Calculations in the Era of Contemporary Computing

D. V. Chudnovsky, G. V. Chudnovsky, T. Morgan
IMAS
NYU Tandon School of Engineering
6 MetroTech Center
Brooklyn, NY 11201

September 26, 2016

Why

How

Hardware Role in Practical Problems

Hardware drives the improvement in solution of large and difficult problems.

Moore's law-holding so far...
Economic vs. technical limitations (around 5 nm feature size?).

Particular areas where combination of massive processing and algorithmic improvements made a difference:
Linear Algebra (LINPACK LAPACK, 5cal PACK, etc.)

Solution of systems of nonlinear equations, including computer algebra approach to Groebner basis;

Breakthroughs in Mixed Integer Linear Programming algorithms, allowing for solution of large systems of linear programming equations and inequalities;

Significant speed-up of various techniques of building neural networks and other "trainable systems" of data analysis;
For 2D and 3D data segmentation, analysis and recognition, quantization based on massive convolution processing, derived from increased DSP capabilities, brings close to reality live video stream interpretation;

Big Problems

Progress in rapid processing of large graphs, expressing layers of relations between diverse items, including those with dynamic updates.

Big Data Needs Big silicon

In general hard problem come from big data, but to solve *NP* problems of small sizes, even by approximation, still requires exponential (in problem size) time – ameliorated by parallel computing.

Parallelism to the Rescue?

Not everything is "infinitely parallelizable", even when problem sizes increase. Famous Amdahl's and Gustafson laws state that a non-parallel portion of the program is a low bound no matter how much one can execute in parallel other parts of the program.

In the simplest case if you run programs on N processors (even if programs are "embarrassingly parallel"), and you need to bring to the outside the results accumulated from these N processors,
then the time needed to do so is bounded from below by:

$$O(1) \cdot \mathrm{Size} \cdot \mathrm{Diameter}$$

where "Size" is a size of data you output, and "$\mathrm{Diameter}$" is a Diameter of the graph of the network connecting your processors.

In the case of the best switching network you get

$$O(\mathrm{Size} \cdot \log N)$$

as a low bound, but more typically for very large N it is:

$$O(\mathrm{Size} \cdot \sqrt{N})$$

Physical Limits to Parallelism

Reason: the physical reality is that the switching network requires large 3D volume.

Kolmogorov was first who gave the bound on the volume of connections ("cables and wires") required by network graphs.
For moderate N (many thousands) one can use exotic graphs arising form Number Theory (Cayley graphs), see (D.V., G.V., M., 1986).

However they are practically difficult to assemble and extremely difficult to repair. Thus people always use in giant machines modular hierarchical arrangements that can be build and monitored:
M-dimensional grids, torus, trees, shuffle networks –
as building blocks.

Sorting as a Prototype

Use parallel computations to speed up the solution, not to improve the utilization (communication cost typically increases).

Sorting example. The average (and worst case) cost of sorting of N items on a serial device is bounded by

$$O(N \cdot \log N)$$

with a small constant (depending on a cost of compare).
Parallel hardware speeds up sort tremendously. The theoretical lower bound is

$$O(\log N)$$

and there is a wonderful algorithm that shows that this bound can be met.
Alas, the constant in $O()$ term makes existing theoretical algorithm practically unfeasible.

Nevertheless if your data are sufficiently random, there are heuristics that approach this speedup on large parallel machines.
For more complex (and realistic cases) the best practical parallel algorithm runtime is:

$$O(\log^2 N)$$

but with small constant; in practice this one is also good in keeping communication patterns regular and manageable.

Graphs

A potent abstract device.

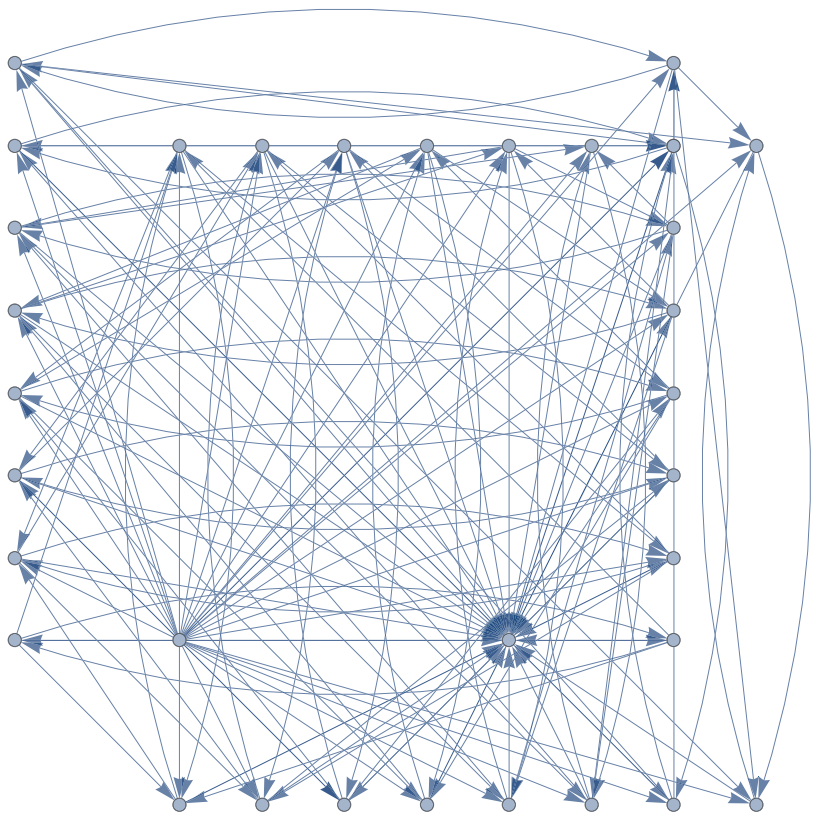
An arbitrary set V of objects, called vertices.

Associated with it a set E of "relations" between vertices, called edges:

$$e = \{u, v\} \quad \text{for (some)} \quad u, v \in V$$

directed ($e = u \to v$) or undirected.

This is an inner node in our graphs of the moment:



NP-hard Problems and Integer Linear Programming Formulation

Classical *NP*-hard problem: Traveling Salesman Problem.
This problem, like many other hard combinatorial ones, can be reduced to the solution of Integer Linear Programming (ILP) equations/inequalities.

We have sites: $0, \ldots, n$. Introduce auxiliary 0/1 variables $x_{i,j}$, and integer variables u_i. Here $x_{i,j} = 1$ iff there is a path from a site i to j. Let $d_{i,j}$ is the "distance" from i to j.
The problem is represented as the following integer linear programming problem. Constraints:

$$\sum_{i, i \neq j} x_{i,j} = 1; \; j = 0, \ldots, n$$

$$\sum_{j, i \neq j} x_{i,j} = 1; \; i = 0, \ldots, n$$

$$u_i - u_j + n x_{i,j} \leq n - 1; \; 1 \leq i \neq j \leq n;$$

Under these constraints minimize:

$$\min \sum_{i=0}^{n} \sum_{i \neq j; j=0}^{n} c_{i,j} x_{i,j}$$

Complexity and Parallelization

The best provable upper (worst) bound for the (sequential) run time is $O(n^2 2^n)$; even an algorithm of $O(1.9999^n)$ is so far unknown.

For a general graph this is an *NP*-hard problem (harder than Hamiltonian path). However, for Euclidian distances, the $(1+\epsilon)$ approximations are solvable in polynomial time in n. The dependency on $\frac{1}{\epsilon}$ is exponential, of course.

For symmetric and metric distances there are approximation solutions within a small constant factor of the optimal.

However, in all practical cases a combination of modern ILP techniques with additional heuristic gives solutions often within small percentages of the optimal ones.

Graph Embedding

The problem here is to find an embedding of one graph, X, into another, (prototype) G, bounding the "dilation" D of the embedding $g : X \to G$:

$$\mathrm{dist}_G(g(x), g(x')) \leq D$$

for all $x, x', \mathrm{dist}_X(x, x') = 1$.

In many interesting cases either of two graphs X or G is a subgraph of a Grid Graph (for example, a full rectangle).

Computer Applications

Bounded dilation embedding of such graphs (typically to or from a square or near square) initially became very popular because of these acute problems:

A) The Placement Problem. How to create/move complex shapes of gate/transistor designs in a particular area of a VLSI design without much distortion of the wiring. Here typically the mapping is into a square. This problem is still of a vital importance.

B) The Mesh of Computers. How to align complex placement and connection of computers into conventional two-dimensional grids. Particular attention is devoted to computer clusters with possibly failing computer nodes, remapped as rectangles in $2D$ grids.

More recently:
C) Creation of simplified low-dimensional graphs from high-dimensional data sets. Usable as a rigorous technique of multidimensional scaling analysis; sometimes provides insights.

"Simple" mappings

It may be counterintuitive (unlike a toothpaste): it is easier (lower dilation) to map a skinny rectangle (say, a line $N \times 1$) into a square ($N \times N$) than vice versa.

Of course, finding optimal solution (or even determining if it exists for a fixed D) in NP-hard (e.g., the finding a Hamiltonian path/tour is equivalent to the embedding of a line/circle in a graph).

It does not mean, however, that practical problems cannot be efficiently solved with enough hardware.

Boolean Linear Programming Framework

To get the optimal solution of the graph embedding problem with a given Dilation (D) one formulates it as an Integer Programming problem, similar to what maximal graph matching problem or TSP/Hamiltonian Path problem looks like.

To study the 1-1 embedding $g = g(x)$ of the graph X into G one introduces auxiliary Boolean: 0/1 variables $F[g, x]$ with a meaning that $F[g, x] = 1$ iff $g = g(x)$.
Then the basic equations that describing that this is a 1-1 embedding of all X into G are:

$$\sum_{g \in G} F[g, x] \;=\; 1 \tag{1}$$

for every vertex $x \in X$, and

$$\sum_{x \in X} F[g, x] \;\leq\; 1 \tag{2}$$

for every vertex $g \in G$.

Dilation part

The equations (1), (2) are really just a graph matching problem, solvable in a polynomial (or often in nearly linear) time. The important part here is the control of mapping "dilation" D. Various equivalent formulations are possible:

$$F[g,x] - \sum_{g' \in G, \mathrm{dist}(g,g') \leq D} F[g',x'] \ \leq \ 0 \tag{3}$$

for all $g, g' \in G$, $x, x' \in X$, $\mathrm{dist}_X(x,x') = 1$, or instead:

$$F[g,x] + F[g',x'] \ \leq \ 1 \tag{4}$$

for all $g, g' \in G$, $\mathrm{dist}(g,g') > D$, $x, x' \in X$, $\mathrm{dist}(x,x') = 1$.

Here (4) provides more equations than (3), but the system (2),(4) has a huge advantage. It forms a totally unimodular matrix. The ILP problem for a totally unimodular matrix is simply reducible to just an LP problem (those can be solved in a polynomial time).

While (1) spoils the unimodularity (otherwise NP=P), the (2), (4) parts help with solving rather giant ILP problems.

Complexity

Because the bandwidth problem for grid subgraphs is *NP*-complete, even simplest dilation problems (for grid subgraphs) are *NP*-complete as well.

While the embedding of the minimal dilation is very hard to find, for many classes of grid subgraphs optimal or nearly optimal dilation embedding are possible to find using ILP methodology.

This turns out is very important in many chip design problems (placement/routing) with millions of objects (gates, wires) in 2D areas (typically at $250 \times 250\mu^2$ areas).

However, not all graph mapping problems are actually that hard.
The graph isomorphism problem – on whether two graphs are actually the same if one changes the labels of vertices and edges – was determined (less than a year ago) to be solvable in quasi-polynomial time.

Cliques and Bicliques

In a bipartite graph $bg =< V, E >$ the set V of vertices is a union of two non-intersecting groups: L and R, $V = L \cup R$, such that all edges in E only connect vertices in L with vertices in R.

A clique C in a (undirected) graph g is a complete subgraph of g (i.e. every two vertices in C are connected by an edge).
A maximal clique is not contained in any other clique.

For a bipartite graph $bg =< L \cup R, E >$ a proper notion is a biclique.
A set $B = BL \cup BR$ is a biclique in bg iff $BL \subset L, BR \subset R$, and all vertices $u \in BL, v \in BR$ are connected by an edge.
Maximal biclique is not contained in any other biclique.

Problems of finding maximal clique of a given size $|C|$ in graphs, and maximal bicliques B of a given size ($\geq |BL| \cdot |BR|$) in bipartite graphs are NP-hard.

The number of maximal cliques and bicliques in interesting graphs (e.g. in random graphs) is exponential.
Still they can be enumerated by massive parallel computational efforts.

Combinatorial methods (starting from Malgrange, 1962).

Optimization Methods for (bi)Cliques

Motzkin-Straus Lagrangian formulation for a graph g:

$$\Lambda_g(\mathbf{x}) \ = \sum_{\{u,v\} \in E(g)} x_u \cdot x_v$$

Maximize $\Lambda_g(\mathbf{x})$ over all $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbf{R}^n$, $x_i \geq 0, i = 1, \ldots, n$; $\sum_{i=1}^{n} x_i = 1$ for $\mathbf{x} = \mathbf{x}_m$. Then the size of the largest clique in g is

$$\frac{1}{1 - 2 \cdot \Lambda_g(\mathbf{x}_m)}$$

Integer Linear Programming formulation of finding (maximal) bicliques in the bipartite graph $bg =< L \cup R, E >$ uses the bi-Adjacency 0/1 matrix $BA(L, R)$ of bg, with

$$BA(l, r) \ = 1 \text{ iff } \{l, r\} \in E(bg);\ l \in L,\ r \in R$$

ILP Formulation and Maximum Bicliques

To determine maximum bicliques in a graph bg with a (possibly weighted) bi-Adjacency matrix $BA(L, R)$ one can set these rather simple linear constraints on Boolean (0/1) vectors $L(l), l \in L$, $R(r), r \in R$:

$$\sum_{l \in L} L(l) \cdot B(l, r) \geq |BL| \cdot R(r), \ \ r \in R,$$

$$\sum_{r \in R} R(r) \cdot B(l, r) \geq |BR| \cdot L(l), \ \ l \in L,$$

$$\sum_{l \in L} L(l) = |BL|,$$

$$\sum_{r \in R} R(r) = |BR|,$$

Here $L(l) = 1$ in the Boolean vector L iff $l \in L$ is a element of the set BL in the biclique $B = BL \cup BR$ (and similarly for $R(r)$).

Replacing $< |BL|, |BR| >$ in the first two constrains by $(1 - \epsilon) \cdot < |BL|, |BR| >$ provides the conditions for finding so-called ϵ-bicliques (when the suspicion of noise is high).

Binary/Boolean Matrix Decomposition Problem

Start with Boolean (0/1) matrix $B(n, m)$ that one should think of as a bi-Adjacency matrix of a bipartite graph $bg =< L \cup R, E >$, $|L| = n$, $|R| = m$.

Binary decomposition of a matrix B means binary/Boolean product representation:

$$B(n, m) = S(n, k) \times_B T(k, m).$$

Here $\times_B$ is Boolean matrix product. In Boolean ring, the multiplication (among $0, 1$) is as usual, but the addition is replaced by an Or, i.e. $1 + 1 = 1$. Thus one can write elements of the matrix B as:

$$B_{l,r} = \vee_{i=1}^{k}(S_{l,i} \wedge T_{i,r}),\ l = 1, \ldots, n;\ r = 1, \ldots, m$$

S is called a "usage matrix", reporting which subjects appear in observations, and T is called the "basis vector matrix", describing which attributes appear in each subject.

Singular Value Decomposition

Compare with the SVD, a backbone of conventional linear and nonlinear multidimensional analysis, optimal in L_2 norm.
In the case of a real matrix B its SVD representation has the form:

$$B = U.\Sigma.V$$

with orthogonal matrices U, V, $U^T.U = V.V^T = I_k$, and a diagonal matrix Σ with non-negative elements.

This effectively solves the optimal k-rank approximation problem for B, when one retains k largest singular values in Σ.
Here rank 1 matrix is just

$$\vec{a}.\vec{b}^T$$

with elements $\{i, j\}$ as $a_i \cdot b_j$. Thus real rank k matrix B is represented as:

$$B_{l,r} = \sum_{i=1}^{k} (\vec{a}_i)_l \cdot (\vec{b}_j)_r$$

as compared with binary/Boolean matrix representation of k "tiles":

$$B_{l,r} = \vee_{i=1}^{k} (\vec{a}_i)_l \cdot (\vec{b}_j)_r$$

Tiles as bicliques

Tiles are just bicliques with

$$BL = \{l \in L : a_l = 1\}; BR = \{r \in R : b_r = 1\};$$

Thus a problem of determining the Boolean rank of a matrix B (for a bipartite graph bg) becomes a problem of:
finding a minimal number of (maximal) bicliques in the graph bg that cover all vertices.

Of course, this is also an NP-hard problem, arising from yet another NP-complete classical "Set Basis Problem".

Luckily, there is very good heuristics that can select from a complete (or large enough) set of bicliques a proper subset that gives a good Boolean k-rank approximate decomposition.