Two Data Tiling Problems Based On Graphs

Indranil Banerjee

Louisiana State University

ibanerjee@lsu.edu

September 20, 2018
Graphs and Hypergraphs

- A Hypergraph $G(V, E)$ is a set of vertices $V$ along with a collection $E$ of non-empty subsets of $V$.
- We call a subset $e \in E$ an edge.
- A graph is a special hypergraph where edges have cardinality $\leq 2$.
- Since we are going to talk about computations on graphs we assume all graphs discussed here are finite.
- We take $|V| = n$ and $\sum_{e \in E} |e| = p$ and $|E| = m$.
- For graphs $p = 2m$. 
Graphs and Hypergraphs

Figure: A Hypergraph
Assumptions

- All (hyper)graphs are undirected and connected.
- There may be a weight function associated with vertices or edges of the graph.
- In the former case we call it vertex weighted and in the latter we call it edge weighted.
- A graph may be both vertex and edge weighted and an weight of an edge may depend on the weight of the vertices incident to that edge. (we use this for tiling)
- If the weights are integers from the domain \([k]\) for some \(k \in \mathbb{N}\) then we can identify them with colors.
Graph and Hypergraph Coloring

- A \( k \)-\( (\text{vertex}) \) coloring of a graph is a function \( C : V \rightarrow [k] \) that assigns one of the \( k \) colors to each vertex such that no pair of vertices incident to an edge have the same color.
- For hypergraphs there are more than one way to define a coloring.
  - \textit{Strong Coloring}: All vertices sharing an edge have unique colors.
  - \textit{Weak Coloring}: For every edge there exists a pair of vertices having differing colors.
- For the purpose of posing tiling problems in terms of hypergraphs we will use the latter definition.
Constraints Satisfaction Problems (CSPs)

- General form: 1) A set of variables $X$ over integer domain $[d]$.  
  2) A fixed collection of functions $\mathcal{F}$ of the form $[d]^k \rightarrow \{0, 1\}$.  
  3) An instance is specified by a set of constraints $C_i = (f_i, X_i)$ where $f_i \in \mathcal{F}$ and $X_i \subseteq X$.  

- Boolean CSP: The domain is $\{0, 1\}$ and functions in $\mathcal{F}$ are boolean functions.  

- Example: SAT where $\mathcal{F} = \{OR\}$
NP-Hardness and Approximation

- All optimization problems we are going to discuss here are at least \( NP \)-hard.
- They do not have any deterministic polynomial (\( O(n^c) \) for some constant \( c \)) time exact algorithms unless \( P = NP \).
- So we turn our attention to approximation algorithms.
Approximation Ratio

- Let $P$ be some $NP$-hard (minimization) optimization problem
- Let $A$ be some algorithm that approximates $P$
- Approximation ratio $\alpha = \frac{\text{output of } A}{\text{optimal value of } P}$
- $\alpha \geq 1$
- $A$ must have a polynomially bounded runtime (i.e., $O(n^c)$ for some constant $c$)
- $A$ may be randomized. For many cases a randomized algorithm can be derandomized.
Hardness of Approximation

- $P$ has a FPTAS if $\alpha$ can be made arbitrarily close to 1.
- If $\alpha \geq \alpha^*$ then $P$ is $\alpha^*$-approximation hard.

Figure: Approximation Hierarchy
Tiling

Figure: A simple example of tiling 3 matrices among 2 nodes. Here A,C are tiled column-wise and B is tiled row-wise.
Tiling

- \( P = \{p_1, p_2, \ldots, p_k\} \) be a set of \( k \) nodes of a distributed computing network.
- Let \( A \) be some \( d \)-dimensional array of size \( n_1 \times n_2 \times \ldots \times n_d \).
- A tiling of \( A \) is a function \( t_A : \prod_i [n_i] \to P \).
- Assume \( A (|A| = n) \) is 1-dimensional then \( t_A : [n] \to P \).
- If copying is allowed then \( t_A : [n] \to 2^P \).
- Partial tiling with copying: \( t_A : [n] \to 2^{P'}, P' \subseteq P \).
Tiling Cost

- There are many ways we can define this.
- Depends on what problem we want to solve.
- Basic idea: if \( t_A(i) = t_B(j) \) for two arrays \( A, B \) then we assume an atomic operation on \( A[i], B[j] \) is "free".
- Otherwise, there is a cost involved with the operation.
Assumptions

- Tiling cost = cost of communication given a specific tiling configuration.
- We do not consider the cost of creating a tiled array.
- Reading/Allocating an array and distributing it among different processors is ignored.
- Since
  1) These operations are (mostly) unavoidable.
  2) Occurs infrequently (most cases at the beginning of the execution)
  3) It is reasonable to assume the cost of tiling an array is invariant to the choice of tiling.
Problem 1: When Tiling Choices are Fixed

- **Input:**
  1. A set $\mathcal{A}$ of arrays that need to be tiled. ($|\mathcal{A}| = n$)
  2. A set fixed set of tiling functions $T$ ($|T| = d$).
  3. A set of matrix/array operations $OP = \{op_1, \ldots, op_k\}$
  4. A set of constraints $f : OP \times 2^\mathcal{A} \rightarrow \mathbb{R}$ ($|\{f\}| = m$)

- Find an assignment (decide a tiling for each array) that minimize the total cost of the constraints.

- Why not formulate this as a maximization problem?
Problem 1: An Example

- Let $\mathcal{A} = \{A, B, C\}$.
- A “straight line” program = $(\text{op}_1(A, B), \text{op}_2(A, B, C), \text{op}_3(A), \text{op}_1(B, C))$.
- Total cost = 
  \[f(t_A, t_B, \text{op}_1) + f(t_A, t_B, t_C, \text{op}_2) + f(t_A, \text{op}_3) + f(t_B, t_D, \text{op}_1)\]
- This model does not consider loops, but this can be fixed by associating weights to the terms above.
- Recursion and branching can be modeled with non-linear composition of the cost functions.
- Even for the straight line case this is already hard to solve.
Problem 1: Unary Operations

- Consider operations of the following type: \( A = \text{transpose}(B) \).
- If \( t_A = t_B \) then there is non-zero communication cost.
- However for \( A = B \), only \( t_A = t_B \) ensures no communication is necessary.
Problem 1: Simplifying Assumption

- We assume there are two types of operators.
- For the first type (AE) the cost of communication is 0 if all operands have the same tiling. Otherwise, it costs 1 unit.
- For the second type (NAE), the cost is 0 iff there is a pair of operands that have differing tilings. Otherwise it is 1.
- We can model this as a min-cut problem on (hyper) graphs.
- We call it TP1($t, k$), where $k$ is the largest arity of any operation and $t$ is the number of different tiling choices.
Problem 1: A Special Case

- If we further assume that all operations involve at most 2 operands then this problem is equivalent to the min-2-LIN or min-UnCut problem.

- **min-2-LIN(2):**
  Input: A set of boolean constraints of the form $x_i \oplus x_j = 1$ or $x_i \oplus x_j = 0$ over a set of variables $X$.

- Find an assignment of $X$ that minimizes the number of unsatisfied constraints.
Problem 1: A Special Case

- This can also be viewed as a graph problem.

**Figure:** We want to minimize the total \# of solid edges crossing a cut and dashed edges that are non-crossing.
Problem 1: A Special Case

- 2-LIN(2) is “equivalent” to TP1(2, 2)
- Hence $TP1(2, 2)$ has a $\sqrt{\log n}$-approximation algorithm (w.h.p).
- It is also known that there is some constant $\alpha$ such that $TP1(2, 2)$ cannot be approximated with $\alpha$ (*).

(*) conditional.

- We want to extend this to $TP1(2, k)$ with $k > 2$.
- It is known either $TP1(2, k)$ has a $\sqrt{\log n}$-approximation algorithm or it is $2^{(\log^{1-\epsilon} n)}$-approximation hard.
Problem 1: Maximization/Conditional Problems

- Two variants:
  1) Maximization: Find an assignment that maximizes the number of satisfied constraints.
  2) Given an $1 - \epsilon$ satisfying instance of the problem find an assignment that satisfy at least $1 - f(\epsilon)$ fraction of the constraints.

Maximization and in some cases (2) are generally considered easy to solve.

For example: Every boolean CSP has a constant factor deterministic approximation algorithm.

Why work with minimization?
Problem 2

... and now for something completely different

Let $\mathcal{A} = \{A_1, \ldots, A_k\}$ be a set of arrays.

Suppose $DP$ is a (deterministic) distributed program (algorithm) that computes some result using the arrays in $\mathcal{A}$.

$DP$ determines a sequence $S$ of collection of elements from a subset of arrays in $\mathcal{A}$

Example: Suppose $\mathcal{A} = \{A_{2\times2}, B_{2\times2}, C_{2\times2}\}$. Let $DP$ compute $C = AB$ using say the Strassen’s matrix multiplication.

$S = ((A_{11}, A_{22}, B_{11}, B_{22}), (A_{21}, A_{22}, B_{11}), (A_{11}, B_{12}, B_{22}), \ldots)$
Problem 2

From $A, S$ we can create a multi-graph as follows.

- for each array $A$ create a vertex for corresponding to every element.
- For each tuple $s \in S$ create a clique using the vertices corresponding to the elements in the tuple.
- Let $G_{A,S}$ be this graph.
Problem 2

- Assume we have $P$ nodes ($|P| = k$) and node $i$ has memory budget of $m_i$.
- If two elements participating in an expression comes from the same node then there is no cost of communication.
- We can formulate this as a tiling problem $\text{TP2}(G_A, S, \{m_1, \ldots, m_k\})$
- Find a coloring of $G_A, S$ where color $i$ is used at most $m_i$ times such that number of non-monochromatic (cut) edges are minimized.