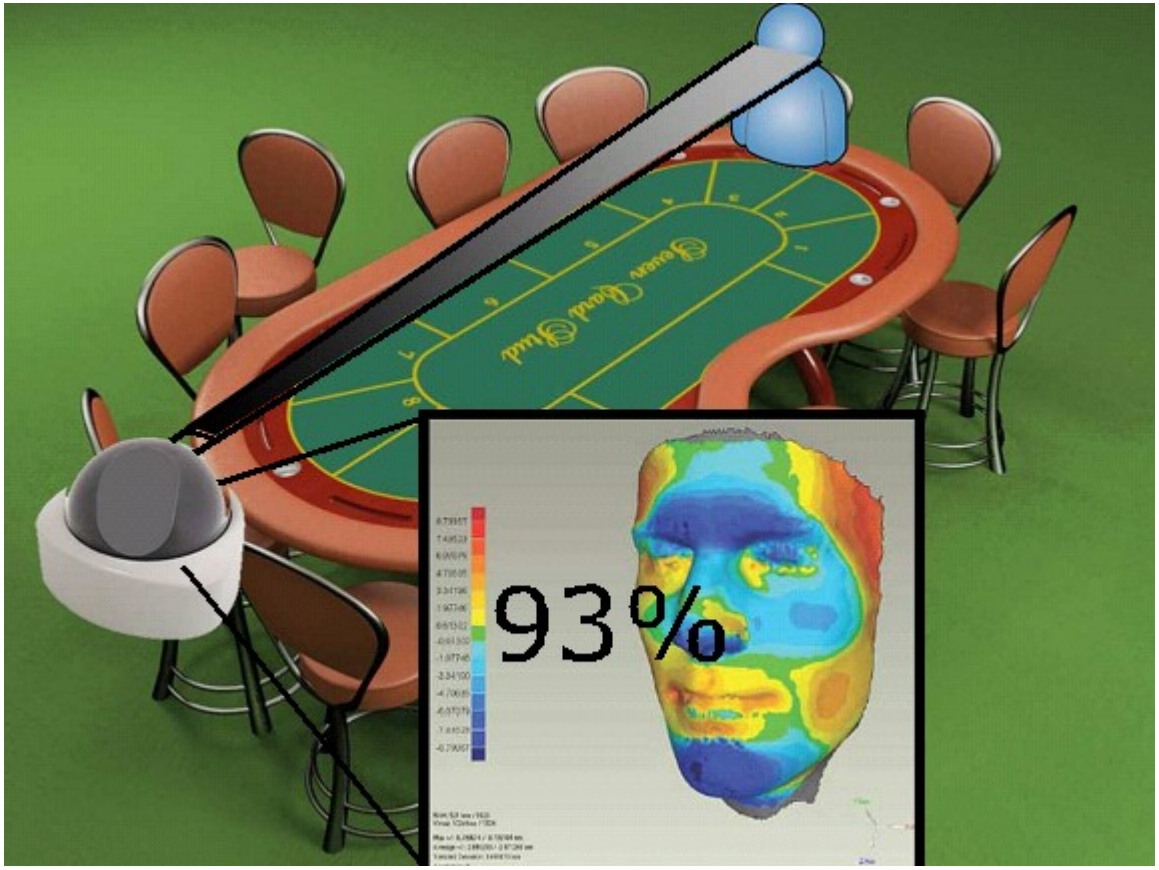
Filtered Cartesian Flattening

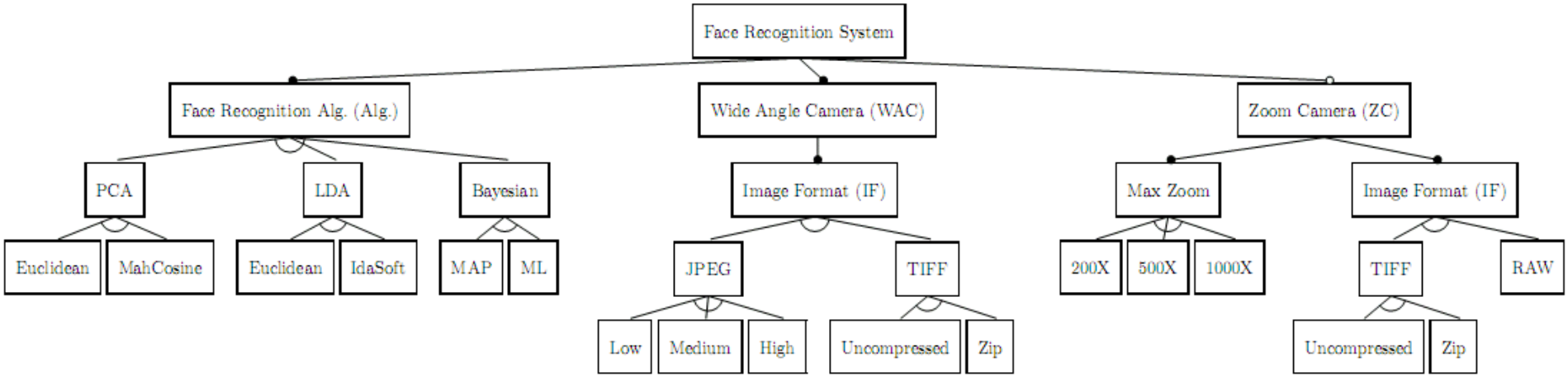
Jules White, Brian Doughtery, and Douglas C. Schmidt
Distributed Object Computing (DOC) Group
Institute for Software Integrated Systems
Vanderbilt University

Motivating Example



- **Context:** Facial recognition system identifies known cheaters in a casino. The system can be configured with different cameras, facial recognition algorithms, image resolutions, etc.

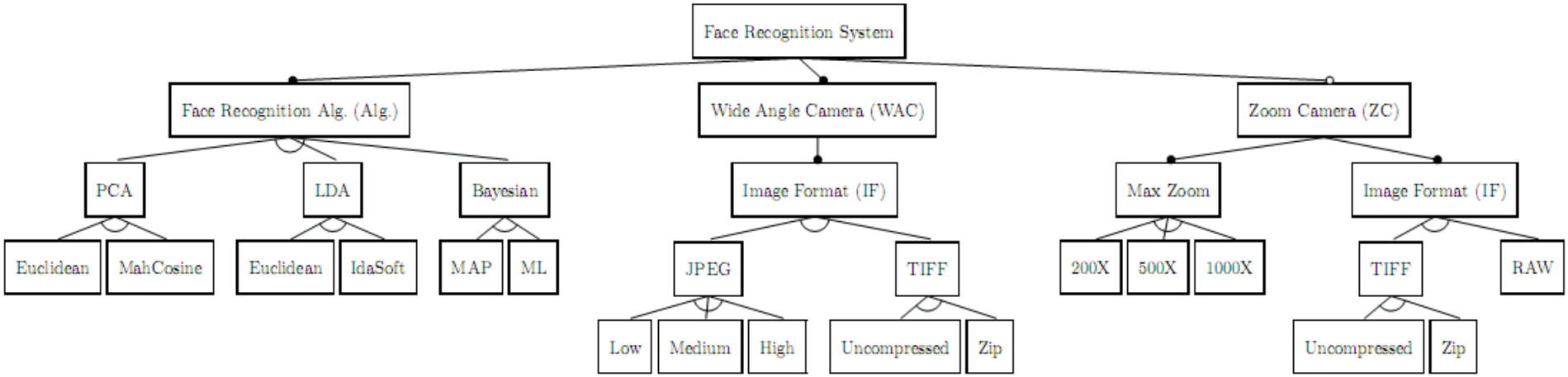
Motivating Example



- **Context:** If we use a more precise image processing algorithm, it utilizes more RAM and CPU cycles. Higher resolution images also improve accuracy but further increase the RAM and CPU load.

- **Problem:** What if we want to optimally select features to maximize accuracy without exceeding the resource constraints of a target platform? This can be reduced to the knapsack problem.

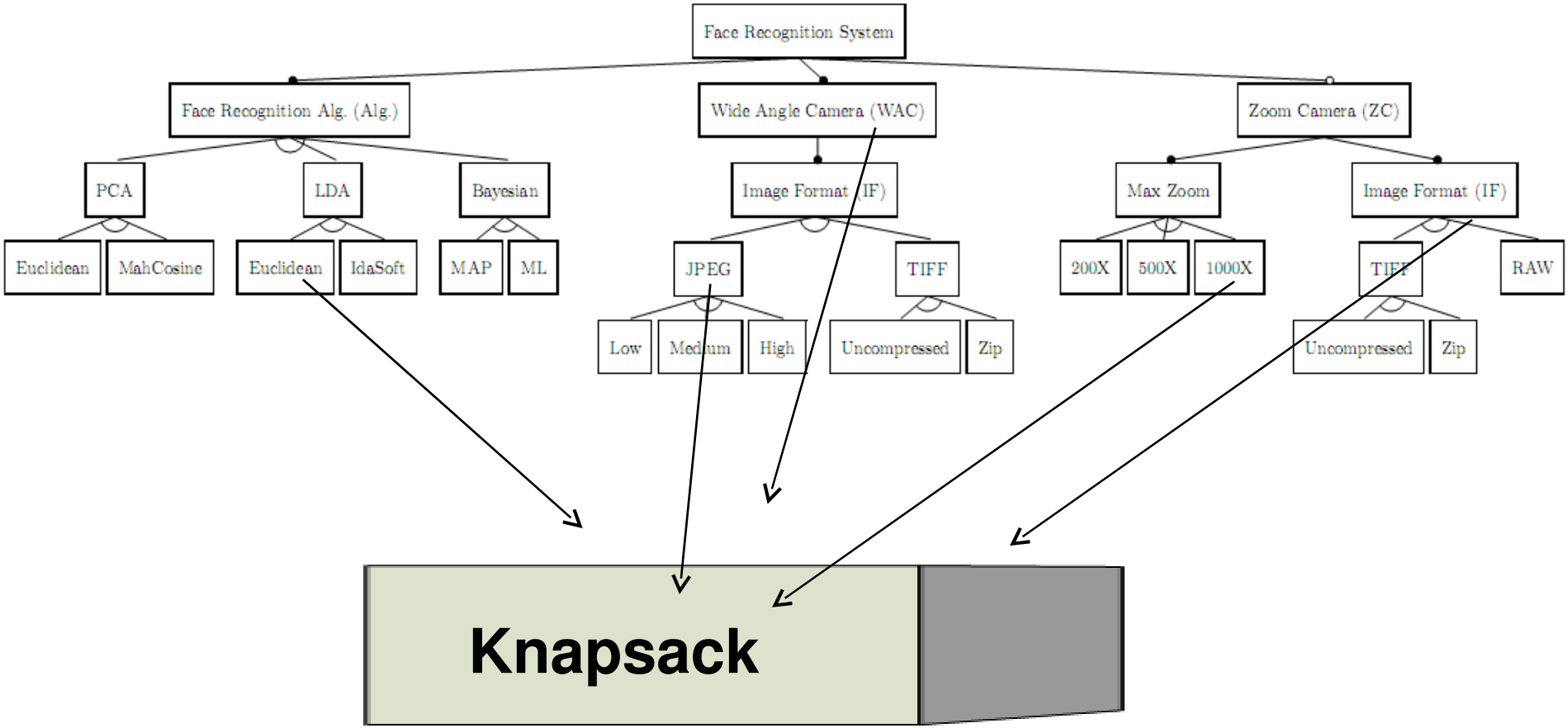
Motivating Example



- **For some problems, a CSP solver works fine….**
- **Generally, for big problems, CSP techniques do not**

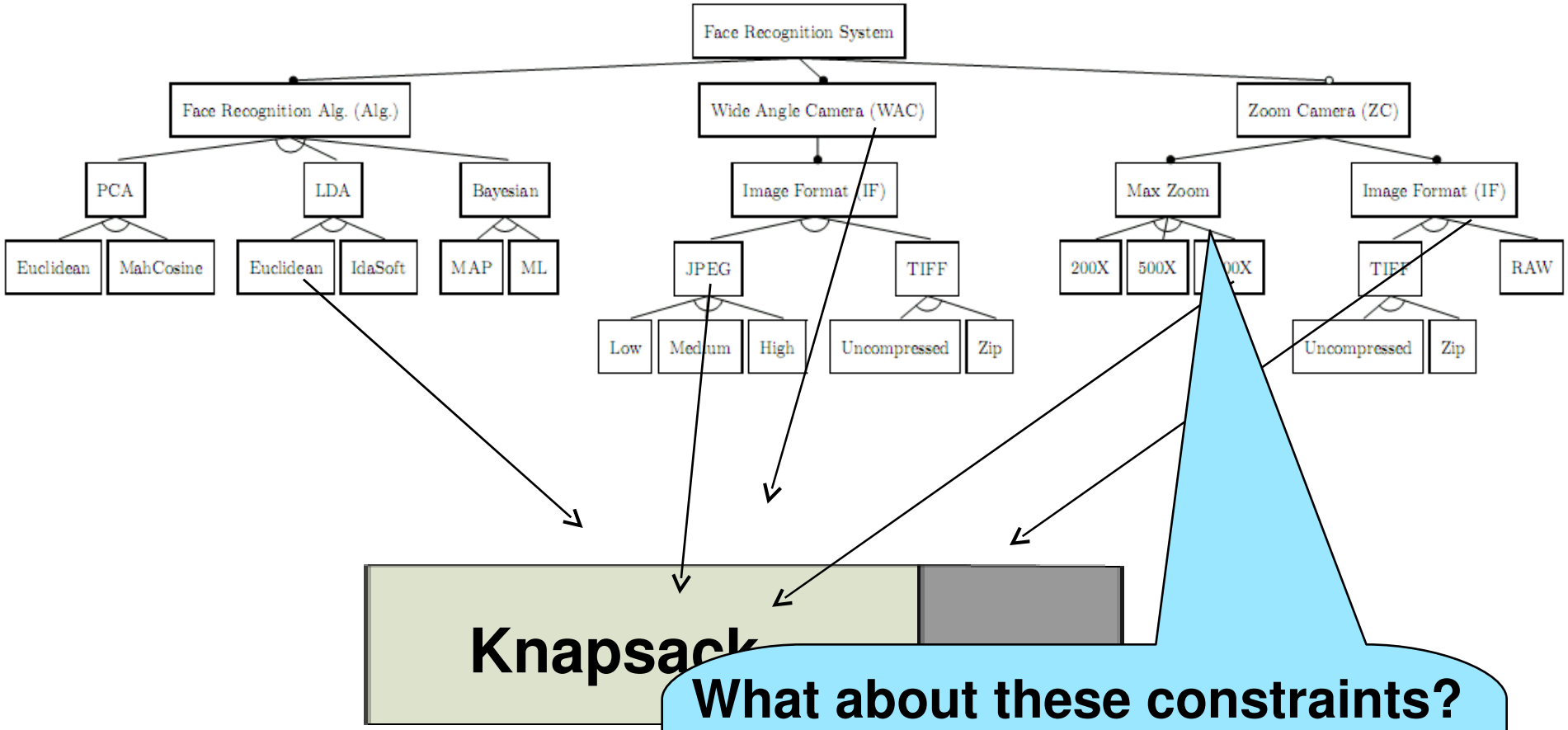
- **We wanted a technique to handle the cases where an exact exponential technique just wouldn't work**
- **We wanted to avoid inventing our own solver**

Knapsack?



- Can we just turn this into a knapsack problem?

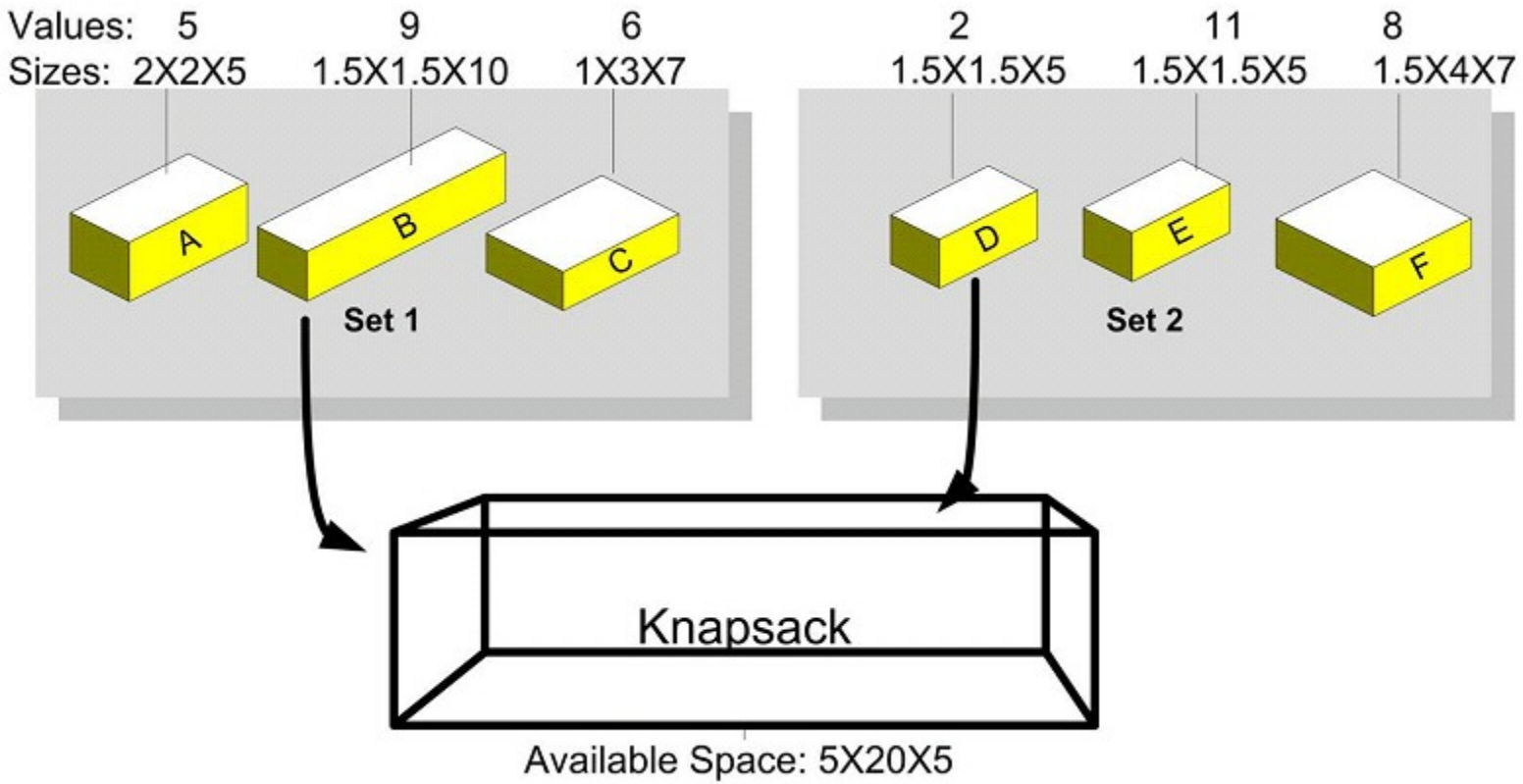
Knapsack?



- Can we just turn this into a kn

What about these constraints? We can't arbitrarily choose feature sets that fit into the knapsack.

MMKP

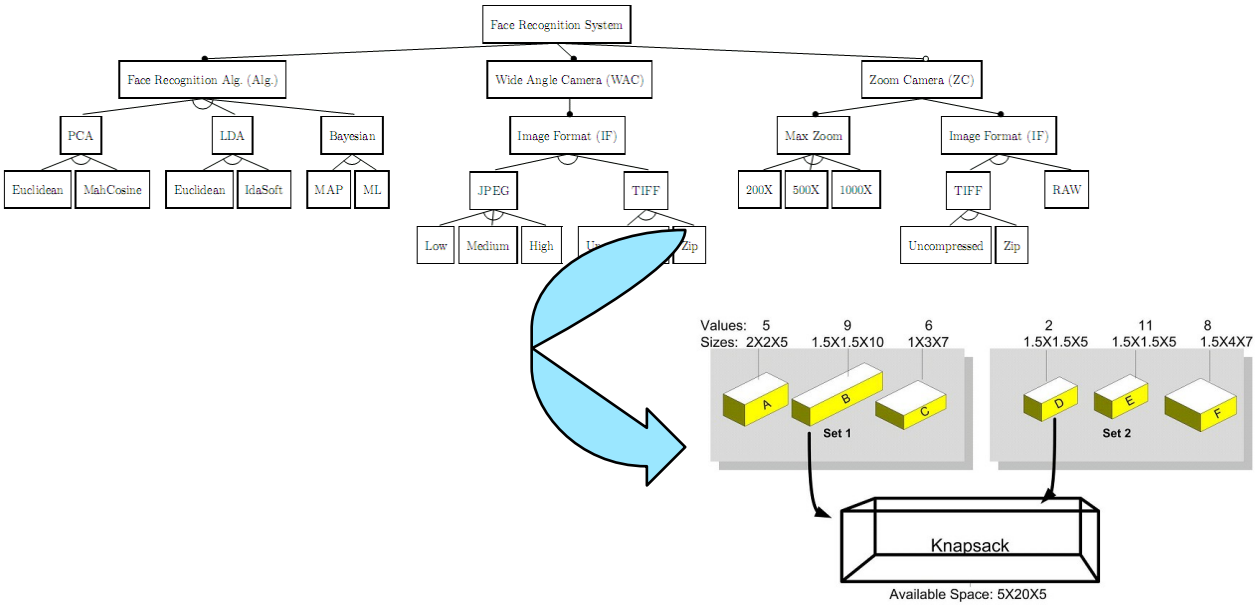


- Multidimensional Multiple-choice Knapsack Problem (MMKP)
  - Divides the items into sets and requires exactly one item to be chosen from each set.

Why MMKP?

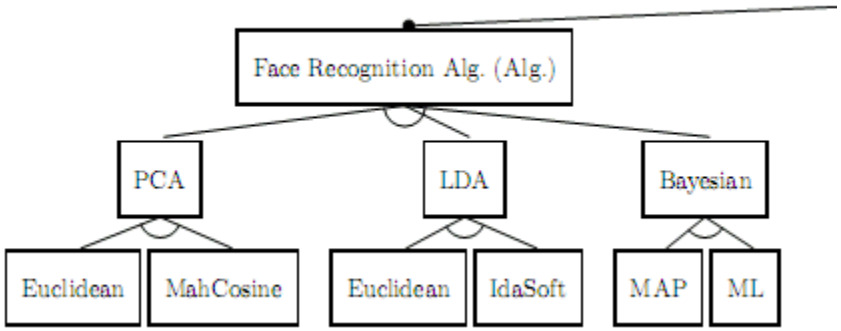
- Why do we care about MMKP?
- Knapsack doesn't allow us to constraint the selection of features but MMKP gives us some level of constraints
- These problems are NP-Hard, we don't want to reinvent the wheel
- There are excellent polynomial-time approximation algorithms with near optimal results for MMKP problems
  - M-HEU, one of the best MMKP algorithms, averages 98% optimal solutions for more than 10 sets
  - The bigger the problem gets, the more optimal M-HEU's answers are
    - See Heuristic Solutions to the MMKP Problem, Akbar et al.

Converting an FM to MMKP Sets/Items



- **We create a series of MMKP sets, such that**
  - **Each of the items that can go in knapsack represent partial configurations of parts of the feature model**
  - **The partial configurations are divided into sets so that choosing one configuration from each set results in a complete and valid configuration**

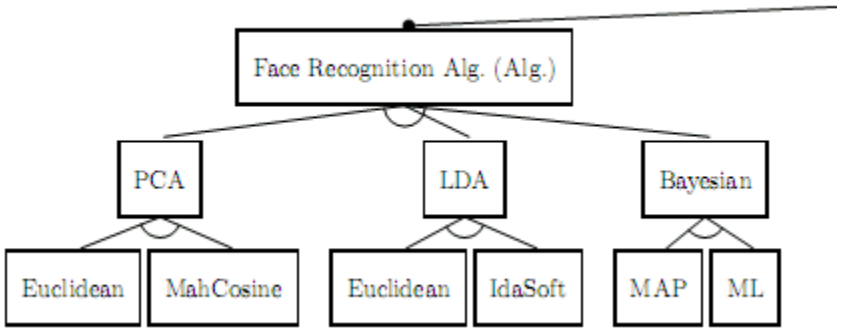
Flattening/Filtering (Compilation)



{
(Euclidean,PCA,Face..),
(MahCosine,PCA,Face..),
(IdaSoft,LDA,Face..),
….
}

- **We create the items and sets by enumerating the possible configurations of subtrees**
- **A state explosion occurs as we flatten the subtrees by enumerating the configurations**
- **We filter the configurations to avoid the state explosion**

Flattening/Filtering (Compilation)



{
(Euclidean,PCA,Face..),
(MahCosine,PCA,Face..),
(IdaSoft,LDA,Face..),
….
}

- **Filtering:**
  - **Take the M best possible configurations**
  - **Randomly choose M possible configurations**
  - **Try to select M possible configurations evenly distributed across the range of values**
  - **Etc….**

Experimental Design

- We generated random MMKPs that we knew the exact optimal answer
  - Our problem generation technique is described in: http://www.dre.vanderbilt.edu/~jules/white-ascent.pdf
  - Problems have a random number of higher and lower valued features than the optimal feature
  - A random number of features will have a better ratio of value/size than the optimal item
- We ran the FCF solver on large numbers (up to 450,000 problem instances) of these generated MMKPs and derived the average optimality
- Optimality measured as: FCF Answer / Optimal Answer = % Optimal

FCF Solving Time vs. CSP Solver

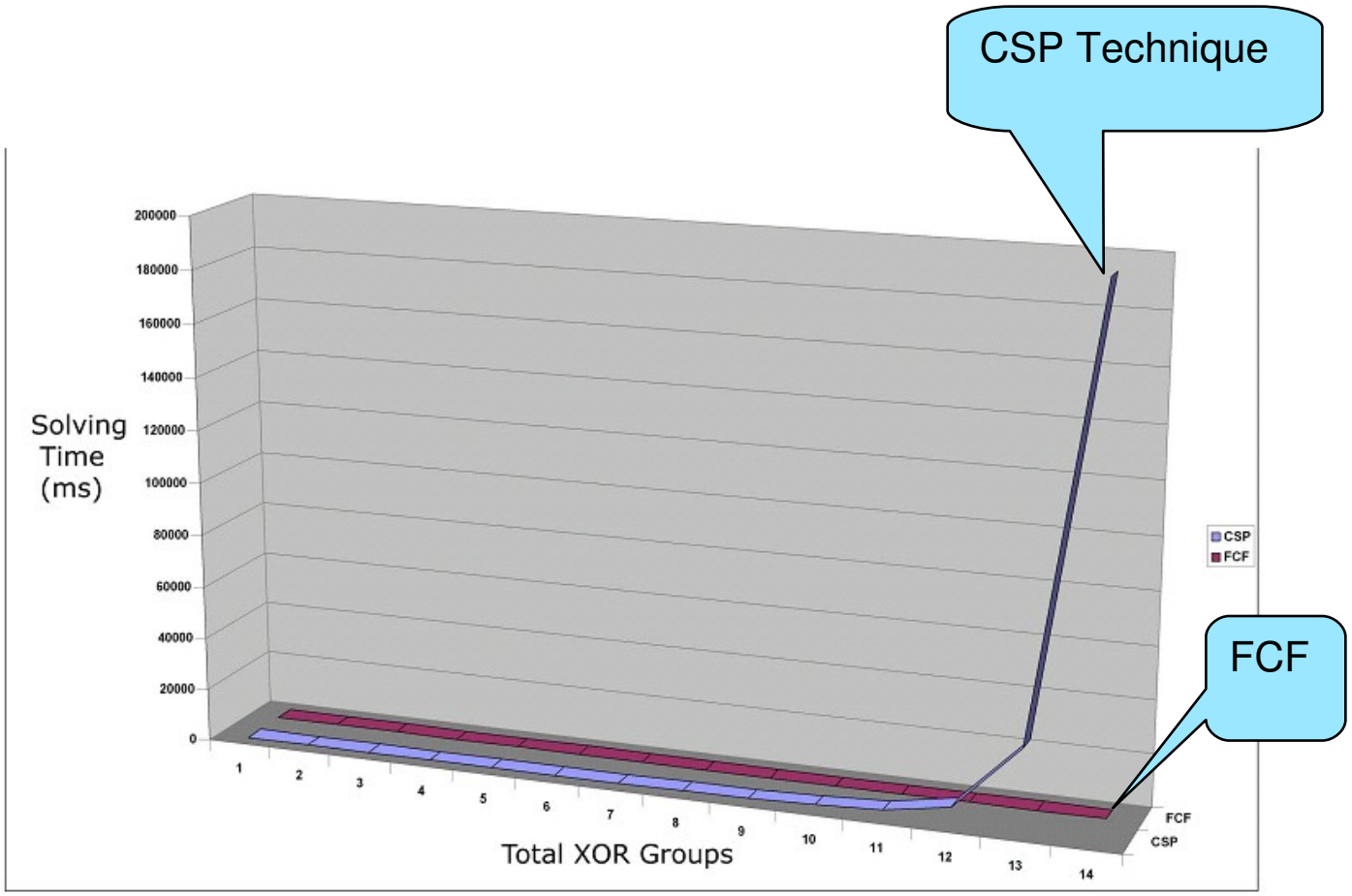


Fig. 17. Comparison of Filtered Cartesian Flattening and CSP-based Feature Selection Solving Times

Average Opt. for 5,000 Features

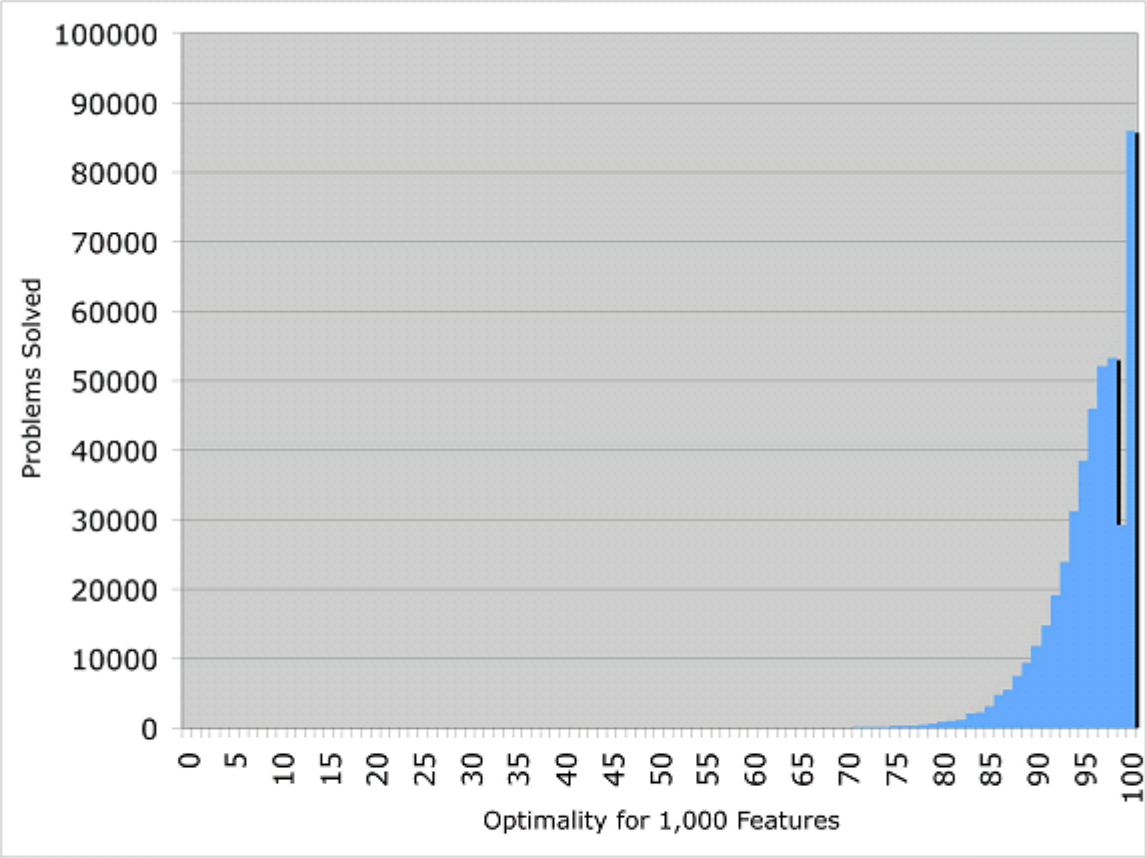


Fig. 19. A Histogram Showing the Number of Problems Solved with a Given Optimality from 450,000 Feature Models with 1,000 Features

Average Opt. for 10,000 Features

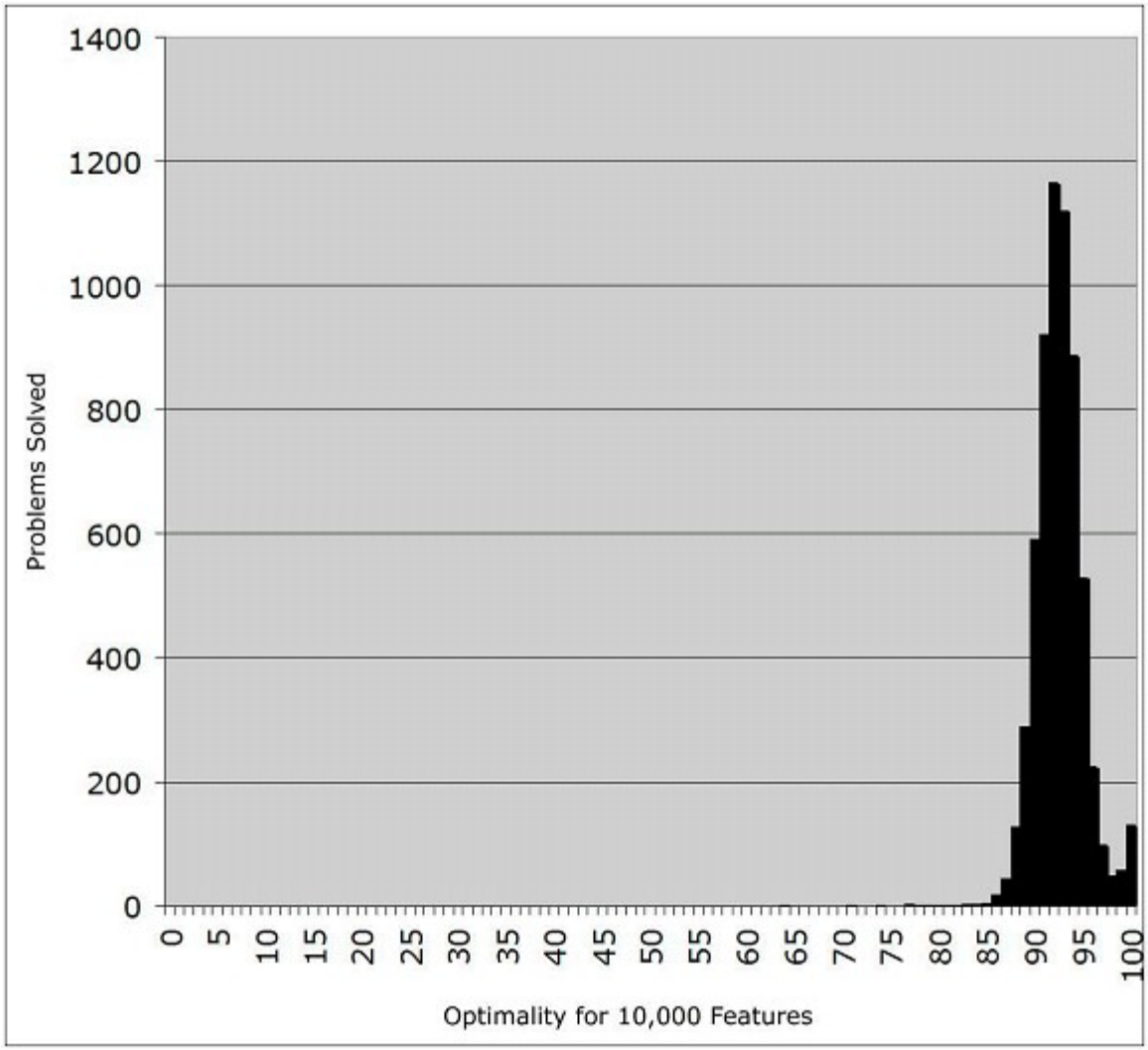


Fig. 20. A Histogram Showing the Number of Problems Solved with a Given Optimality from 8,000 Feature Models with 10,000 features

When Not to Use FCF

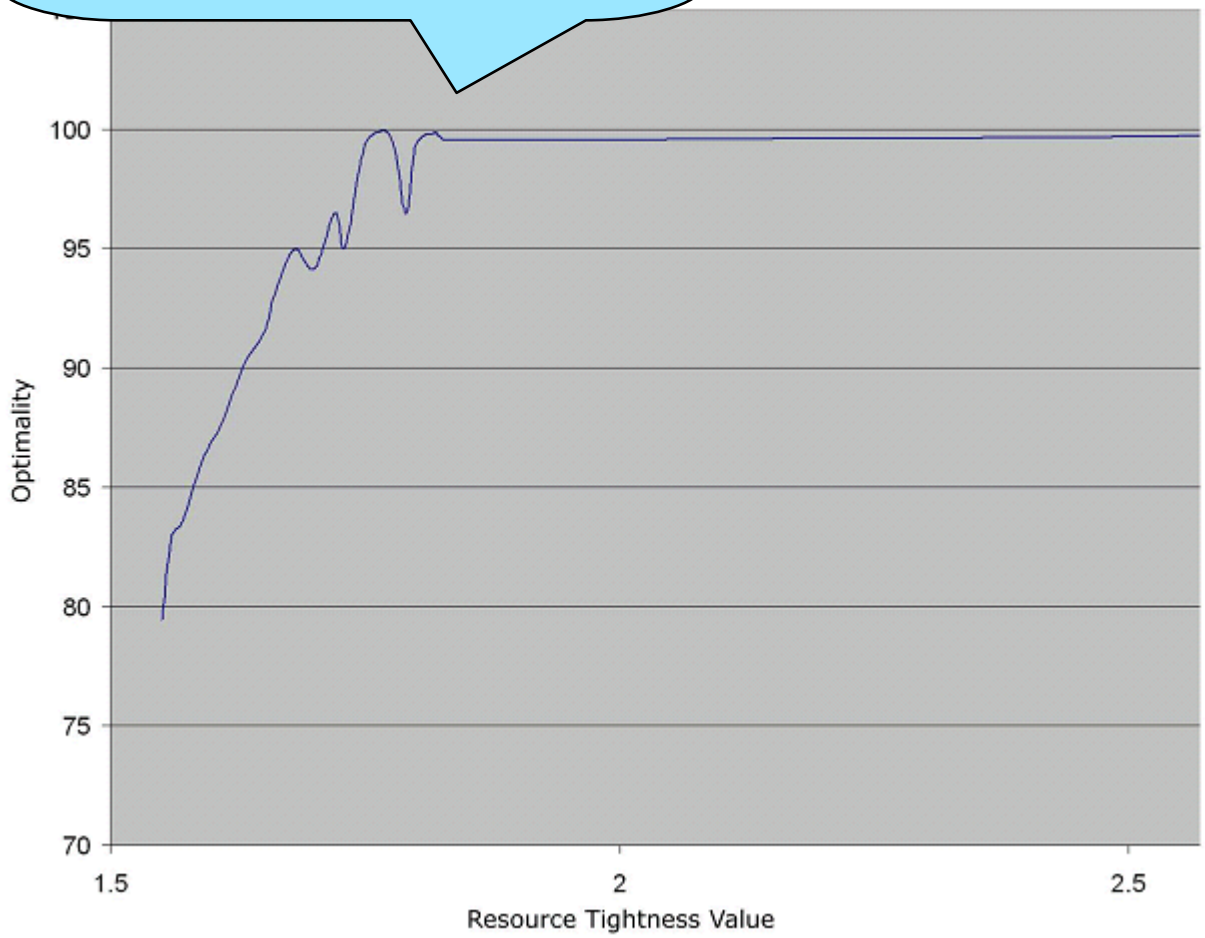
$$\frac{\sqrt{R_0^2 + \ldots R_m^2}}{\sqrt{(\sum_{i=0}^{n} r(i,0)^2 + \ldots r(i,m)^2)/n}}$$

- Resource tightness metric, where:
  - Ri is the amount of the ith resource available for consumption (e.g. how big the knapsack is in that direction)
  - r(i,j) is the amount of the jth resource consumed by the ith item

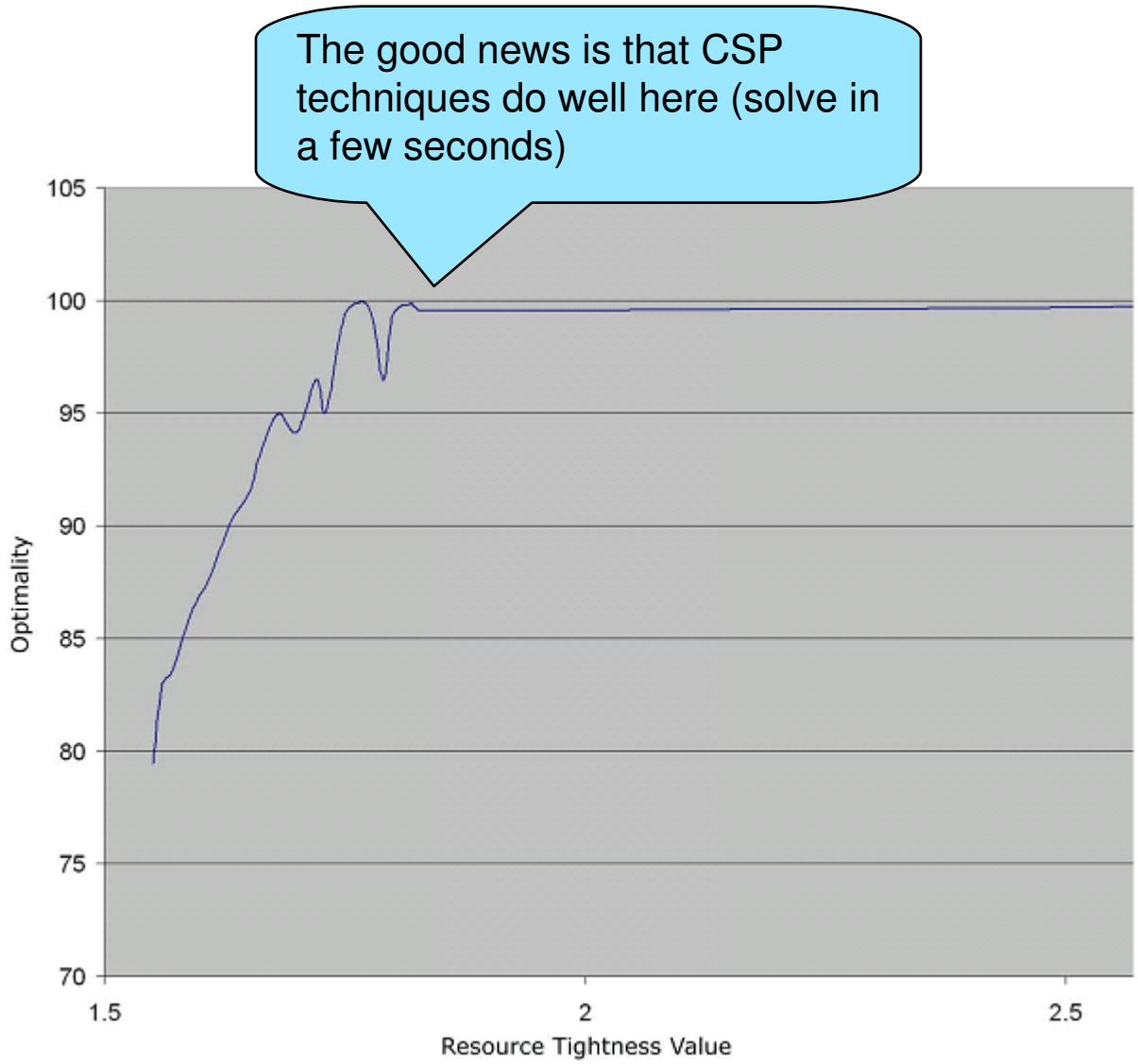
- This resource estimates how many of the average size items will fit into the knapsack
- If less of the average size items will fit, it means that there is a tighter fit between the items and the knapsack

Resource Tightness vs. Opt.

If less than ~1.6 of the average size items will fit into the knapsack, FCF performs poorly



Resource Tightness vs. Opt.



Summary of Results

- Solving time ~10s for 10,000 features
  - Orders of magnitude faster than a CSP approach
- Average optimality
  - ~95% for 5,000 features
  - ~93% for 10,000 features
  - > 99% solved with > 80% optimality
- Resource tightness is the key indicator of how FCF performs
  - Less available resources and more resource consumptive items produce poorer FCF solutions
  - Good news is that CSP techniques tend to solve these *tight* MMKPs quickly

Conclusion

- Throwing away possible configurations during the filtering does have an affect on optimality
- M-HEU normally averages 98% optimal for big problems and gets better the larger the problems get
- FCF slowly degrades in optimality with bigger problems
  - A larger percentage of the possible configurations are thrown away
  - FCF only degrades ~2% from 5,000 to 10,000 features
- FCF can solve extremely large feature selection problems with resource constraints very fast
- FCF averages a very high level of optimality