [Frank ’99] [Nøhr ’15] [Guo, Peng, He ’18]
  - Shortest Path and APSP
  - Minimum Spanning Tree
- **Data Structures** [Yokoyama, Axelsen, Glück ’08] [Axelsen, Glück ’13] [Nøhr ’15] [Demaine, Lynch, Mirano, Tyagi ’16]
  - Adjacency List
  - Binary Search Tree
  - Dynamic Array
  - Disjoint Set
  - Min-priority Queue
- **FFT** [Yokoyama, Axelsen, Glück ’08]
- **Matrix Multiply** [Frank ’99] [Demaine, Lynch, Mirano, Tyagi ’16]
Techniques for Efficient Reversible Algorithms

- Input logging
- Reversible sub-routines
  - Input can be calculated from the output
  - Call and Uncall the routine, saving space when outside the function
- Paired branches and protected conditionals
  - Control flow must know where to return
  - Conditionals unedited inside a loop are easier to maintain
- Pointer swapping
  - Always maintain back pointers
  - Instead of destroying a pointer, change where it is stored
- Permutation representations
  - Permutations are bijective
Special Purpose Reversible Computing

- Accelerators and heterogeneous architectures
  - Hardware designed to be extremely performant on certain classes of algorithms.
  - GPU, TPU, ASIC, etc.
  - Restricted computing classes could make reversible architecture design easier.

- Embedded/ubiquitous computing and extreme environments
  - Often requires extreme energy efficiency.
  - Often only needs special purpose computation.

- High-performance computing
  - Massively parallelizable algorithms, again a restricted class.
  - Willing to invest in hardware to get performance.

- Quantum computing
  - Requires substantial classical computation to control the quantum computation.
  - The classical computing must interface with quantum circuits.
  - Already uses adiabatic hardware, ex. superconducting circuits.
Special Purpose Reversible Computing

- Special classes of algorithms
- Requires low-energy / high-performance computing
- Higher investment costs acceptable

- Requires study and optimization of specific classes of algorithms
- Software/Hardware co-design
- Provides targets and stepping stones for both algorithmic and hardware advances
Suggested Targets

- Optimization algorithms
- Machine Learning
- Differential equation solvers
- Computational Geometry
  - Triangulation
  - Point/range query
  - Ray/polygon intersection
- Perfect matchings
- Edit distance and other string comparison
Bibliography