Confronting Hardness Using a Hybrid Approach

Virginia Vassilevska
Carnegie Mellon University

SODA 2006

Joint work with Ryan Williams and Maverick Woo

Introduction

Conventional algorithms guarantee *good* performance under a prescribed *measure:*

Introduction

Conventional algorithms guarantee *good*

performance under a prescribed *measure:*

Running Time

Introduction

Conventional algorithms guarantee *good* performance under a prescribed *measure:*

Running Time

Space

Introduction

Conventional algorithms guarantee *good* performance under a prescribed *measure:*

Running Time

Space

Simultaneous Time and Space

Introduction

Conventional algorithms guarantee *good* performance under a prescribed *measure:*

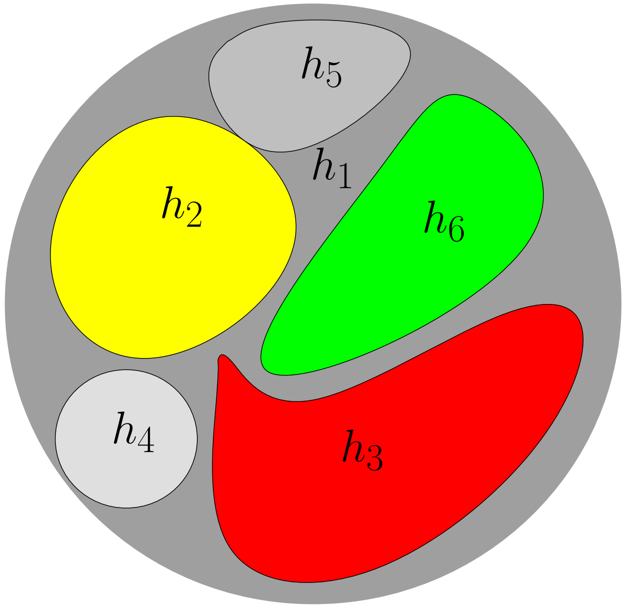
Running Time | **Simultaneous Time and Space**

Space | **Approximation Ratio and Time…**

A Hybrid Approach

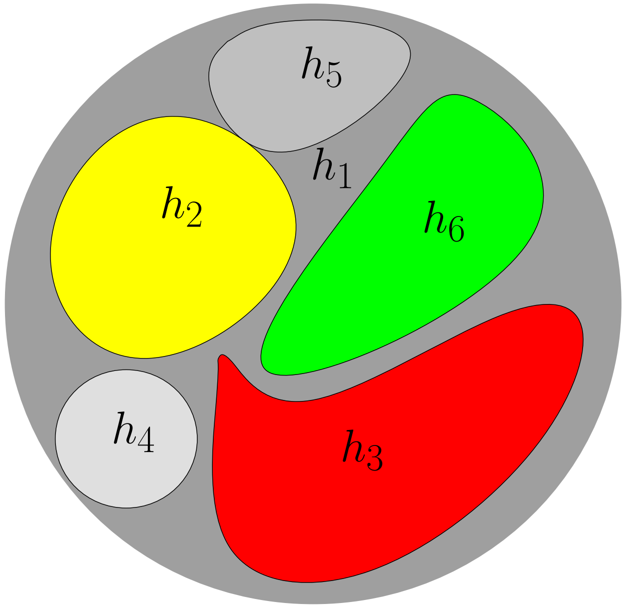
A Hybrid Approach

Consider a set $H = \{h_1, \ldots, h_k\}$ of ***heuristics***, good w.r.t. different complexity measures, partitioning the instance space.



A Hybrid Approach

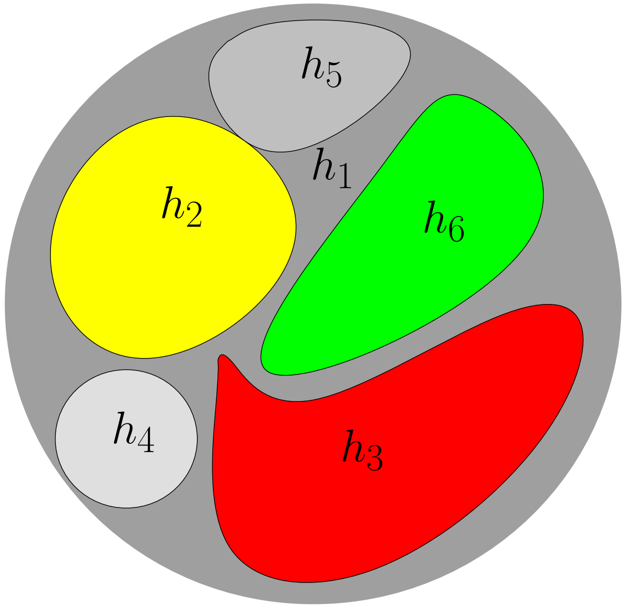
Consider a set $H = \{h_1, \ldots, h_k\}$ of *heuristics*, good w.r.t. different complexity measures, partitioning the instance space.



E.g.

A Hybrid Approach

Consider a set $H = \{h_1, \ldots, h_k\}$ of *heuristics*, good w.r.t. different complexity measures, partitioning the instance space.

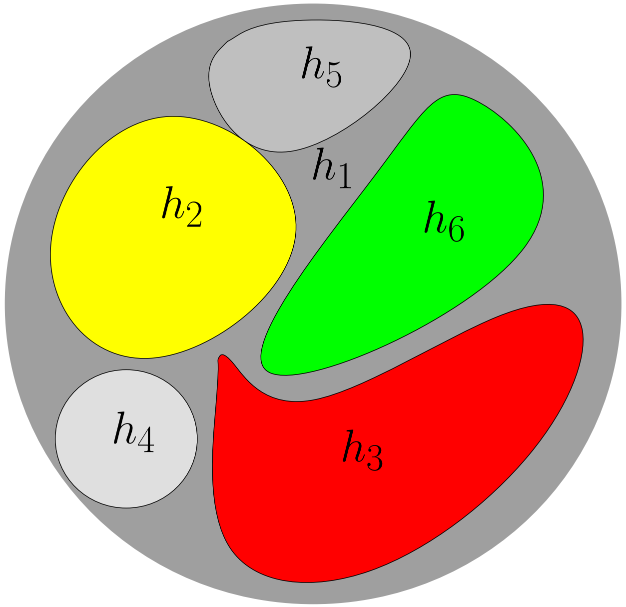


E.g.

h_1 approximates the optimal solution within a factor of α and runs in polynomial time, on all dark gray instances.

A Hybrid Approach

Consider a set $H = \{h_1, \ldots, h_k\}$ of *heuristics*, good w.r.t. different complexity measures, partitioning the instance space.



E.g.

h_1 approximates the optimal solution within a factor of α and runs in polynomial time, on all dark gray instances.

h_2 solves the problem exactly but runs in subexponential time ($2^{o(n)}$) on all yellow instances.

A Hybrid Algorithm is...

A Hybrid Algorithm is...

A set $H = \{h_1, \ldots, h_k\}$ of *heuristics*, good w.r.t. different complexity measures.

A Hybrid Algorithm is...

A set $H = \{h_1, \ldots, h_k\}$ of *heuristics*, good w.r.t. different complexity measures.

A *selector* S which on each instance selects a heuristic in polynomial time.

Hybrid Algorithms cont.

Hybrid Algorithms cont.

"Defying" Hardness: Some NP-hard problems are known or conjectured to be *hard* for several complexity measures m_i .

Hybrid Algorithms cont.

"Defying" Hardness: Some NP-hard problems are known or conjectured to be *hard* for several complexity measures m_i .

E.g. Max Independent Set can't be approximated within a factor of $n^{1-\varepsilon}$ unless $\mathsf{P} = \mathsf{NP}$ (Håstad, 1999), and can't be solved in $2^{o(n)}$ time unless SNP is in $2^{o(n)}$ time (Impagliazzo, Paturi, Zane, 1998).

Hybrid Algorithms cont.

"Defying" Hardness: Some NP-hard problems are known or conjectured to be *hard* for several complexity measures m_i .

E.g. Max Independent Set can't be approximated within a factor of $n^{1-\varepsilon}$ unless P $=$ NP (Håstad, 1999), and can't be solved in $2^{o(n)}$ time unless SNP is in $2^{o(n)}$ time (Impagliazzo, Paturi, Zane, 1998).

There exist hybrid algorithms for NP-Hard problems which for each h_i (on the instances on which S chooses to run h_i) do *strictly* better than the corresponding known hardness guarantees m_i.

MAX-CUT

Problem: Given a graph G, find a cut which maximizes the number of edges crossing it.

MAX-CUT

Problem: Given a graph G, find a cut which maximizes the number of edges crossing it.

Solvable exactly in $O(2^{m/5.2})$ by Kneis et al, 2005, or in $O(2^{\omega n/3})$ by Williams, 2004.

MAX-CUT

Problem: Given a graph G, find a cut which maximizes the number of edges crossing it.

Solvable exactly in $O(2^{m/5.2})$ by Kneis et al, 2005, or in $O(2^{\omega n/3})$ by Williams, 2004.

Approximable within $0.8785\ldots$ using SDP by Goemans and Williamson, 1995 and within 0.5 by Sahni and Gonzales, 1976 without SDP.

MAX-CUT

Problem: Given a graph G, find a cut which maximizes the number of edges crossing it.

Solvable exactly in $O(2^{m/5.2})$ by Kneis et al, 2005, or in $O(2^{\omega n/3})$ by Williams, 2004.

Approximable within $0.8785\ldots$ using SDP by Goemans and Williamson, 1995 and within 0.5 by Sahni and Gonzales, 1976 without SDP.

No better than $(1/2+\delta)$-approximation is known which runs in less than quadratic time.

Hybrid Algorithm for Max-Cut

There's a simple *hybrid* algorithm which for any $\epsilon > 0$, after a linear time test produces

- either a maximum cut in $\tilde{O}(2^{\epsilon m})$ time, or
- a $(\frac{1}{2} + \frac{\epsilon}{4})$-approximation in linear time.

A Simple Fast Hybrid Algorithm for Max-Cut

A Simple Fast Hybrid Algorithm for Max-Cut

Find a maximal matching, M.

A Simple Fast Hybrid Algorithm for Max-Cut

Find a maximal matching, M.

If $|M| < \varepsilon \frac{m}{2}$,

try all $2^{\varepsilon m}$ cuts of the vertices in M. Add the vertices from the independent set $V - M$ so that the cut is maximized.

A Simple Fast Hybrid Algorithm for Max-Cut

Find a maximal matching, M.

If $|M| < \varepsilon \frac{m}{2}$,

try all $2^{\varepsilon m}$ cuts of the vertices in M. Add the vertices from the independent set $V - M$ so that the cut is maximized.

If $|M| \geq \varepsilon \frac{m}{2}$,

for each edge in M, with probability $1/2$ choose which of its endpoints to put in A. Put the other endpoint in B;

A Simple Fast Hybrid Algorithm for Max-Cut

Find a maximal matching, M.

If $|M| < \varepsilon \frac{m}{2}$,

try all $2^{\varepsilon m}$ cuts of the vertices in M. Add the vertices from the independent set $V - M$ so that the cut is maximized.

If $|M| \geq \varepsilon \frac{m}{2}$,

for each edge in M, with probability $1/2$ choose which of its endpoints to put in A. Put the other endpoint in B;

for each vertex v not in M, with probability $1/2$ choose whether to place it in A or B.

Max Cut cont.

Max Cut cont.

If $\left|M\right| < \varepsilon \frac{m}{2}$,

we get an exact solution in $\tilde{O}(2^{\varepsilon m})$ time.

Max Cut cont.

If $|M| < \varepsilon\frac{m}{2}$,

we get an exact solution in $\tilde{O}(2^{\varepsilon m})$ time.

If $|M| \geq \varepsilon\frac{m}{2}$,

the expected size of the cut is at least

$$(\varepsilon\frac{m}{2}) + \frac{1}{2}(m - \varepsilon\frac{m}{2}) = (\frac{1}{2} + \frac{\varepsilon}{4})m.$$

We get a *linear time* $(\frac{1}{2} + \frac{\varepsilon}{4})$-approximation.

The Longest Path Problem

The Longest Path Problem

Karger, Motwani and Ramkumar, 1993: LONGEST PATH is hard to approximate within $2^{O(\frac{\log n}{\log \log n})}$, unless NP$\subseteq \bigcap_{\delta>0}$DTIME$(2^{O(n^\delta)})$.

The Longest Path Problem

Karger, Motwani and Ramkumar, 1993: LONGEST PATH is hard to approximate within $2^{O(\frac{\log n}{\log\log n})}$, unless NP$\subseteq \bigcap_{\delta>0}$DTIME$(2^{O(n^\delta)})$.

Bellman and Karp, 1962: Best known exact algorithm by dynamic programming in $\tilde{O}(2^n)$;

Alon, Yuster, Zwick, 1994: Can be extended to $2^{O(L)}$, where L is length of longest path.

The Longest Path Problem

Karger, Motwani and Ramkumar, 1993: LONGEST PATH is hard to approximate within $2^{O(\frac{\log n}{\log\log n})}$, unless NP$\subseteq \bigcap_{\delta>0}$DTIME$(2^{O(n^\delta)})$.

Bellman and Karp, 1962: Best known exact algorithm by dynamic programming in $\tilde{O}(2^n)$;
Alon, Yuster, Zwick, 1994: Can be extended to $2^{O(L)}$, where L is length of longest path.

There's a simple *hybrid* algorithm which for any $\ell(n)$

- either finds a path of length ℓ, or
- solves the LONGEST PATH exactly in $2^{O(\ell \log \ell)} n^{O(1)}$ time.

The Longest Path Problem

Karger, Motwani and Ramkumar, 1993: LONGEST PATH is hard to approximate within $2^{O(\frac{\log n}{\log\log n})}$, unless NP$\subseteq \bigcap_{\delta>0}$DTIME$(2^{O(n^\delta)})$.

Bellman and Karp, 1962: Best known exact algorithm by dynamic programming in $\tilde{O}(2^n)$;
Alon, Yuster, Zwick, 1994: Can be extended to $2^{O(L)}$, where L is length of longest path.

There's a simple *hybrid* algorithm which for any $\ell(n)$

- either finds a path of length ℓ, or
- solves the LONGEST PATH exactly in $2^{O(\ell\log\ell)}n^{O(1)}$ time.

Note for $\ell = n/polylog(n)$ we get subexponential exact running time and a polylog approximation.

A Path-or-Decomposition Lemma

A Path-or-Decomposition Lemma

Given any graph G and any $\ell > 0$ there is a poly time algorithm Path-Decomp which either finds a path of length at least ℓ or a *path decomposition* of G of width at most ℓ.

A Path-or-Decomposition Lemma

Given any graph G and any $\ell > 0$ there is a poly time algorithm Path-Decomp which either finds a path of length at least ℓ or a *path decomposition* of G of width at most ℓ.

1. Do DFS from a node v.

A Path-or-Decomposition Lemma

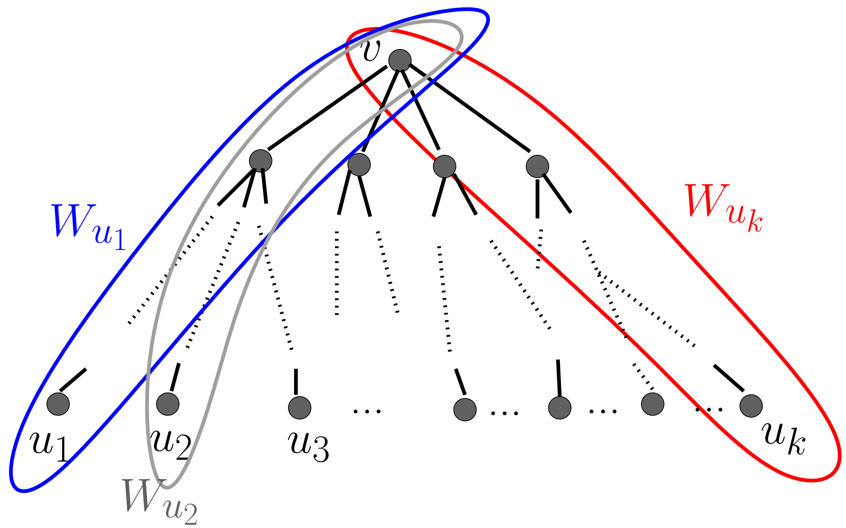
Given any graph G and any $\ell > 0$ there is a poly time algorithm Path-Decomp which either finds a path of length at least ℓ or a *path decomposition* of G of width at most ℓ.

1. Do DFS from a node v.
2. If a path P from v has length at least ℓ, stop and output P.

A Path-or-Decomposition Lemma

Given any graph G and any $\ell > 0$ there is a poly time algorithm Path-Decomp which either finds a path of length at least ℓ or a *path decomposition* of G of width at most ℓ.

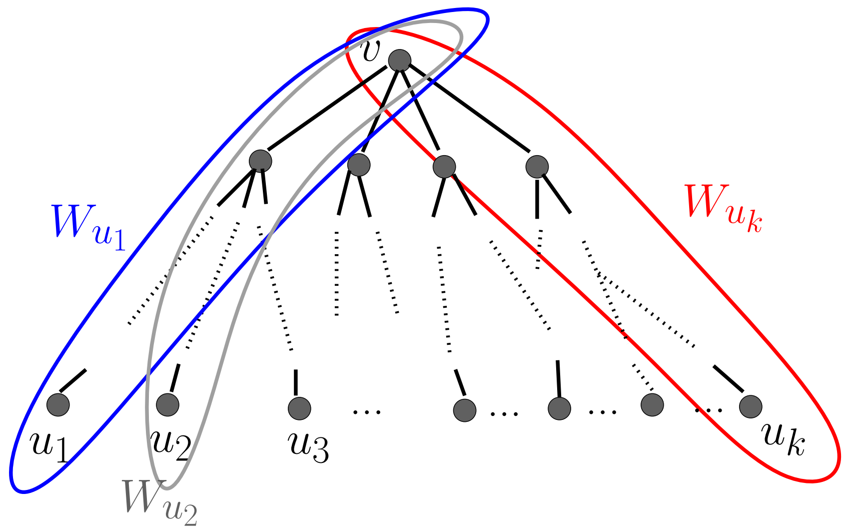
1. Do DFS from a node v.
2. If a path P from v has length at least ℓ, stop and output P.
3. Else, we have a DFS tree T of low depth. We can form a path decomposition $(P, \{W_{u_i}\})$ of width at most ℓ: For every leaf u let W_u contain u and its ancestors in T.



A Path-or-Decomposition Lemma

Given any graph G and any $\ell > 0$ there is a poly time algorithm Path-Decomp which either finds a path of length at least ℓ or a *path decomposition* of G of width at most ℓ.

1. Do DFS from a node v.
2. If a path P from v has length at least ℓ, stop and output P.
3. Else, we have a DFS tree T of low depth. We can form a path decomposition $(P, \{W_{u_i}\})$ of width at most ℓ: For every leaf u let W_u contain u and its ancestors in T.



$P = \{(u_1, u_2), \ldots, (u_{k-1}, u_k)\}$ where $u_1, u_2, \ldots, u_k$ are the leaf nodes in an inorder traversal of T.

Hybrid Algorithm for Longest Path

Hybrid Algorithm for Longest Path

1. Run Path-Decomp algorithm on G and ℓ.

Hybrid Algorithm for Longest Path

1. Run Path-Decomp algorithm on G and ℓ.
2. If a path of length ℓ is found, return it.

Hybrid Algorithm for Longest Path

1. Run Path-Decomp algorithm on G and ℓ.
2. If a path of length ℓ is found, return it.
3. Otherwise the algorithm returns a path decomposition P of width at most ℓ.

   Run an algorithm for LONGEST PATH on graphs of bounded treewidth (based on dynamic programming) by Bodlaender, 1993 to get the longest path in $2^{O(\ell \log \ell)} n^{O(1)}$.

Minimum Bandwidth

Minimum Bandwidth

Problem: Given a graph G, give a permutation π on the vertices of G so that the maximum edge *stretch* $\max_{(i,j)\in E(G)} |\pi(i) - \pi(j)|$ is minimized.

Minimum Bandwidth

Problem: Given a graph G, give a permutation π on the vertices of G so that the maximum edge *stretch* $\max_{(i,j)\in E(G)} |\pi(i) - \pi(j)|$ is minimized.

Notoriously hard. Best approximation: $O(\log^3 n)$ by Krauthgamer et al., 2003, $O(\sqrt{\frac{n}{B}} \log n)$ by Blum et al., 1998 where B is the optimum bandwidth.

Minimum Bandwidth

Problem: Given a graph G, give a permutation π on the vertices of G so that the maximum edge *stretch* $\max_{(i,j)\in E(G)} |\pi(i) - \pi(j)|$ is minimized.

Notoriously hard. Best approximation: $O(\log^3 n)$ by Krauthgamer et al., 2003, $O(\sqrt{\frac{n}{B}} \log n)$ by Blum et al., 1998 where B is the optimum bandwidth.

Best Exact Algorithm: $\tilde{O}(10^n)$ by Feige and Killian, 2000.

Bandwidth Hybrid

For any unbounded constructible $\gamma(n)$, MINIMUM BANDWIDTH admits a hybrid algorithm which produces either

- a linear arrangement achieving the minimum bandwidth in $4^{n+o(n)}$ time, or
- an $O(\gamma(n) \log^2(n) \log \log n)$-approximation in polynomial time.

What affects Bandwidth?

What affects Bandwidth?

One factor: a low diameter subgraph.

Simple Fact. If G contains a subgraph H of diameter d, then the bandwidth of G is at least $(|H| - 1)/d$.

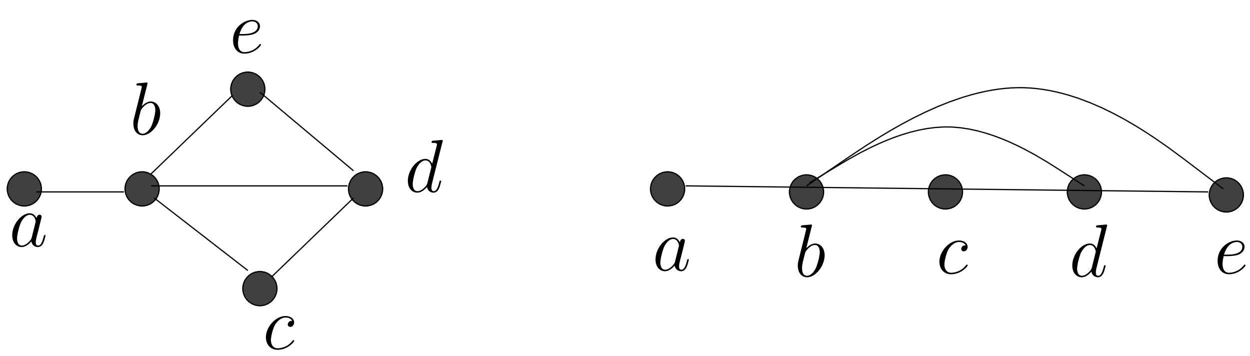
(Take the endpoints of any linear arrangement of H. They have a length d path between them...)

What affects Bandwidth?

One factor: a low diameter subgraph.

Simple Fact. If G contains a subgraph H of diameter d, then the bandwidth of G is at least $(|H|-1)/d$.

(Take the endpoints of any linear arrangement of H. They have a length d path between them...)



Bandwidth Hybrid

Bandwidth Hybrid

Idea: Attempt to find a “large” subgraph H with low diameter.

If you fail, output a “small” separator.

In the first case, can approximate bandwidth well.

In the second case, can find a separator tree and get a good exact algorithm for bandwidth.

Bandwidth Hybrid

Idea: Attempt to find a “large” subgraph H with low diameter.

If you fail, output a “small” separator.

In the first case, can approximate bandwidth well.

In the second case, can find a separator tree and get a good exact algorithm for bandwidth.

Intuitively, the absence of a large subgraph with low diameter means that the graph does not expand by much, so it has a smallish node bisection.

Conclusion

Conclusion

We introduced *hybrid algorithms*.

Conclusion

We introduced *hybrid algorithms*.

We gave simple hybrid algorithms for

MAX-CUT, LONGEST PATH, MINIMUM BANDWIDTH.

Conclusion

We introduced *hybrid algorithms*.

We gave simple hybrid algorithms for

MAX-CUT, LONGEST PATH, MINIMUM BANDWIDTH.

We also have hybrids for

COUNTING 2-CNF SOLUTIONS, MAX-Ek-LIN-p.

Conclusion

We introduced *hybrid algorithms*.

We gave simple hybrid algorithms for

MAX-CUT, LONGEST PATH, MINIMUM BANDWIDTH.

We also have hybrids for

COUNTING 2-CNF SOLUTIONS, MAX-Ek-LIN-p.

Overarching Idea:

Beat the inadequacies of worst-case analysis on a fixed complexity measure, by *choosing* which measure to beat on each instance.

Conclusion

We introduced *hybrid algorithms*.

We gave simple hybrid algorithms for

MAX-CUT, LONGEST PATH, MINIMUM BANDWIDTH.

We also have hybrids for

COUNTING 2-CNF SOLUTIONS, MAX-Ek-LIN-p.

Overarching Idea:

Beat the inadequacies of worst-case analysis on a fixed complexity measure, by *choosing* which measure to beat on each instance.

Two interesting problems arise in designing a hybrid algorithm for some Π

- How to split the cases of Π?
- How to select the right heuristic?

Thank You!