Computer-Aided Search for Matrix Multiplication Algorithms

Matthew Anderson Zongliang Ji Anthony Yang Xu

UNION COLLEGE

December 13, 2017

Simons Institute for the Theory of Computing

Matrix Multiplication

Problem

Input: $A \in \mathbb{F}^{n\times n}$, $B \in \mathbb{F}^{n\times n}$

Output: $C = A \times B \in \mathbb{F}^{n\times n}$.

For example:

$$\begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix} \times \begin{bmatrix} -1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 5 \\ -2 & 6 \end{bmatrix}$$

How many operations does it take to multiply two n-by-n matrices?

- $O(n^3)$ by naively computing n^2 dot products of rows of A and columns of B.
- $\Omega(n^2)$ because there are at n^2 cells to output.

Question

What is the smallest $\omega \leq 3$ such that n-by-n matrix multiplication can be done in time $O(n^\omega)$?

Progress on ω

3	Naive
$\underline{2}.808$	Strassen 1969
$2.\underline{7}96$	Pan 1978
$2.7\underline{8}$	Bini et al 1979
$2.\underline{5}22$	Schönhage 1981
$2.\underline{4}96$	Coppersmith & Winograd 1982
$2.4\underline{7}9$	Strassen 1986
$2.\underline{3}75477$	Coppersmith & Winograd 1987
$2.37\underline{4}$	Stothers 2010
$2.37\underline{2}8642$	Williams 2011
$2.37286\underline{3}9$	Le Gall 2014

Outline

- Introduction
- Cohn-Umans Framework
- Checking
- Search
- Lessons

Cohn-Umans Framework

In 2003, Cohn and Umans proposed an approach for improving the upper bound on ω.

- Inspired by the $\Theta(n \log n)$ FFT-based algorithm for multiplying two degree n univariate polynomial, c.f., e.g., [CLRS 2009, Chap 30].

$$A \times B = C \text{ becomes } \mathrm{FFT}^{-1}(\mathrm{FFT}(A) * \mathrm{FFT}(B)) = C$$

Cohn-Umans Framework

In 2003, Cohn and Umans proposed an approach for improving the upper bound on ω.

- Inspired by the $\Theta(n \log n)$ FFT-based algorithm for multiplying two degree n univariate polynomial, c.f., e.g., [CLRS 2009, Chap 30].

$$A \times B = C \text{ becomes } \mathrm{FFT}^{-1}(\mathrm{FFT}(A) * \mathrm{FFT}(B)) = C$$

Idea determine a suitable group G to embed multiplication into the group algebra $\mathbb{C}[G]$ using sets $X, Y, Z \subseteq G$, with $|X| = |Y| = |Z| = n$.

$$\overline{A} = \sum_{i,j \in [n]} (x_i^{-1} y_j) A_{i,j}, \quad \overline{B} = \sum_{j,k \in [n]} (y_j^{-1} z_k) B_{j,k}, \quad \overline{C} = \sum_{i,k \in [n]} (x_i^{-1} z_k) C_{i,k}$$

where triple product property holds: $\forall x, x' \in X, \forall y, y' \in Y, \forall z, z' \in Z$,

$$x^{-1} y y'^{-1} z = x'^{-1} z' \text{ iff } x = x', y = y', z = z'.$$

ω implied by G depends on $|G|$ and aspects of its representation.

Puzzles

Definition (Puzzle)

An (s,k)-*puzzle* is a subset $P \subseteq U_k = \{1,2,3\}^k$ with $|P| = s$.

Consider

$$P = \{(3,1,3,2),(1,2,3,2),(1,1,1,3),\\ (3,2,1,3),(3,3,2,3)\}$$

- P is a (5,4)-puzzle.
- P has five *rows*.
- P has four *columns*.

P

3	1	3	2
1	2	3	2
1	1	1	3
3	2	1	3
3	3	2	3

Note that:

- The columns are ordered.
- The rows are unordered (as P is a set).

Uniquely Solvable Puzzles – Intuition

We're interested in puzzles that are uniquely solvable.

Uniquely Solvable Puzzles – Intuition

We're interested in puzzles that are uniquely solvable.

3	2	3	2
1	1	3	2
1	2	1	3
3	1	1	3
1	3	2	1

- This puzzle is not uniquely solvable.

Uniquely Solvable Puzzles – Intuition

We're interested in puzzles that are uniquely solvable.

3	2	3	2

1	1	3	2

1	2	1	3

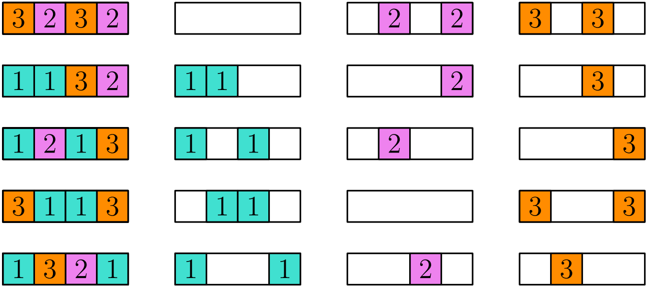
3	1	1	3

1	3	2	1

- This puzzle is not uniquely solvable.

Uniquely Solvable Puzzles – Intuition

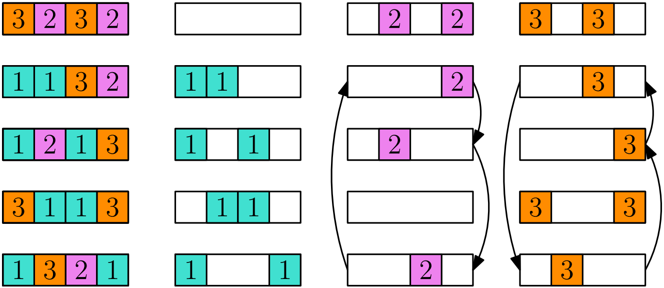
We're interested in puzzles that are uniquely solvable.



- This puzzle is not uniquely solvable.

Uniquely Solvable Puzzles – Intuition

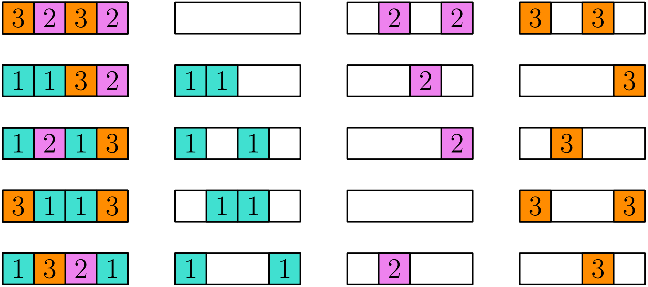
We're interested in puzzles that are uniquely solvable.



- This puzzle is not uniquely solvable.
- Can be witnessed by three permutations:
  $\pi_1 = (1)(2)(3)(4)(5)$
  $\pi_2 = (1)(2\ 3\ 5)(4)$
  $\pi_3 = (1)(2\ 5\ 3)(4)$

Uniquely Solvable Puzzles – Intuition

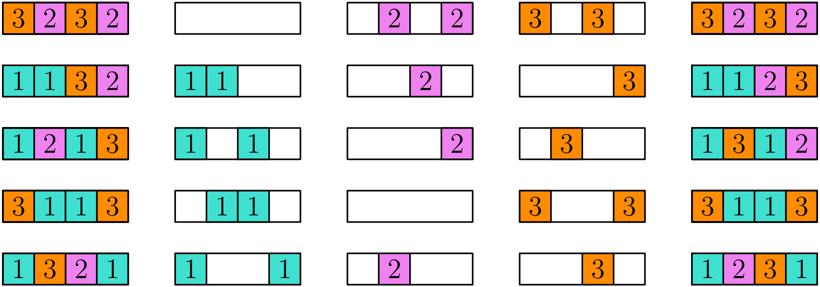
We're interested in puzzles that are uniquely solvable.



- This puzzle is not uniquely solvable.
- Can be witnessed by three permutations:
  $\pi_1 = (1)(2)(3)(4)(5)$
  $\pi_2 = (1)(2\ 3\ 5)(4)$
  $\pi_3 = (1)(2\ 5\ 3)(4)$

Uniquely Solvable Puzzles – Intuition

We're interested in puzzles that are uniquely solvable.



- This puzzle is not uniquely solvable.
- Can be witnessed by three permutations:
  $\pi_1 = (1)(2)(3)(4)(5)$
  $\pi_2 = (1)(2\ 3\ 5)(4)$
  $\pi_3 = (1)(2\ 5\ 3)(4)$
- Since the resulting puzzles is not the same as the original puzzle (even reordering rows), the puzzle is not uniquely solvable.

Uniquely Solvable Puzzles – Formal

Definition (Uniquely Solvable Puzzle)

An (s, k)-puzzle P is *uniquely solvable* if $\forall \pi_1, \pi_2, \pi_3 \in \mathrm{Sym}_P :$

1. either $\pi_1 = \pi_2 = \pi_3$, or
2. $\exists r \in P, \exists i \in [k]$ such that at least two of the following hold:
    1. $(\pi_1(r))_i = 1$,
    2. $(\pi_2(r))_i = 2$,
    3. $(\pi_3(r))_i = 3$.

Basically, for every way of non-trivial way of reordering the 1-, 2-, 3-pieces according to π_1, π_2, π_3, they cannot all fit together.

Uniquely Solvable Puzzles – Formal

Definition (Uniquely Solvable Puzzle)

An (s, k)-puzzle P is *uniquely solvable* if $\forall \pi_1, \pi_2, \pi_3 \in \mathrm{Sym}_P$:

1. either $\pi_1 = \pi_2 = \pi_3$, or
2. $\exists r \in P, \exists i \in [k]$ such that at least two of the following hold:
    1. $(\pi_1(r))_i = 1$,
    2. $(\pi_2(r))_i = 2$,
    3. $(\pi_3(r))_i = 3$.

Basically, for every way of non-trivial way of reordering the 1-, 2-, 3-pieces according to π_1, π_2, π_3, they cannot all fit together.

- This is a natural property that holds of "good" real-world puzzles:
    - jigsaw puzzles (locally), and
    - sudoku puzzles (globally).

Strong Uniquely Solvable Puzzles

Definition (Strong Uniquely Solvable Puzzle)

An (s, k)-puzzle P is *strong uniquely solvable* if $\forall \pi_1, \pi_2, \pi_3 \in \mathrm{Sym}_P :$

1. either $\pi_1 = \pi_2 = \pi_3$, or
2. $\exists r \in P, \exists i \in [k]$ such that exactly two of the following hold:
   1. $(\pi_1(r))_i = 1$,
   2. $(\pi_2(r))_i = 2$,
   3. $(\pi_3(r))_i = 3$.

Strong Uniquely Solvable Puzzles

Definition (Strong Uniquely Solvable Puzzle)

An (s, k)-puzzle P is *strong uniquely solvable* if $\forall \pi_1, \pi_2, \pi_3 \in \mathrm{Sym}_P$:

1. either $\pi_1 = \pi_2 = \pi_3$, or
2. $\exists r \in P, \exists i \in [k]$ such that exactly two of the following hold:
   1. $(\pi_1(r))_i = 1$,
   2. $(\pi_2(r))_i = 2$,
   3. $(\pi_3(r))_i = 3$.

No good intuition for the "exactly two" part, but a useful implication.

Lemma ([CKSU 05, Corollary 3.6])

For an integer $m \geq 3$, if there is a strong uniquely solvable (s, k)-puzzle,

$$\omega \leq \frac{3 \log m}{\log(m-1)} - \frac{3 \log s!}{sk \log(m-1)}.$$

Useful Strong Uniquely Solvable Puzzles

Lemma ([CKSU 05, Proposition 3.8])

There is an infinite family of SUSP that achieve $\omega < 2.48$.

There are group-theoretic constructions derived from [Strassen 86] and [Coppersmith-Winograd 87] that achieve the ω's of those works.

Useful Strong Uniquely Solvable Puzzles

Lemma ([CKSU 05, Proposition 3.8])

There is an infinite family of SUSP that achieve $\omega < 2.48$*.*

There are group-theoretic constructions derived from [Strassen 86] and [Coppersmith-Winograd 87] that achieve the ω's of those works.

Lemma ([BCCGU 16])

SUSP cannot show $\omega < 2 + \epsilon$*, for some* $\epsilon > 0$*.*

- This was conditionally true if the Erdö-Szemeredi sunflower conjecture held [Alon-Shpilka-Umans 2013].
- Recent progress on cap sets and arithmetic progressions made this unconditional [Ellenberg 2016, Croot-Lev-Pach, 2016].

Our Goals & Approach

Goal Find strong uniquely solvable puzzles (SUSP) that imply smaller ω.

Our Goals & Approach

Goal Find strong uniquely solvable puzzles (SUSP) that imply smaller ω.

Approach

- For fixed width k, the larger height s of a SUSP is, the smaller ω is implied. We want to determine for small values of k, the maximum s that can be achieved. Hopefully, this leads to an improvement in ω.
- Develop software platform to explore and experiment with SUSP.
- Algorithm Design
  - Checking that a puzzle is a SUSP.
  - Searching for large SUSP.
- Implementation
  - Targeted mainly desktop but also HPC environments.
- We only need to find one sufficiently large puzzle to achieve a new algorithm – worst-case performance isn't a good metric!

Our Goals & Approach

Goal Find strong uniquely solvable puzzles (SUSP) that imply smaller ω.

Approach

- For fixed width k, the larger height s of a SUSP is, the smaller ω is implied. We want to determine for small values of k, the maximum s that can be achieved. Hopefully, this leads to an improvement in ω.
- Develop software platform to explore and experiment with SUSP.
- Algorithm Design
  - Checking that a puzzle is a SUSP.
  - Searching for large SUSP.
- Implementation
  - Targeted mainly desktop but also HPC environments.
- We only need to find one sufficiently large puzzle to achieve a new algorithm – worst-case performance isn't a good metric!

Secondary Goal Develop a theory research program that undergraduates can meaningfully participate in.

Outline

- Introduction
- Cohn-Umans Framework
- Checking
- Search
- Lessons

Checking

Problem (SUSP-Check)

Input: *A (s,k)-puzzle P.*

Output: *True iff P is a strong uniquely solvable puzzle.*

It suffices to evaluate the following formula for a puzzle P:

$$\begin{aligned}
&\forall \pi_1, \pi_2, \pi_3 \in \mathrm{Sym}_P. \\
&\qquad \pi_1 = \pi_2 = \pi_3 \\
&\quad \vee \exists r \in P. \exists i \in [k]. ((\pi_1(r))_i = 1) + ((\pi_2(r))_i = 2) + ((\pi_3(r))_i = 3) = 2
\end{aligned}$$

- That a P is not a SUSP is be witnessed by permutations π_1, π_2, π_3.
- SUSP-Check is in coNP.
- When we only want to verify uniquely solvability it is reducible to graph automorphism.
- It is not clear whether SUSP-Check is coNP-hard.

Brute Force

$$\forall \pi_1, \pi_2, \pi_3 \in \mathrm{Sym}_P.$$
$$\pi_1 = \pi_2 = \pi_3$$
$$\vee\ \exists r \in P.\exists i \in [k].((\pi_1(r))_i = 1) + ((\pi_2(r))_i = 2) + ((\pi_3(r))_i = 3) = 2$$

- Brute force model checking takes $O((s!)^3 \cdot \mathrm{poly}(s,k))$ time.
- Easy to implement.
- Run time makes it practically useless for puzzles with width $k > 4$.
- Served as a reference implementation for debugging.
- Good for getting students feet wet with relevant issues with implementation and mathematical objects.
- It will be more convenient to think about checking the complement formula.

$$\exists \pi_1, \pi_2, \pi_3 \in \mathrm{Sym}_P.$$
$$\pi_1, \pi_2, \pi_3 \text{ not all equal}$$
$$\wedge\ \forall r \in P.\forall i \in [k].((\pi_1(r))_i = 1) + ((\pi_2(r))_i = 2) + ((\pi_3(r))_i = 3) \neq 2$$

Pruning

$$\exists \pi_1, \pi_2, \pi_3 \in \mathrm{Sym}_P.$$
$$\pi_1, \pi_2, \pi_3 \text{ not all equal}$$
$$\wedge \forall r \in P. \forall i \in [k].((\pi_1(r))_i = 1) + ((\pi_2(r))_i = 2) + ((\pi_3(r))_i = 3) \neq 2$$

- Force $\pi_1 = 1$ to get:

$$\exists \pi_2, \pi_3 \in \mathrm{Sym}_P.$$
$$1, \pi_2, \pi_3 \text{ not all equal}$$
$$\wedge \forall r \in P. \forall i \in [k].(r_i = 1) + ((\pi_2(r))_i = 2) + ((\pi_3(r))_i = 3) \neq 2$$

- Results in an equivalent formula because the rows of a puzzle are unordered.
- Removes a $s!$ factor from runtime, achieving $O((s!)^2 \cdot \mathrm{poly}(s, k))$.

Preprocessing

$\exists \pi_2, \pi_3 \in \mathrm{Sym}_P.$

π_2, π_3 not both 1

$\wedge \forall r \in P. \forall i \in [k].(r_i = 1) + ((\pi_2(r))_i = 2) + ((\pi_3(r))_i = 3) \neq 2$

- The innermost $\exists$ can be precomputed in $O(s^3 k)$ time by creating a Boolean relation $T_P \in P \times P \times P$, where

  $$(p, q, r) \in T_P \Leftrightarrow \forall i \in [k].(r_i = 1) + ((\pi_2(r))_i = 2) + ((\pi_3(r))_i = 3) \neq 2.$$

- This simplifies the formula we are checking to:

  $$\exists \pi_2, \pi_3 \in \mathrm{Sym}_P . \pi_2, \pi_3 \text{ not both } 1 \wedge \forall r \in P.(r, \pi_2(r), \pi_3(r)) \in T_P.$$

- This makes the dominant term of the running time independent of k and is useful for the next step.

Reduction to 3D Matching

This results in the formula below which is true iff P is not a SUSP.

$$\exists \pi_2, \pi_3 \in \mathrm{Sym}_P . \pi_2, \pi_3 \text{ not both } 1 \wedge \forall r \in P . (r, \pi_2(r), \pi_3(r)) \in T_P .$$

This is an instance of a natural NP problem.

Problem (3D Matching)

Input: *A 3-hypergraph* $G = \langle V, E \subseteq V \times V \times V \rangle$.

Output: *True iff* $\exists M \subseteq E$ *with* $|M| = |V|$ *and* $\forall e_1 \neq e_2 \in M$, *for each coordinate* e_1 *and* e_2 *are vertex disjoint.*

We can reduce verifying P is not a SUSP to 3D matching.

- Consider $G_P = \langle P, T_P \rangle$.
- Observe that P is a not a SUSP iff G_P has a 3D matching that isn't the identity matching, i.e., $M = \{(r_1, r_1, r_1), \ldots, (r_s, r_s, r_s)\}$.
- That M isn't identity matching is necessary, but not interesting so we won't talk about it anymore.

Dynamic Programming

We can determine 3D matchings using dynamic programming.

- Fix some ordering of P: $r_1, \ldots, r_s$.
- Consider iteratively constructing a matching M of G_P where in the i^{th} step you select an edge $(r_i, *, *) \in T_P$.
- After the i^{th} step, the remaining edges that can be selected are

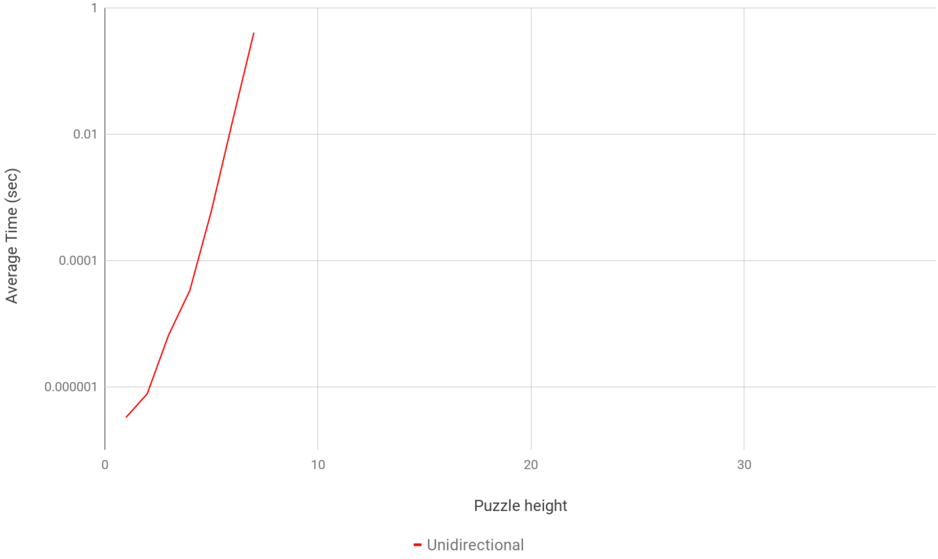
  $$T_P^{X,Y} = T_P \cap (\{r_{i+1}, \ldots, r_s\} \times (P - Y) \times (P - Z))$$

  where Y and Z are the vertices that have already be selected for the second and third coordinate respectively and $|Y| = |Z| = i$.
- Call $S(i, X, Y)$ the subproblem of whether a 3D matching can be completed on $T_P^{X,Y}$ and $i = |X| = |Y|$.
- Observe that $S(i, X, Y)$ has a 3D matching iff there exists $(r_{i+1}, p, q) \in T_P^{X,Y}$ and $S(i+1, X \cup \{a\}, Y \cup \{b\})$ has a 3D matching.

This gives an $O(2^{2s} s^2)$ algorithm via dynamic programming.

Practical Running Time – Dynamic Programming

Average checking time versus puzzle height for 50,000 (*,8)-puzzles.



Dynamic Programming + Bidirectional Search

Perform two searches using dynamic programming:

- The first selects edges whose first coordinates are $r_1, r_2, \ldots, r_{\lfloor s/2 \rfloor}$.
- The second selects edges whose first coordinates are $r_s, r_{s-1}, \ldots, r_{\lfloor s/2 \rfloor + 1}$.
- The searches use the other's memoization table in the last step.

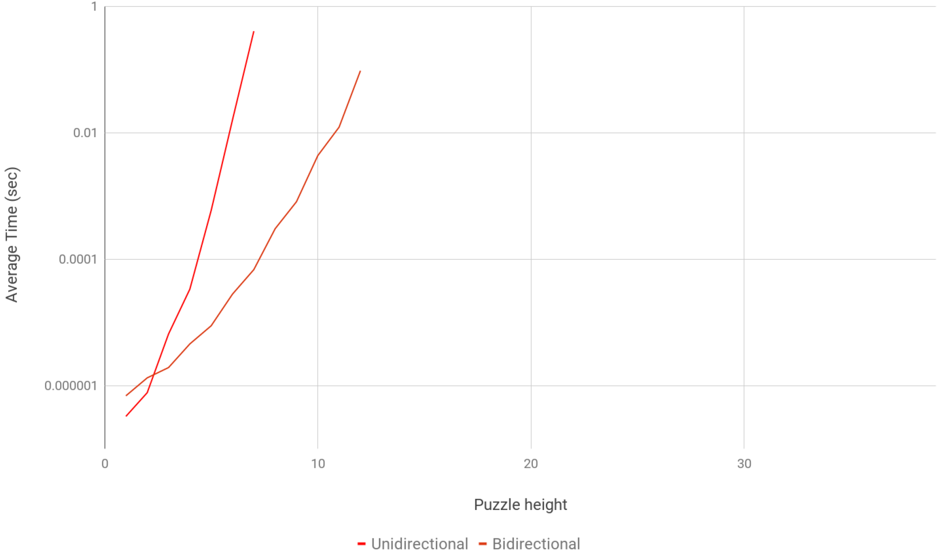
This improves performance by about a squareroot.

- The worst-case running time becomes $O(2^s s^2)$.
- The worst-case memory usage is $O(2^s s)$.

These are the best worst-case bounds we could bounds we could devise.

Practical Running Time – Bidirectional

Average checking time versus puzzle height for 50,000 (*,8)-puzzles.



Other Reductions

We tried reducing 3D matching to CNF satisfiability.

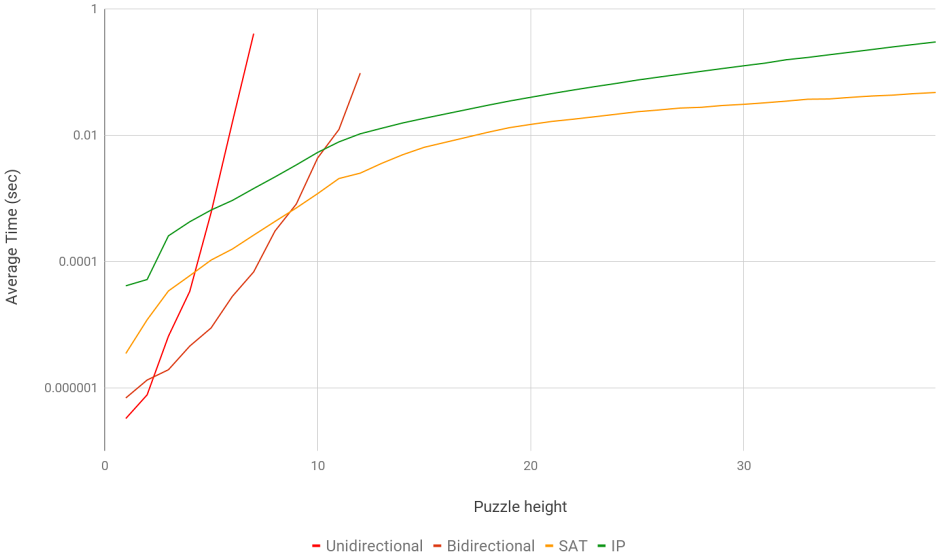
- Reduced satisfiability instance had $2s^2$ variables and $O(s^3)$ clauses.
- Used an open-source conflict-driven clause-learning SAT solver MapleCOMSPS that won the general category of the 2016 SAT Competition. Solver written in part by Jia Hui Liang, Vijay Ganesh, and Chanseok Oh.
  `http://www.satcompetition.org`

We tried reducing 3D matching to 0-1 integer programming.

- Reduced IP instance had s^3 variables and $O(s^3)$ equations.
- Used a close-source optimization library Gurobi.
  `http://www.gurobi.com`

Practical Running Time – SAT / IP

Average checking time versus puzzle height for 50,000 (*,8)-puzzles.



Implementation

Current implementation is hybrid of several algorithms.

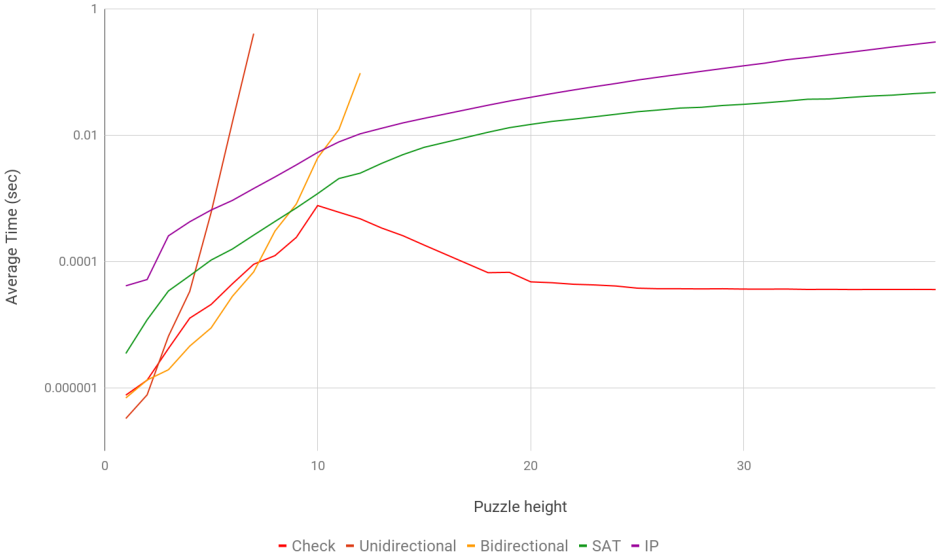
- Brute force for very small instances, $k \leq 3$.
- Bidirectional Dynamic programming for moderate instances $k \leq 6$.
- SAT for large instances with $k > 6$ and $s < 40$.
- IP for all bigger instances.

We implemented a number of heuristics that are not always conclusive, but frequently can determine the result early.

- Briefly trying to randomly or greedily generate 3D matchings.
- Verifying that all pairs of rows or triples of rows form a SUSP.
- Testing whether the puzzle is uniquely solvable using the graph isomorphism library Nauty:
  `http://users.cecs.anu.edu.au/~bdm/nauty/`
- Simplifying the 3D matching instance using properties of the puzzle, e.g., using that a column only contains two of $\{1, 2, 3\}$.

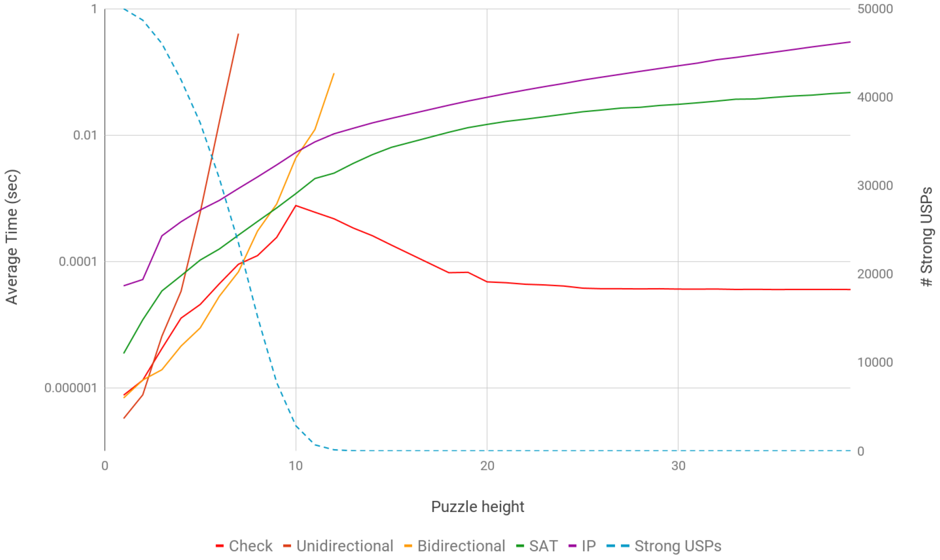
Practical Running Time – Final

Average checking time versus puzzle height for 50,000 (*,8)-puzzles.



Practical Running Time – Final

Average checking time versus puzzle height for 50,000 (*,8)-puzzles.



Outline

- Introduction
- Cohn-Umans Framework
- Checking
- Search
- Lessons

Search

Problem (SUSP-Search)

Input: $k \in \mathbb{N}$

Output: *The maximum* $s \in \mathbb{N}$ *such that there exists a* (s, k)*-puzzle that is a strong uniquely solvable puzzle.*

- Considered constructive approaches to solving this problem that use SUSP-Check as a subroutine.
- (s, k)-puzzles have sk entries and there are 3^{sk} such puzzles.
- Even eliminating symmetries, searching the full space for $k > 4$ is infeasible.
- Density of SUSPs quickly approaches 0.
- Our approaches are ad hoc and use domain knowledge for heuristics.
- SUSP do not form a matroid, augmentation property fails.

Tree Search

Lemma

If P is a SUSP and $P' \subseteq P$, then P' is a SUSP.

- This lemma allows us to construct SUSP from the bottom up.

Tree Search

Lemma

If P is a SUSP and $P' \subseteq P$, then P' is a SUSP.

- This lemma allows us to construct SUSP from the bottom up.
- BFS allowed us to explore the set of all SUSP for $k \leq 5$.
  - Implement a sequential desktop version and a parallel version to run on Union's ≈900-node HPC cluster.
  - Parallel version used MPI and Map-Reduce to maintain the search frontier and support faster verification via lookup.
  - Searching $k = 5$ originally required the cluster, but improvements to the verification algorithm made it unnecessary.
  - Searching $k = 6$ would have exceeded cluster's 32TB memory.

Tree Search

Lemma

If P is a SUSP and $P' \subseteq P$, then P' is a SUSP.

- This lemma allows us to construct SUSP from the bottom up.
- BFS allowed us to explore the set of all SUSP for $k \leq 5$.
  - Implement a sequential desktop version and a parallel version to run on Union's ≈900-node HPC cluster.
  - Parallel version used MPI and Map-Reduce to maintain the search frontier and support faster verification via lookup.
  - Searching $k = 5$ originally required the cluster, but improvements to the verification algorithm made it unnecessary.
  - Searching $k = 6$ would have exceeded cluster's 32TB memory.
- For $k \geq 6$ we implemented "greedy" algorithms for a variety of metrics:
  - $\#$ of (single, pairs, triples of) rows P could be extended by.
  - Density of the graph G_p.
  - $\#$ of columns of P which only have two entries from $\{1, 2, 3\}$.
  - Size of interval spanned by the rows of P in lexicographic order.

Combining SUSP

We've noticed the following behavior of SUSPs under set concatenation:

Observation (Experimental)

Let P_1 and P_2 be "distinct" strong uniquely solvable puzzles, then

$$P_1 \circ P_2 = \{r_1 \circ r_2 \mid r_1 \in P_1, r_2 \in P_2\}$$

is a strong uniquely solvable puzzle.

- Here "distinct" means that P_1 and P_2 each decompose into the concatenation of pairwise non-equivalent indecomposible SUSPs.
- Useful for constructing larger puzzles from smaller ones.
- No loss in implied ω.

Strong USP Found – Examples

(1,1):

1

(2,2):

1	3
2	1

(3,3):

1	1	1
3	2	1
3	3	2

(5,4):

3	1	3	2
1	2	3	2
1	1	1	3
3	2	1	3
3	3	2	3

(8,5):

3	3	3	1	1
1	1	2	2	1
2	1	3	3	2
3	2	2	2	3
2	1	2	1	3
2	2	3	1	2
3	2	3	2	1
3	1	2	1	1

(14,6):

2	3	3	1	1	1
2	1	1	2	1	1
3	3	1	2	1	1
3	2	2	2	1	1
2	3	1	1	2	1
2	2	3	1	2	1
3	3	1	3	2	1
3	2	3	3	2	1
2	1	1	3	1	2
2	3	1	3	2	2
3	1	1	1	1	3
3	3	2	3	1	3
3	3	2	1	2	3
2	2	3	2	2	3

Strong USP Found – Trends and Comparison

	[CKSU05]		This work		
Width	Height	ω^*	Height	ω^*	Search Algo
1	≤ 1		$1 =$	3.000	BFS
2	≤ 3		$2 =$	2.670	BFS
3	$3 \ldots 6$	2.642	$3 =$	2.642	BFS
4	≤ 12		$5 =$	2.585	BFS
5	≤ 24		$8 =$	2.562	BFS
6	$10 \ldots 45$	2.615	$14 \leq$	2.521	Greedy
7	≤ 86		$21 \leq$	2.531	Greedy
8	≤ 162		$30 \leq$	2.547	Greedy
9	$36 \ldots 307$	2.592	$42 \leq$	2.563	Concat
10	≤ 581		$64 \leq$	2.562	Concat
11	≤ 1098		$112 \leq$	2.540	Concat
12	$136 \ldots 2075$	2.573	$196 \leq$	2.521	Concat

- ω^* is the approximate ω in the limit of composing puzzles of these dimensions via direct product.
- [CKSU05]'s construction asymptotically implies $\omega < 2.48$.

Outline

- Introduction
- Cohn-Umans Framework
- Checking
- Search
- Lessons

Lessons

- Practical performance $\neq$ worse-case performance.
- Problem transformation is effective in theory and in practice.
- It's easy to experimentally invalidate specific hypotheses.
- It's hard to find patterns in mountains of data.
- It's hard to turn patterns from data into proofs.
- Domain knowledge is useful for pruning.
- Communication is expensive in HPC.

Future Work / Conjectures

Conjecture

There is a construction that takes SUSPs of size (s_1, k_1) *and* (s_2, k_2) *and produces a* $(s_1 + s_2, \max(k_1, k_2) + 1)$*-puzzle that is a SUSP.*

- Would imply $\omega < 2.445$.
- Consistent with the SUSP we found for $k = 1 \ldots 7$.

Search

- The current bottleneck.
- Try iterated local search.
- Try repair from concatenated puzzles.
- Try to derive better upper bounds.

Check

- Look for reductions with $o(s^3)$ size – no more 3D matching.
- Verify P is SUSP by multiplying random matrices using P.