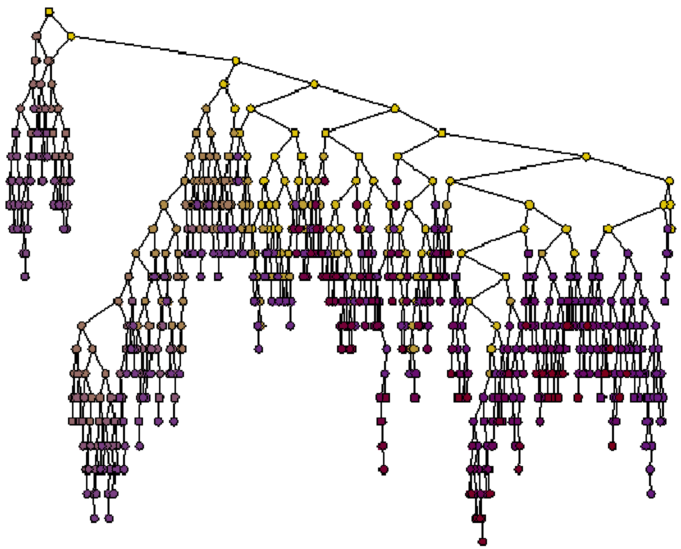
BRANCHstorming (brainstorming about tree search)

Matteo Fischetti, University of Padova



Tree search (the way we teach it)

- **Tree search** (or enumerative) methods evangelized in different ways by different communities

- According to the
  Integer Programming Gospel …

  In the beginning was the Fractional Point
  [John 1:1]

- Apocryphal Gospels however exist that even doubt the **existence** of the fractional point (popular in the **barbarian AI & CP worlds**…)

Role of the fractional point

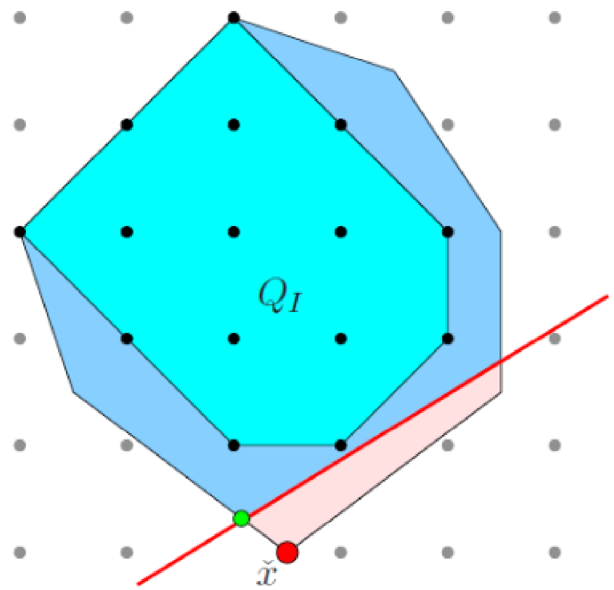
1. Solve the LP (or convex) relaxation of your (M)IP and let x* be an optimal solution

2. If x* is integer, jubilate!

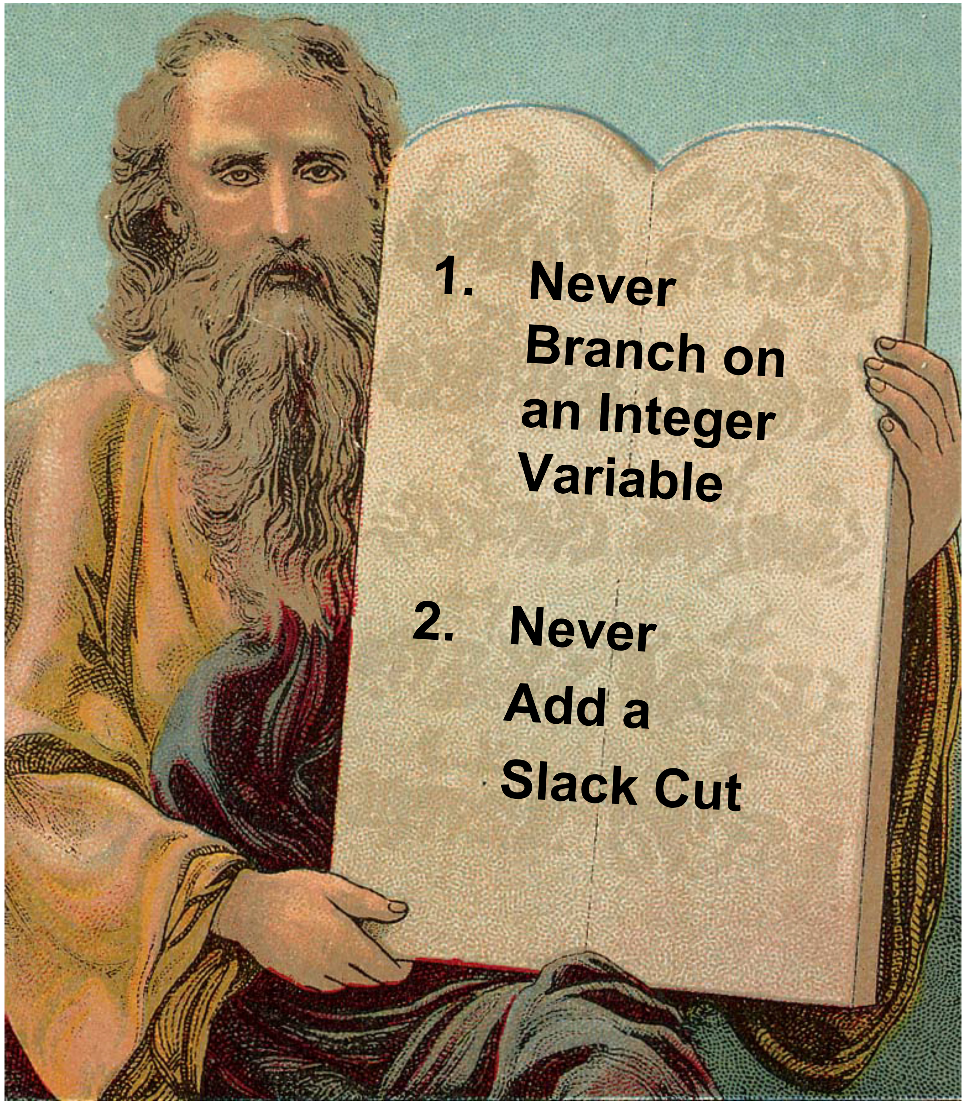
3. Otherwise, x* is the devil and you have to dispel it

4. Try with **cutting planes** first (the more violated by x* the better)

5. Then **branch** on a fractional component of x*



The IP Commandments



The IP verb is spread over the world

Success of IP tree-search paradigm

- The main ingredients of IP tree search deeply studied in the last years
  - Powerful preprocessing
  - Fast LP solvers
  - Better and better cutting planes
  - Improved branching strategies
  - Extensive propagation/probing
  - Improved primal heuristics
- As a result, more and more real-world difficult problems solved to proven optimality
- **Everything well understood and under control !(?)**

But… something strange happens

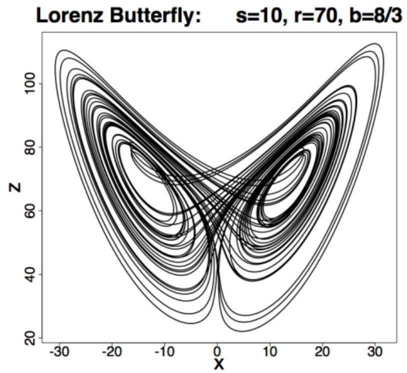
- Different IP solvers may have **very different performance** on a same instance!

	SOLVER #1		SOLVER #2		
	Time	Nodes	Time	Nodes	Time Speedup
glass4	43.08	118,151	12.95	17,725	3.33
neos-1451294	3,590.27	20,258	102.94	521	34.88
neos-1593097	149.94	10,879	16.12	508	9.30
neos-1595230	1,855.69	152,951	770.60	89,671	2.41
neos-603073	452.40	36,530	130.75	10,017	3.46
neos-911970	3,588.54	5,099,389	3.29	1,767	1,090.74
ran14x18_1	3,287.59	1.480,624	2.066.70	759,265	1.59

- **SOLVER #1: IBM ILOG Cplex 12.2 (default parameters)**
- **SOLVER #2: IBM ILOG Cplex 12.2 (default parameters)**

Deterministic runs on the same PC, only change is the initial random seed

Tree search as a chaotic system?



- Common observation (Danna, 2008): even when implemented in a **deterministic** way, tree search is **highly dependent on initial conditions**
  → small changes can result into completely different trees

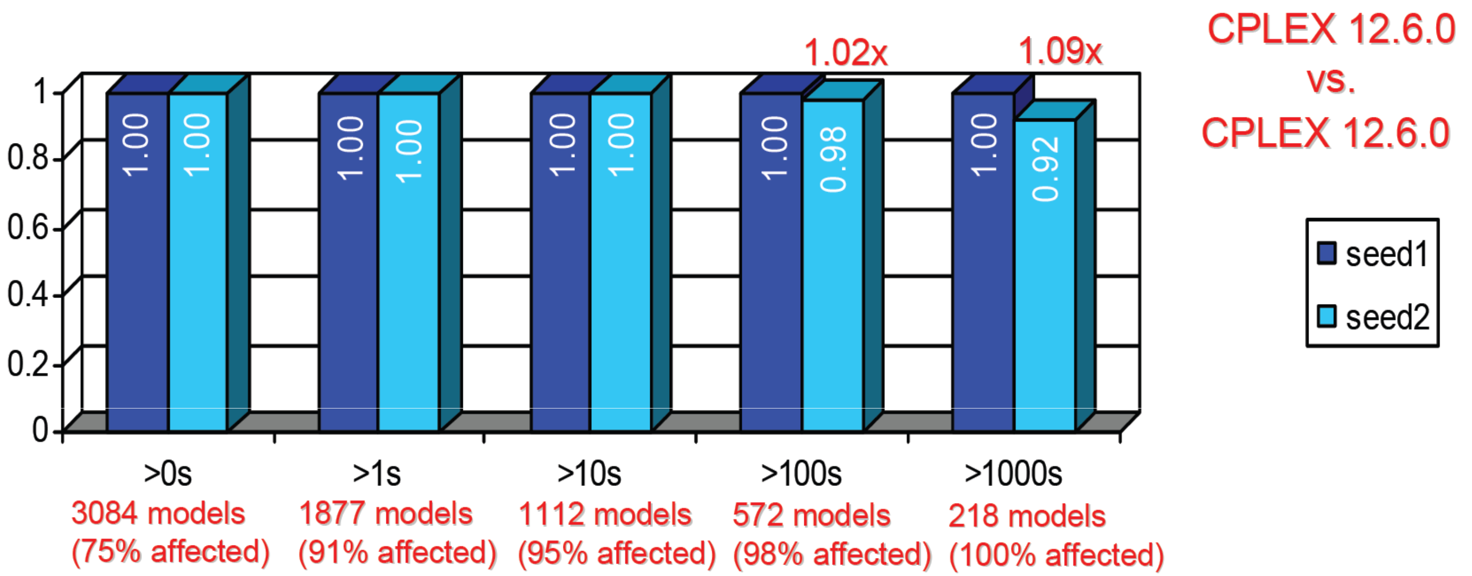
- Changes can be related to the **external environment** (same code compiled for different hardware or OS's) …
- … or to the internal **problem representation** (permutation of rows and col.s)
- … or to the **internal parameters** (initial random seed)

- In all cases, it is impossible to **predict** which initial condition will produce the best performance

- **The more sophisticated the code, the more variability is expected!**

Erratic performance variability

Performance Variability: does it matter?

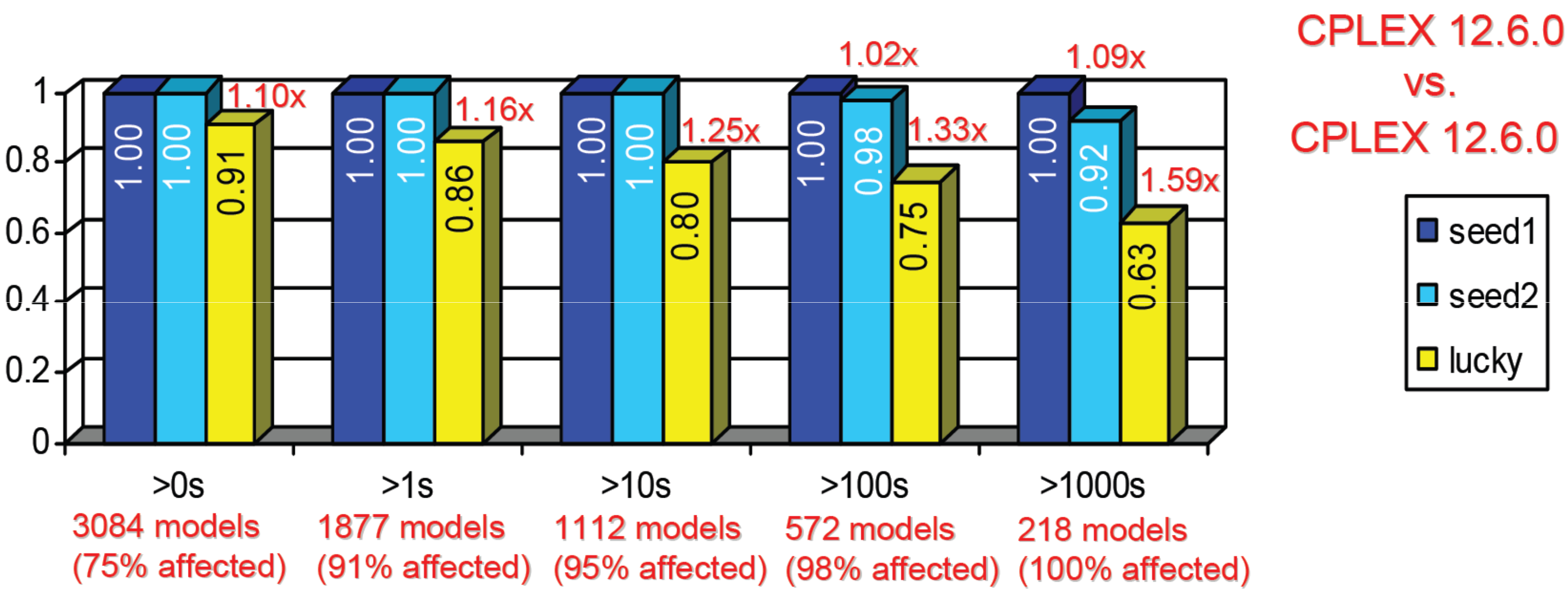


- 91% of the models in [1,10k] affected:
  the random seed is effective to simulate performance variability.
- More than 200 models in [1k,10k] can not measure performance difference of less than 10%:
  - Small test sets are less robust to outliers,
  - Harder models are typically the ones exhibiting the larger variability.

(courtesy of Andrea Tramontani, IBM ILOG Cplex)

Erratic performance variability

Performance Variability: good news from the lucky solver



- **Lucky solver**: for each model, take the best time among the competitors.

(courtesy of Andrea Tramontani, IBM ILOG Cplex)

Variability as an opportunity

- F. and Monaci (Op. Res. 2014): **bet-and-run**
    1. Run CPLEX *k* times with different seeds, for just few B&C nodes
    2. bet on the winner and let it run up to completion

- Carvajal, Ahmed, Nemhauser, Furman, Goel, Shao (Opt. Online, 2013):
    - run *k* single-thread B&C with different **parameters** (instead of single B&C with *k* threads)

- F., Lodi, Monaci, Salvagnin, Tramontani (submitted)
    - **Concurrent root cut loops**; in the Cplex default since version 12.5

- Powerful way to distribute B&C computation on a cluster of PCs
  → **Distributed Concurrent Optimization**

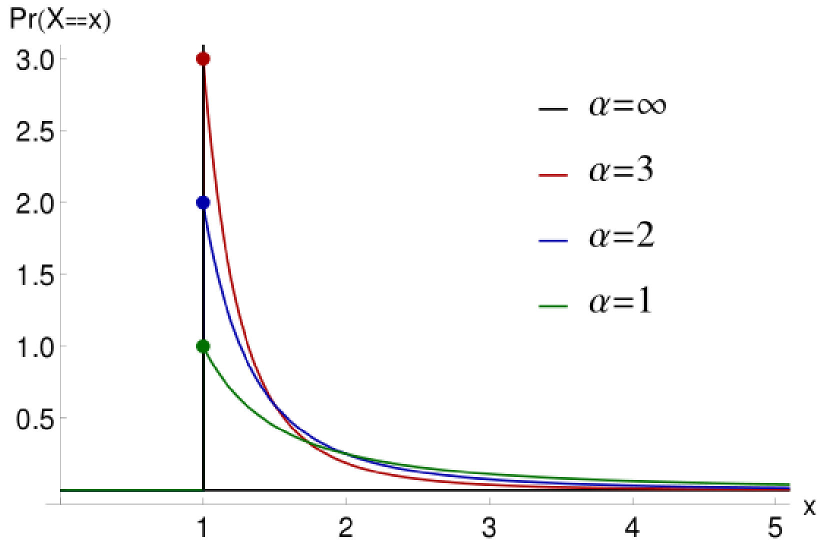
Variability as an issue

- High performance variability helps when **a same instance** is solved in parallel → computation ends when the **FIRST** solver ends its job

- High performance variability is **very bad** when different parts of the instance (e.g., subtrees) are solved in **parallel** → computation ends when the **LAST** part is solved

- **Number of nodes in a subtree as a random variable**

  Heavy tailed distributions → there is a <u>small</u> but nonzero probability that the n. of tree nodes explodes!

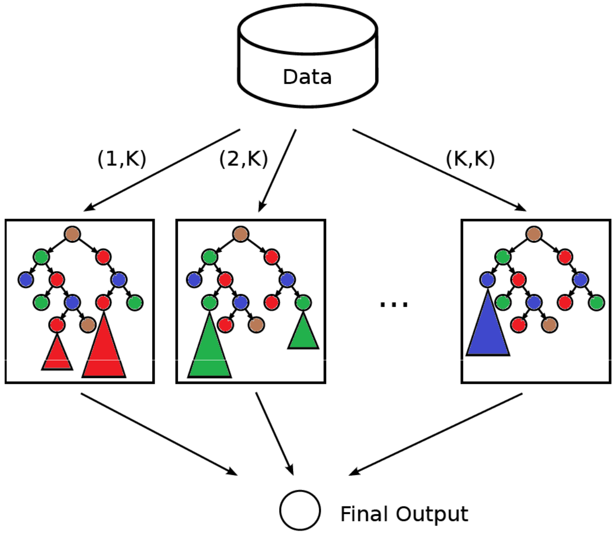


SelfSplit for tree search parallelization

- A new framework recently proposed by F., Monaci and Salvagnin (2013)

- Super-easy way to convert a **sequential tree-search code** into a **parallel** one

- Each worker reads the original input data and receives an **additional input** pair *(k,K),* where *K* is the total number of workers and *k=1,…,K* identifies the current worker



- The same deterministic **sequential** computation is initially performed by all workers (**sampling phase**), without any communication

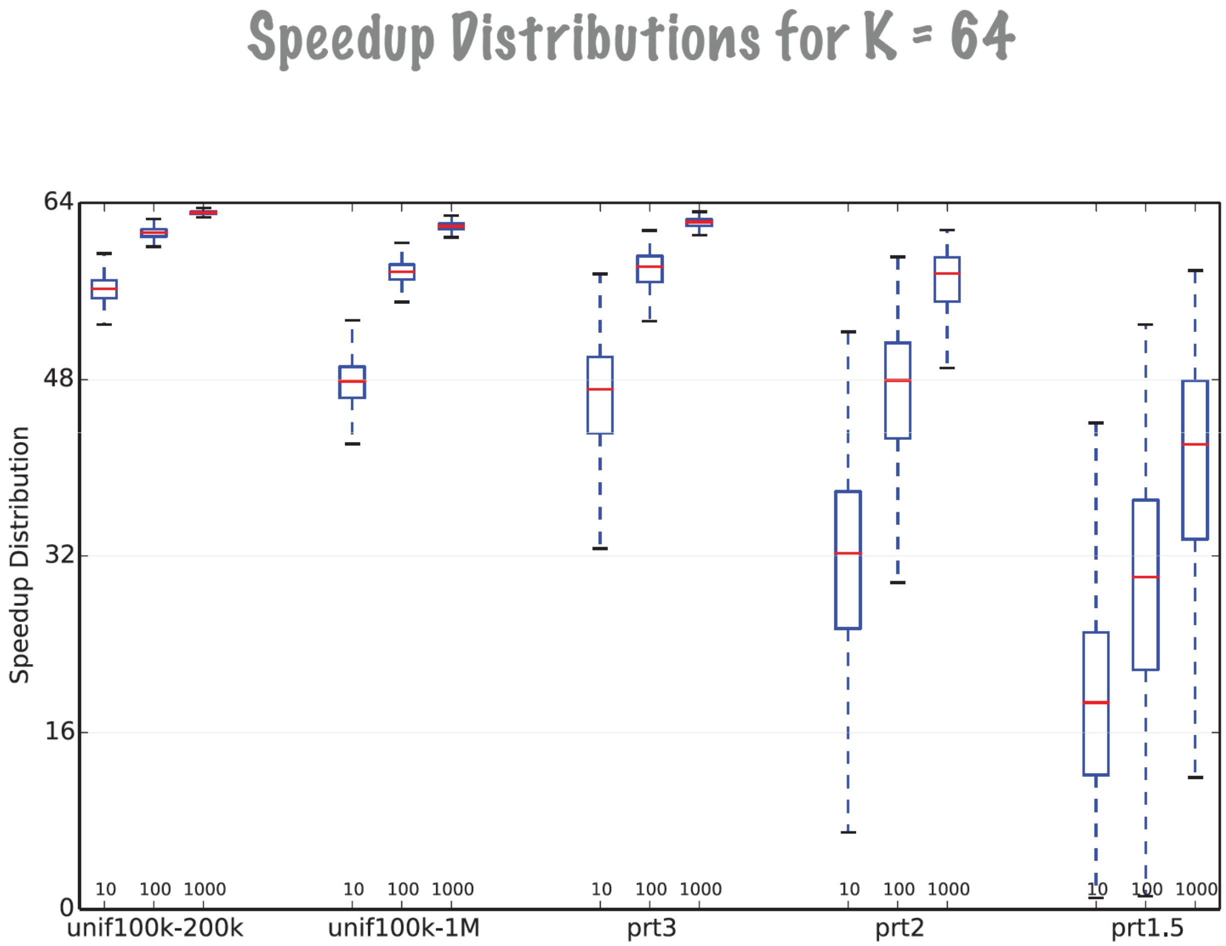
- When enough open nodes have been generated, each worker applies a **deterministic rule** to identify and **skip the nodes that belong to the other workers**, with no (or very little) communication among workers.

Role of variability in workload split

- Synthetic experiments with 10, 100, 1000 random subtrees per worker (subtree size as a random variable)

unif = uniform

prt = Pareto heavy t.



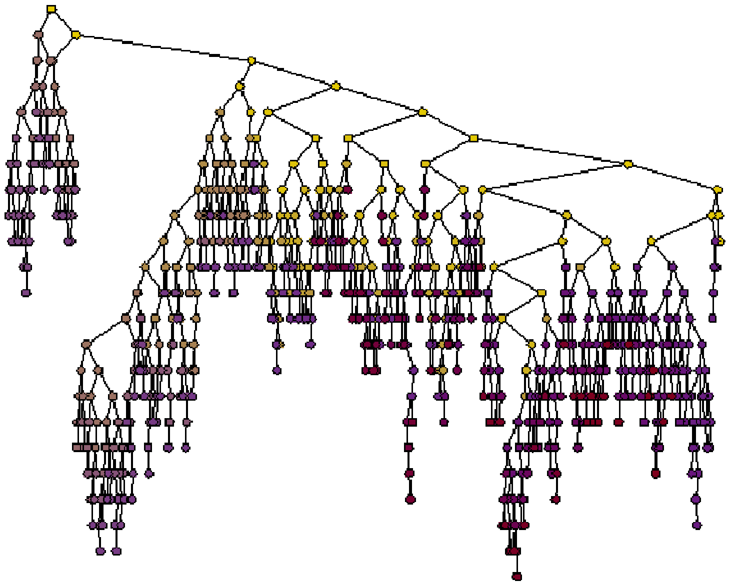
A computational conjecture

- **Recursive nature of tree search**

  → overall tree is a collection of subtrees

  → the overall tree-search performace
  is averaged over subtrees

  → but still there is a large probability that
  some subtrees require a vary large computing time just because of erraticity…



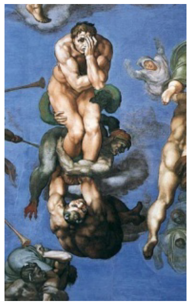
- **Computational conjecture**: reducing variability inside the tree can help a lot even a sequential code as "no subtrees explode"

Where does erraticity come from?

- A main source of erraticism in our branch-and-cut (or branch-and-bound) codes is the emphasis we give to the **fractional solution**

- Indeed, even if we believe we are good fellows who respect the IP commandments…

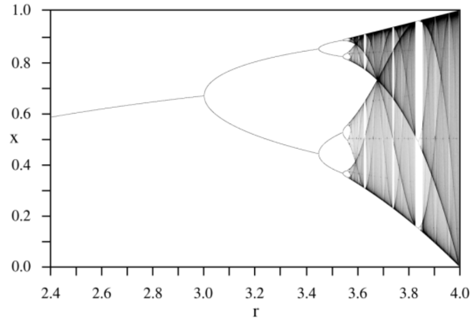
- … we still commit the **original sin** of being driven by the **fractional point**
  … and we insist on **branching on integer variables** and on **adding slack cuts**



The reason is that we are **truly degenerate** (not in the sense of being immoral, but because of the existence of equivalent optimal fractional solutions)

Bifurcation points and simplex method

- Fractional points are typically computed by the **simplex method**
- The simplex method follows a **path along the edges** of the LP polyhedron
- Degeneracy triggers a **random perturbation** in the simplex method → bifurcation point in the simplex search paths

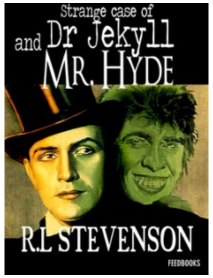


- Any small change (even the random seed) acting at the bifurcation point will produce a **completely different** final solution on the optimal face
- Different fractional solutions lead to **different cuts** and **heuristics** and **branching** at the root node
- Branching itself acts as a **exponential chaos amplifier** → the pinball effect

Don’t trust the fractional point!

- Dual degeneracy is a **structural property** of the LP relaxation of NP-hard problems -- if we could exclude dual degeneracy at every step, we could solve any IP in pseudo-polynomial time by Gomory’s integer cutting planes

- So, at each B&B node the fractional solution we consider is by no mean **THE fractional solution**
- **…** but **just a random sample** among **millions alternatives**



- **…** possibly **biased** because of the algorithm used to select it (e.g., dual simplex favors bases not too far from the previous one, hence inducing a potentially-dangerous correlation)

- Whenever LP bound does not improve after a cut or after branching, we are in fact **adding a nonviolated cut**, or we are **branching on an integer variable**, w.r.t. a different (equivalent) fractional solution!

Brainstorming

- IP tree search designed around with concept of **fractional point**
- But we have seen that **THE** fractional point does not exist…
  … as what we get at each node is just a **random (biased) sample**
- **Is it reasonable to take strategic decisions (notably: cutting planes & branching) based on the analysis of a single fractional solution?**
- Our B&C codes are likely to suffer from large **overfitting**
- Like designing a machine learning tool (say, a Support Vector Machine) with a training set composed by a single point → would you trust it?
- **Research topic:** new generation of B&C codes where **clouds of fractional solutions** are evaluated in a statistically-sound way (bigdata approach?)

Standard branching rules

 **Naïve branching:** based on the value of the fractional components of the single fractional point at hand (e.g., closest to 0.5 or alike)

 **Pseudo-cost branching:** based on several fractional points discovered in previous iterations

 **Strong branching:** based on a global property (lower bound) independent of the particular fractional point at hand – note however that in case no bound increase can be obtained in both son nodes, there is no reason not to branch on a variable that happens to be integer in the single fractional point at hand

 **Propagation effects on binding constraints** (Patel and Chinneck 2007, Pryor and Chinneck 2011).

 **Restart to favor noogoods** (Karzan, Nemhauser and Savelsbergh, 2009)

Backdoor branching

- Proposed by F. and Monaci (2013)
- 0-1 MIP $z^* := \min\{c^T x : Ax \leq b,\ x_j \in \{0,1\}\, \forall j \in J\}$

$$L(S) := \min\{c^T x : Ax \leq b,\ x_j \in \{0,1\}\, \forall j \in S\}$$

- **Backdoor** (Dilkina,Gomes and Malitsky, 2009) $S \subseteq J : L(S) \geq z^*$
- Set-covering model for smallest backdoor (P = LP relaxation)

$$\min \sum_{j \in \mathcal{I}} y_j$$

$$\sum_{j \in frac(\tilde{x})} y_j \geq 1,\ \forall \text{ vertex } \tilde{x} \text{ of } P : c^T \tilde{x} < z^*$$

$$y_j \in \{0,1\}\ \forall j \in J$$

- **Backdoor branching** (basic idea): assume z^* known
  1. **Sampling phase**: collect a large number of vertices $\tilde{x} \in P : c^T \tilde{x} < z^*$
  2. Solve the set covering model on those vertices to get “backdoor” S^*
  3. Give a large branching priority to the variables in S^*

Cloud branching

- Proposed by Berthold and Salvagnin (2012)

- **Idea**: work with a **cloud** $C = \{x^1, \cdots, x^k\}$ of equivalent optimal LP sol.s.
  The cloud can be computed with **reasonable overhead** (fixed number of simplex pivots on the optimal face)

- Use the cloud to define $F(C) = \{j \in J \mid \exists x^i \in C : \; x^i_j \notin \mathbb{Z}\}$

$$l_j = \min\{x^i_j \mid x^i \in C\}$$
$$u_j = \max\{x^i_j \mid x^i \in C\}$$

$\rightarrow \quad F_2 = \{j \in F(C) \mid \lfloor x^\star_j \rfloor < l_j \; \wedge \; u_j < \lceil x^\star_j \rceil\}$

- For a binary problem, F_2 contains the var.s that are fractional $\forall x^i \in C$

- During **strong branching**, try var.s in F_2 first (faster & more robust)

Branching on cuts?

- Most IP solvers branch on a **single variable** $x_j \leq t \quad \text{or} \quad x_j \geq t+1$
- Why not branching on a linear disjunction? $a^T x \leq t \quad \text{or} \quad a^T x \geq t+1$
- Attractive because of the **larger degree of freedom** in the choice
- Encouraging computational results from the literature
- However, not implemented in commercial solvers yet: why?
- A main difficulty is related precisely to the **increased degree of freedom** → much increased **variance** in estimating the best disjunction → much larger **overfitting** → **requires a statistically-sound way to select the disjunction**

Cut filtering

Quality measures typically used to filter the active cuts at root node

- Cut **violation** and/or **distance cut off** w.r.t. to a single fractional point

- Being violated also by a ‘nearby point’ → Cplex’s **pumpreduce** strategy (T. Achterberg)

- Nonzero dual variable (**activity**) at the final root node LP + **density** (Cornuéjols, Margot, Nannicini, and Tjandraatmadja, 2013)

Computational conjecture: perhaps many classes of “potentially strong” cuts are under-utilized within B&C just because a **statistically-sound way to select them is missing** (the more degrees of freedom, the more overtuning and hence the worse practical results)

Thanks for your attention

SelfSplit paper available at
www.dei.unipd.it/~fisch/papers

slides (also of this talk) available at
www.dei.unipd.it/~fisch/papers/slides