Stochastic Algorithms and Approximations

or
You Can't Even Get Close to There from Here

James A. Foster

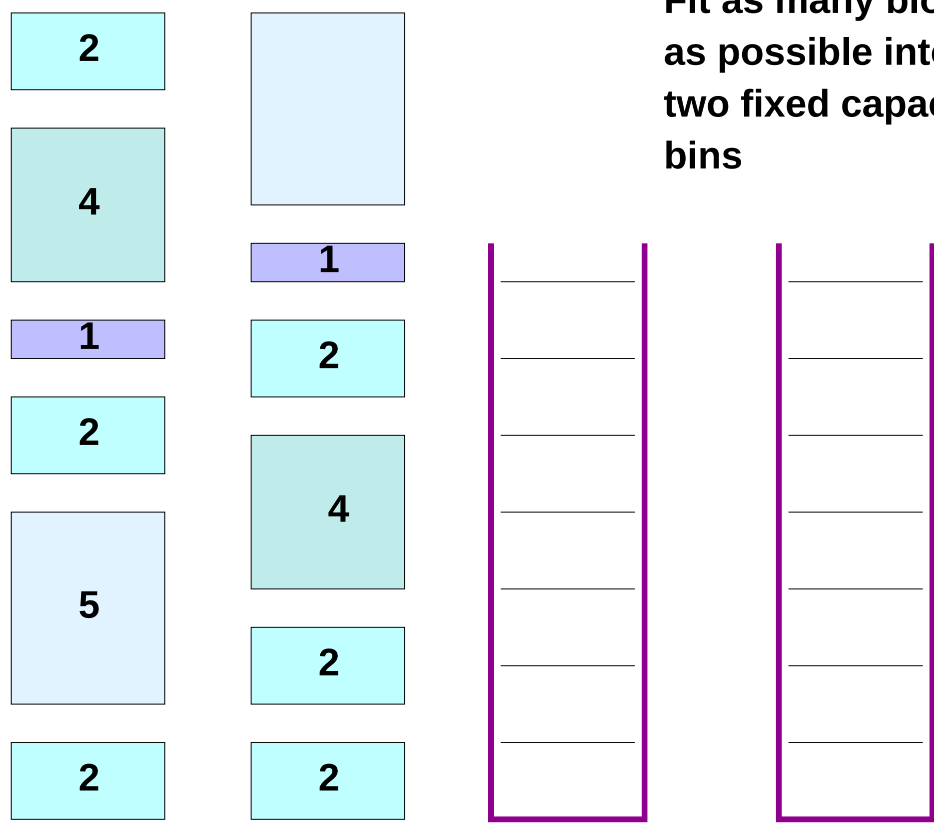
University of Idaho
Department of Computer Science
Laboratory for Applied Logic

Outline

- Some hard problems
- How hard they are
- Approximating their solutions
- Limits to approximation
- Relations to evolutionary computation

Program Packing

Fit as many blocks as possible into two fixed capacity bins

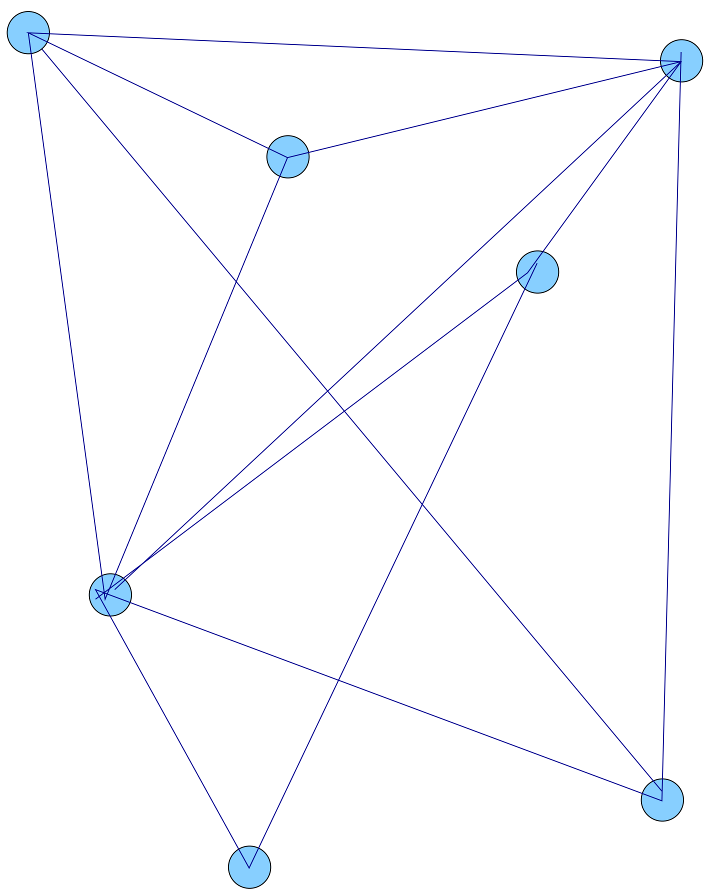


Items **Bin** **Bin**

(capacity 15 each)

Maximum Clique

Find largest set of nodes such that all nodes in the set are connected to all the others



Both are NP hard

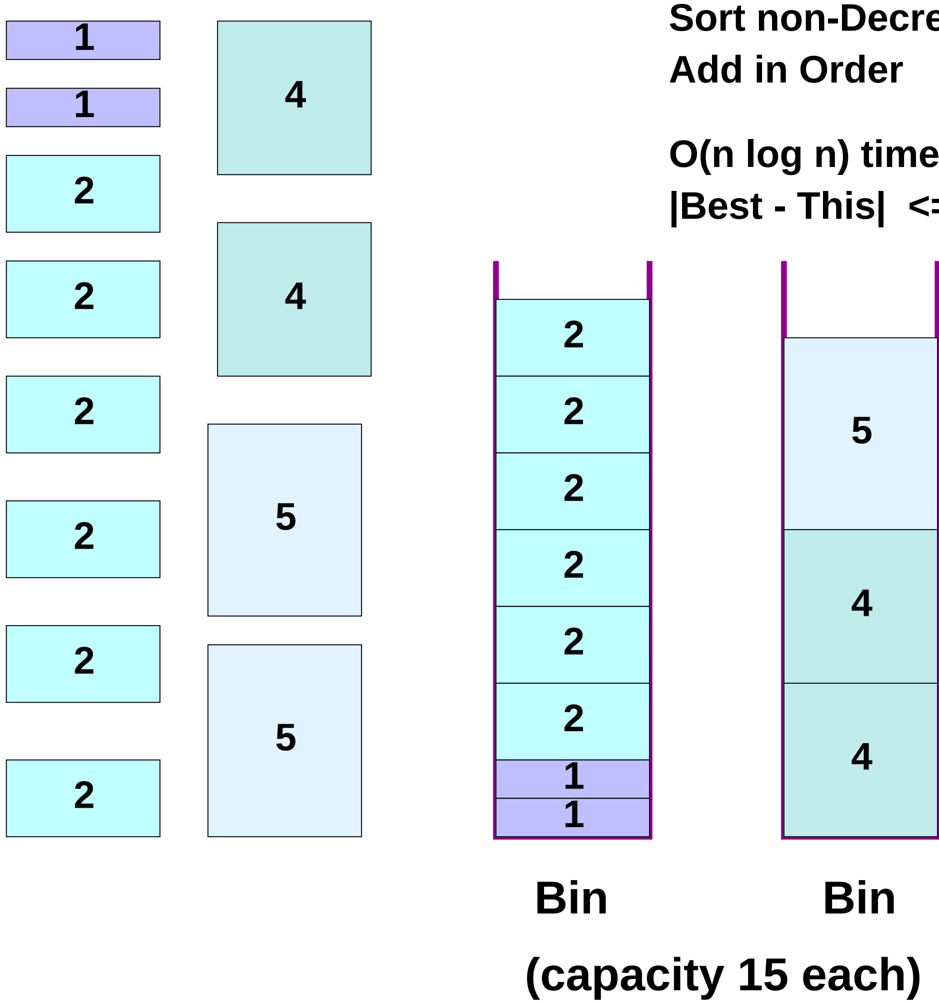
- No $O(n^k)$ solution known
- Best known: $O(2^n)$
- Fast solution exists iff P = NP
- One has fast solution iff both (and hundreds more) do

So approximate!

Program Packing Approximation

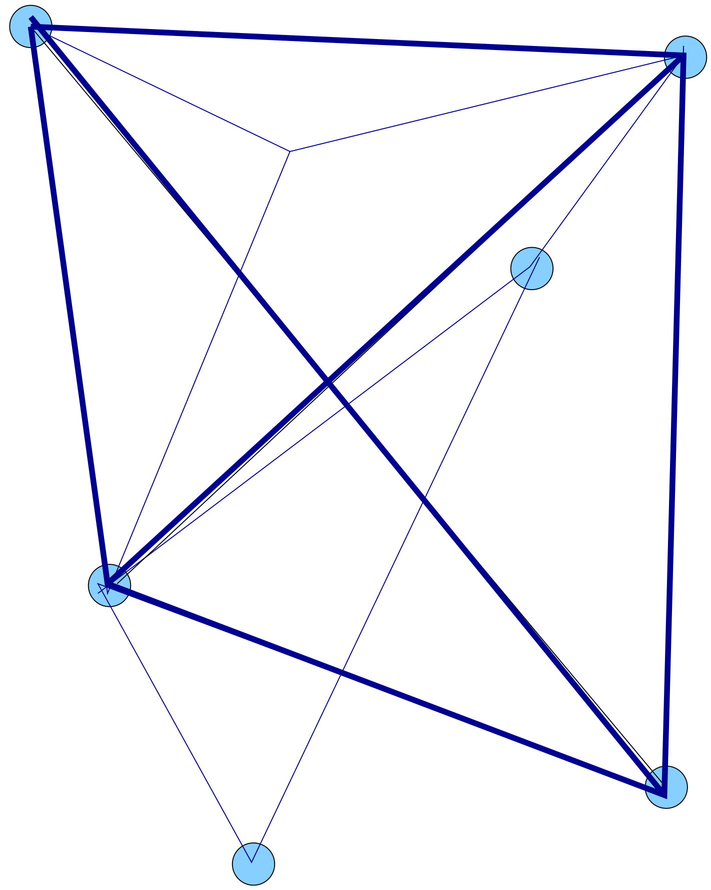
Sort non-Decreasing
Add in Order

O(n log n) time
|Best - This| <= 1



Maximum Clique

No simple approximations



How good are approximations?

Problem: Π

Algorithm A_{Π} returns $A_{\Pi}(x)$

Optimal answer: $Opt_{\Pi}(x)$

Accuracy of $A_{\Pi} = a$ where

$$acc(A_{\Pi}) = \frac{A_{\Pi}(x)}{Opt_{\Pi}(x)} \in O(a(|x|))$$

AppH(a):

$\{L$: There is an A_L with accuracy $a\}$

Some Problems

Problem	$a(n)$	*Description*
BP	1	Pack *maximum* number of programs onto two disks
TSP	2n	Find *minimum* cost Hamiltonian cycle
MaxClique	n^c	Determine the *maximum* subgraph all of whose nodes are pairwise adjacent.
ChrNum	n^c	Determine the *minimum* number of colors needed to color each vertexes so that no adjacent vertexes are the same color
DomSet	$c \log n$	Find the *minimum* collection of vertexes such that all other vertexes are adjacent to one of these
VCover	$1 + c$	Determine the *minimum* number of vertexes whose removal would eliminate all edges
StTree	$1 + c$	Determine the *minimum* number of edges in a subtree which spans the input graph

Is there a fast, good algorithm for MC?

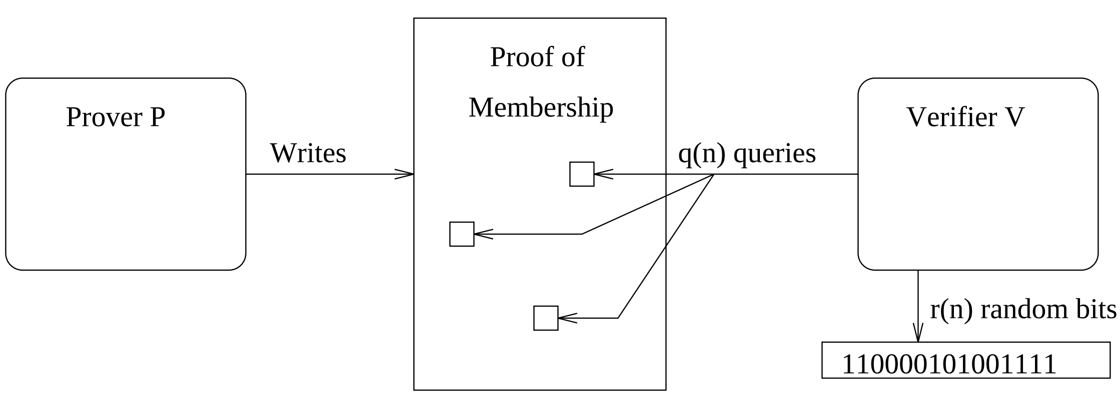
Theorem 1 (almss) *There is a constant c such that a deterministic, polynomial time algorithm A_{MC} with accuracy $|x|^{-c}$ exists iff P = NP*

In other words: "Only if pigs could fly"

Proof sketch of Theorem 1

Lemma 2 (almss) *For any $L \in NP$ and any x, $x \in L$ iff*

1. *there is a small 'proof"*
2. *which can be verified with high confidence*
3. *by a probabilistic, polynomial time, verifier*
4. *which checks a* fixed *number of bits*
5. *and uses $O(\log |x|)$ random bits*



Proof sketch of Theorem 1 (cont'd)

Det., poly time algorithm for any $L \in$ NP:

1. Construct the "proof" that $x \in L$

2. Create a "spot check" graph G:
   (a) Node for every possible bit checked ($O(1)$)
   (b) And every bit found ($O(1)$)
   (c) And every set of random bits ($O(2^{\log |x|})$)

3. Connect all consistent spot check nodes

4. Find the largest clique in G, approximation with accuracy (n^{-c}) suffices

G has large clique iff verifier accepts, $x \in L$

Is there a fast algorithm for MC which is pretty good most of the time?

Theorem 3 (Foster) *There is a constant c such that a probabilistic, polynomial time algorithm A_{MC} exists such that*

$$Pr\left[acc(A) \geq n^{-c}\right] \geq 1/2$$

iff NP = RP

Where RP is problems with probabilistic, polynomial time solutions with one sided error.

In other words: "Only if (small) pigs could fly"

Proof sketch of Theorem 3

1-sided, prob, poly time algorithm for any $L \in$ NP:

1. Construct "proof" that $x \in L$
2. Create "spot check" graph G as before
3. Approximate largest clique in G, with probability at least 1/2

- G has large clique iff verifier would accept proof (iff $x \in L$)
- In step 3, accuracy (n^{-c}) suffices
- If G has large clique, probability of finding it is greater than 1/2. If not, it can't be found. So this is one-sided.

Relation to Evolutionary Computation

EC approaches are stochastic approximations. The theorem says that *there are instances of MaxClique for which EC will always fail to produce good results*—regardless of representation and fitness evaluation.

Questions

- What problems are *hard* for EC?
- Can we quantify *hardness*?
- What instances are *hard*?
- Why?

Research Program

Our hypothesis: EC hardness is directly related to approximation hardness

- Implement ECs for problems of differing approximation complexity
- Analyze effectiveness for these problems
- Look for hard instances
- Characterize these theoretically

Conclusions

- Sometimes, it's hard to get good approximations
- Even with evolutionary computation
- Some problems are harder than others
- Understanding this will tell us when to use EC, and when not to