A Fast Heuristic for GO and MINLP

John W. Chinneck, M. Shafique,

Systems and Computer Engineering

Carleton University, Ottawa, Canada

Introduction

- *Goal:* Find a *good quality* GO/MINLP solution *quickly*.
  - Trade off accuracy for speed
  - No guarantee of finding optimum

- *Target:* very large, highly nonlinear GO/MINLP instances

- *Method:* use a fast approximate Global Optimizer within a B&B framework

The Fast Global Optimizer

- Why is nonconvex GO hard?
  - Multiple disconnected feasible regions
  - Multiple local optima
  - *Many places to look for optima*
- Two main categories of solution methods:
  - Space-covering global optimizers:
    - Accurate, but slow: *inherent tree search*
  - Multi-start local optimizers:
    - Faster, but not as accurate: *whole space not searched*
- ***Goal:*** fast and reasonably accurate GO
  - Trade a little accuracy for speed

GO Components

Main idea:

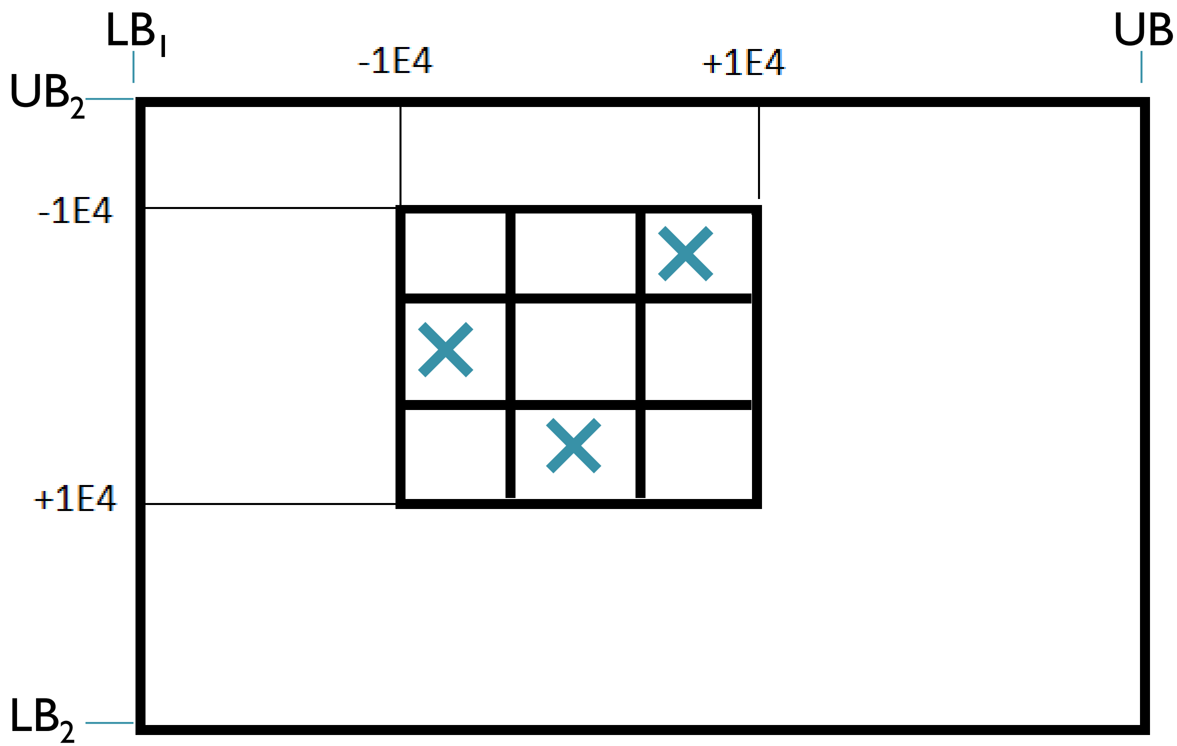
- Multi-start based (for *speed*)
- Better exploration of the variable space before launching the local solver (for *accuracy*)
  - *Our main contribution*

Main steps:

Goal: find local solver launch points that lead to global optimum

1. Latin Hypercube sampling in a defined *launch box*
2. Constraint Consensus concentration
3. Clustering
4. Simple Search
5. Local solver launches

1. LHC Sampling in the *Launch Box*

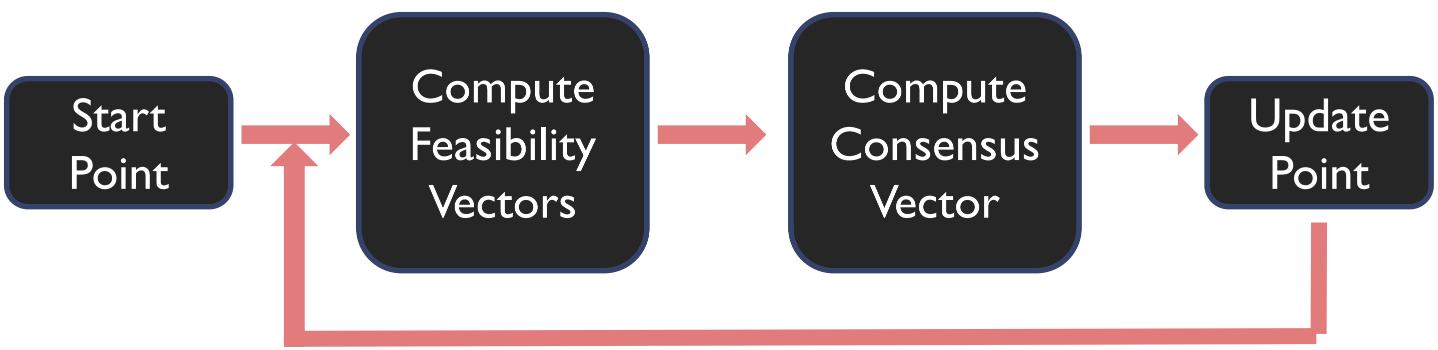


Initial launch box based on empirical results:

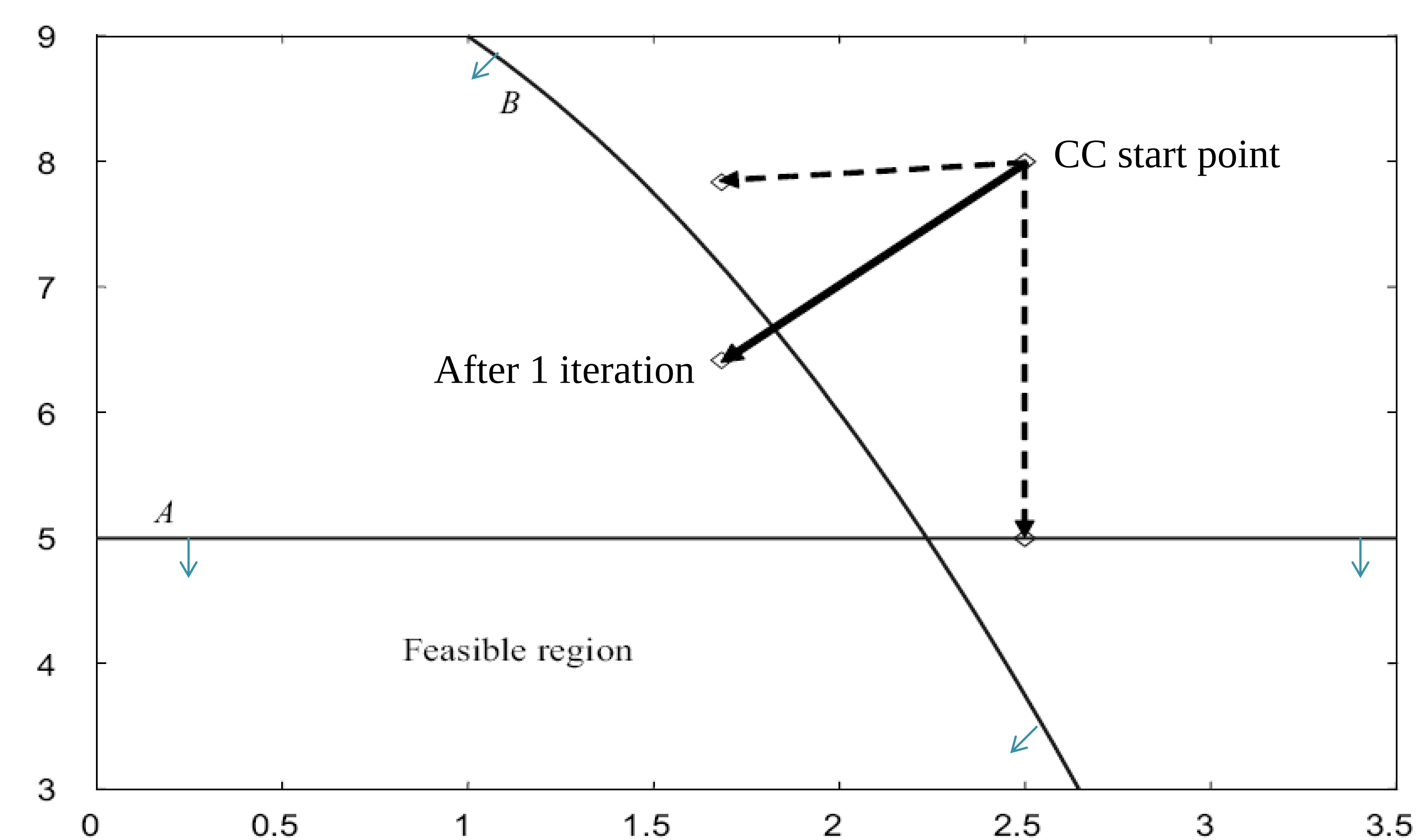
- Most NLP solutions are in this range
- Shifted appropriately according to the bounds

2. Constraint Consensus (CC)[1]

- *Projection method:* iteratively adjusts point to reduce constraint violation(s).
- Quickly moves initial point to near-feasible final point.
- Very fast: no matrix inversion, no line search
- Reduces local solver time, improves success

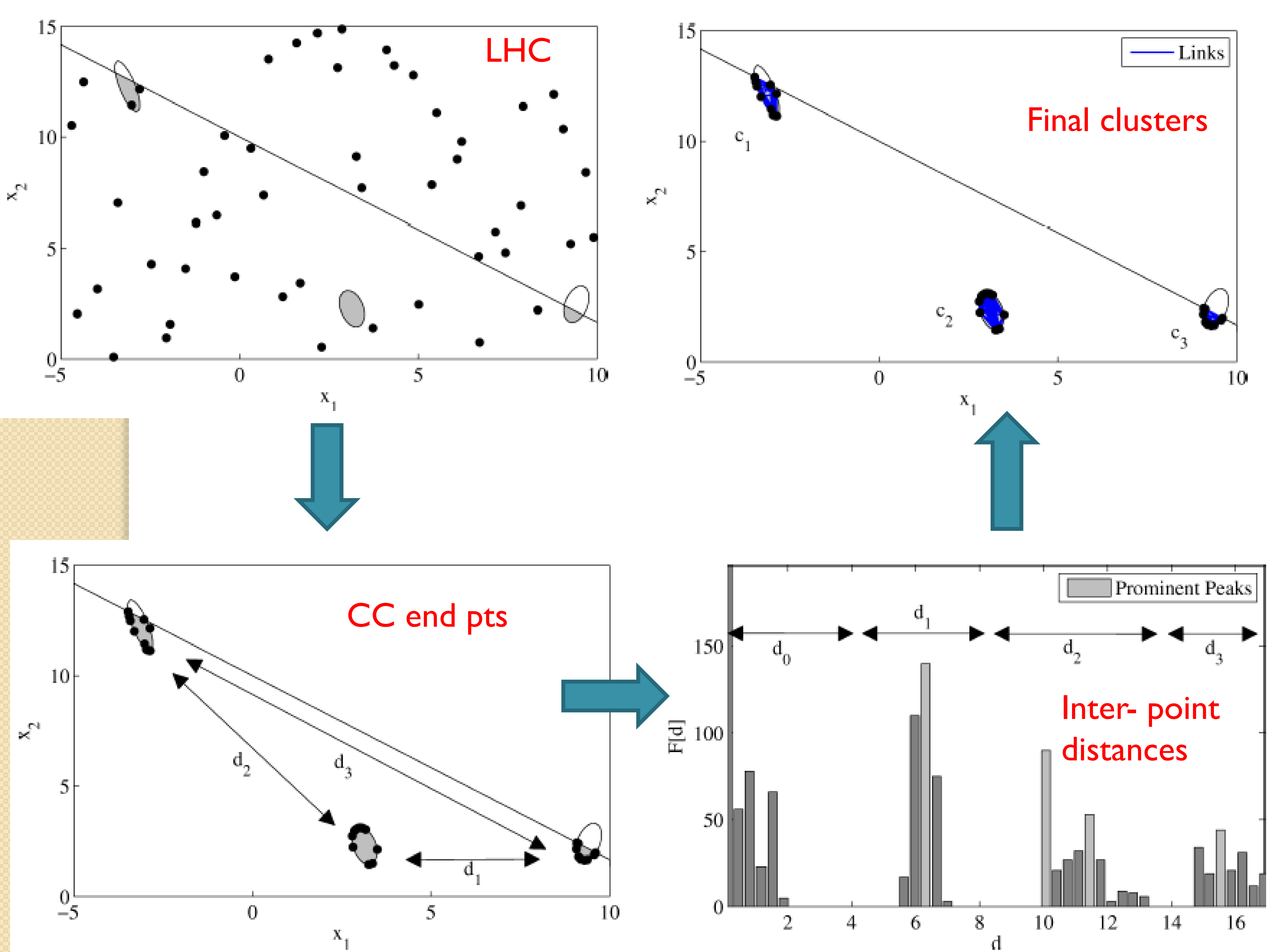


Constraint Consensus



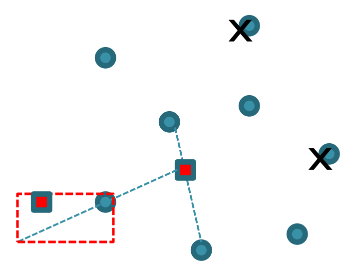
3. Clustering of CC end points (CB)

- ***Single linkage clustering:*** pts closer than *critical distance* assigned to same cluster
- ***Critical distance:*** based on distribution of inter-point distances
  - *Small distances:* points in same cluster
  - *Large distances:* points in different clusters
  - Choose critical distance based on this
- ***Effect:*** clusters correlate with feasible regions


LHC
Final clusters
Links
CC end pts
Inter- point distances
Prominent Peaks
F[d]
d
x1
x2
c1
c2
c3
d0
d1
d2
d3

4. Simple Search (SS)

- Derivative-free neighborhood search for better points
  - *considers both feasibility and objective function*
- ***Point quality metric*** **(minimization):**
  - Penalty function: $P(\boldsymbol{x}) = f(\boldsymbol{x}) + (\text{maximum violation})^2$

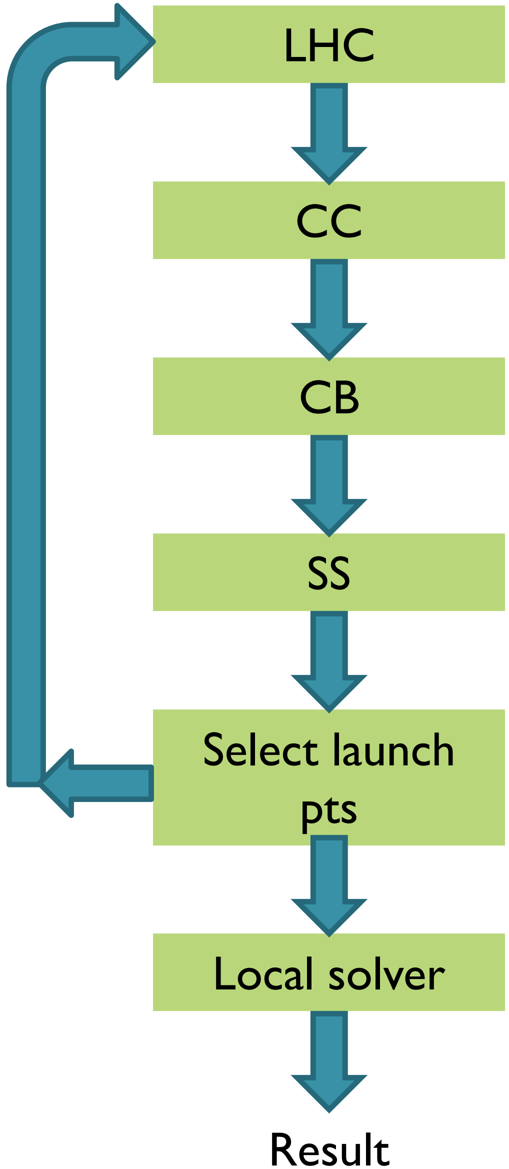


1. Interior random search
2. Exterior random search

Replace worst point

Continue until no improvement for several iterations

Complete GO Algorithm



Serial, but parallelizable. 2 - 4 rounds.

LHC: *Serial*

CC: *Parallel*

CB: *Serial*

SS: *Parallel*

Select launch pts: *Serial.* Identify ***x*** having best P(***x***) value. Note it's round. Take best point in each of 3 best clusters in that round.

Local solver: *Parallel*

Experimental Setup: Software

- **OS:** Fedora 17, 64 bit. Compiler: GCC 4.7.2
- **Modelling language:** AMPL, presolver on
- **Local solver:** IPOPT 3.11.1, linear solver MA86 serial mode, default settings
- **Parameter settings:**
  - *Time limit:* 1800 seconds (half an hour)
  - *Feasibility tolerance:* 1×10^{-6} throughout
  - *LHC parameters:* 60 points, launch box edge length 2×10^{4}
  - *CC parameters:* max 100 iterations per CC run, time limit: 1 sec/run
  - *CB parameter: max 25 clusters*
  - *SS parameters:* at least 10 points per cluster, continue improving until three successive failures.
  - 2 rounds

Experimental Setup

- Hardware:
  - 4-core, 3.4 GHz, 64-bit Intel i7-2600, 16 GB RAM
- Compare to:
  - Knitro (multistart, parallel mode), SCIP, Couenne
  - *BARON not available for AMPL input*
- Test models:
  - **Test set:** 94 CUTEr [2] models having at least one nonlinear function (constraint or objective) and 300+ constraints (before AMPL presolve)
    - 48 have linear constraints with nonlinear objective
    - 46 have nonlinear constraints
  - **Tuning set:** a different set of 35 models

CCGO vs. KNITRO: First Incumbent

- Multistart: 5 runs of each method
- Comparing median values
- Time diff < 1 sec = same

Linear Constraints (48)

	same	CCGO better	KNITRO better
Obj	15	15	18
	0.313	*0.313*	***0.375***
Speed	0	3	45
	0.000	*0.063*	***0.938***
Fails		0	0

Nonlinear Constraints (46)

Comparable Subset (34)

	same	CCGO better	KNITRO better
Obj	25	2	7
	0.735	*0.059*	*0.206*
Speed	0	24	10
	0.000	***0.706***	*0.294*
Fails		11	**3**

CCGO vs. Knitro: Final Solution

- Multistart: 5 runs of each method
- Comparing median values
- Time diff < 1 sec = same

Linear Constraints (48)

	same	CCGO better	KNITRO better
Obj	20	7	21
	0.417	*0.146*	***0.438***
Speed	0	10	38
	0	*0.208*	***0.792***
Fails		0	0

Nonlinear Constraints (46)
Comparable Subset (34)

	same	CCGO better	KNITRO better
Obj	19	1	14
	0.559	*0.029*	*0.412*
Speed	1	29	4
	0.029	***0.853***	*0.118*
Fails		11	**3**

CCGO vs. Knitro: Conclusions

- Both are multistart methods
- *Linear constraints:*
  - similar first incumbent solutions, Knitro better final solutions
  - Knitro faster
- *Nonlinear Constraints:*
  - frequently similar first incumbents and final solutions, Knitro overall better solutions
  - CCGO faster
  - Knitro more robust (fewer failures)
- *Questions*
  - How much of the difference is due to the use of Ipopt in CCGO vs the Knitro local solver?

CCGO vs. SCIP and Couenne: First Incumbent

CCGO median vs. others

Linear Constraints (48)

	CCGO Best	SCIP Best	Couenne Best
Obj	**35 (76%)**	2 (4%)	27 (59%)
Speed	4 (9%)	26 (57%)	**30 (65%)**
Fails	**0 (0%)**	9 (20%)	10 (22%)

Nonlinear Constraints (46)

	CCGO Best	SCIP Best	Couenne Best
Obj	26 (57%)	5 (11%)	**32 (70%)**
Speed	3 (7%)	11 (24%)	**32 (70%)**
Fails	**11 (24%)**	31 (67%)	12 (26%)

CCGO vs. SCIP and Couenne: Final Solution

CCGO median vs. others

Linear Constraints (48)

	CCGO Best	SCIP Best	Couenne Best
Obj	**37 (77%)**	1 (2%)	28 (58%)
Speed	**43 (90%)**	5 (10%)	0 (0%)
Fails	**0 (0%)**	9 (19%)	10 (21%)

Nonlinear Constraints (46)

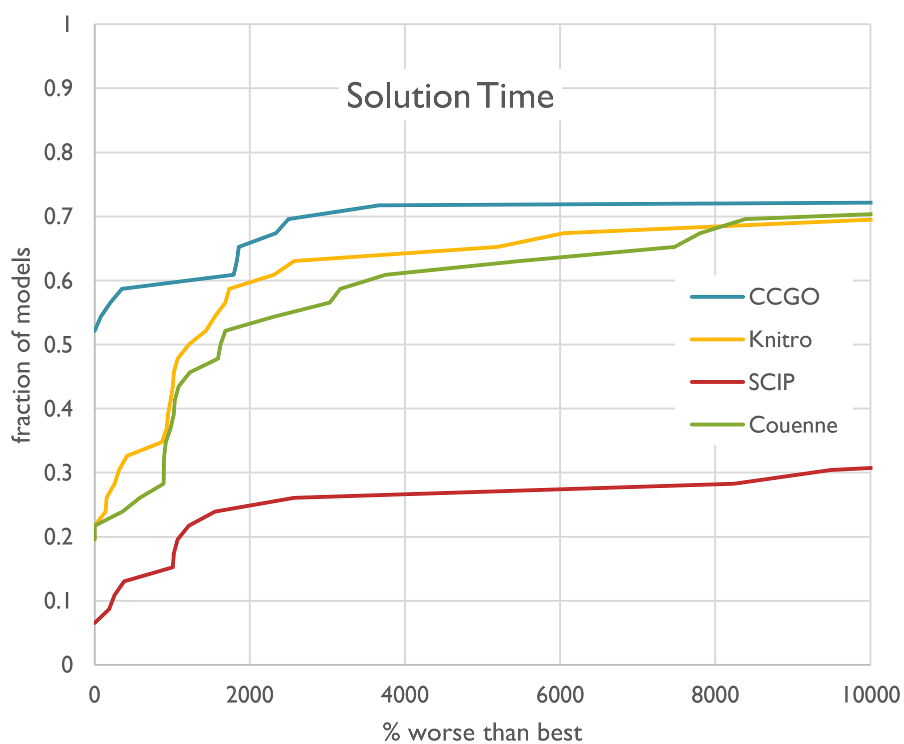
	CCGO Best	SCIP Best	Couenne Best
Obj	26 (57%)	6 (13%)	**36 (78%)**
Speed	**29 (63%)**	7 (15%)	13 (28%)
Fails	**11 (24%)**	31 (67%)	12 (26%)

CCGO vs. SCIP and Couenne: Conclusions

- Linear Constraints:
  - CCGO much more robust
  - 1st inc.: CCGO best solns but slowest
  - Final: CCGO best solns, speed, robustness
- Nonlinear Constraints:
  - CCGO most robust
  - 1st inc.: Couenne best. CCGO good soln quality but slowest.
  - Final: CCGO good soln quality and fastest.
- SCIP and Couenne use initial heuristics that find an early incumbent.

Comparing all 4 Solvers: Nonlinear Constraints

Fraction of models having solution within 1% of best obj fcn value found			
CCGO	**Knitro**	**SCIP**	**Couenne**
63.0%	**89.1%**	4.3%	71.7%



Solution returned for % of models			
CCGO	**Knitro**	**SCIP**	**Couenne**
76.1%	**93.5%**	32.6%	73.9%

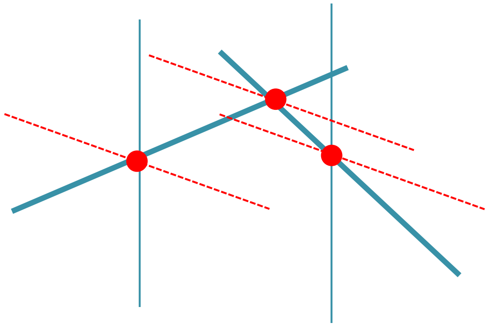
Towards MINLP

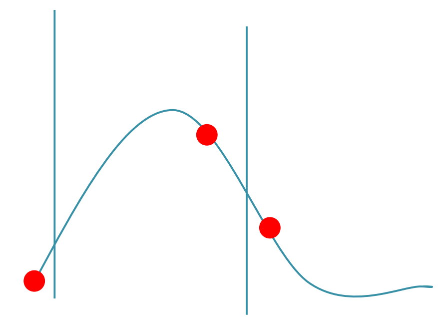
Goal: few local solver launches

1. Solve GO problem approximately
   - LHC-CC-CB-SS, but *no local solver launch*
2. B&B using values for integer variables at approximate GO solution
3. When all integer variables fixed at integer values, launch local solver
4. Continue B&B as usual

Branching Issues

- *Approximate solution affects branching*
- MILP:
  - Exact solver
  - Branching tends to increase integrality
- MINLP with approximate GO solution:
  - Branching may not force early integrality
  - May have to branch until *upper bound = lower bound*



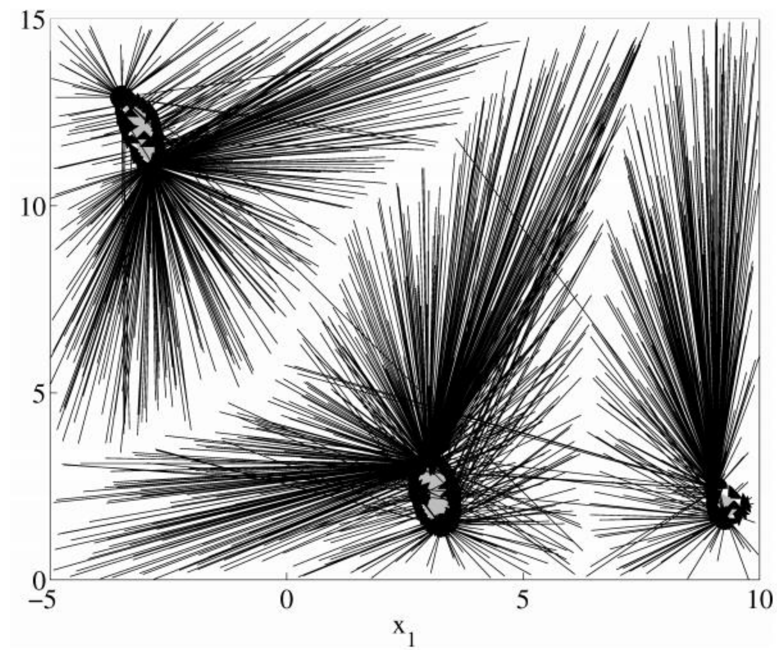


Branching Issues (contd)

- Round to integrality within a (larger) tolerance (e.g. 0.1)?
- Seed the initial random sample of the new subspace with a rounded solution. E.g.
  - Parent solution (11.6, 12.2, 9.5)
  - Down branch special point (11.6, 12.2, 9.0)
  - Up branch special point (11.6, 12.2, 10.0)
- Take action if too many open nodes
  - E.g. round integer variables and launch local solver to get a better incumbent

Spatial Branching

- Likely not needed
- *If needed:* CC start-end pairs map basins of attraction for feasible regions
  - Subdivide using CC start-end pairs to define basins of attraction



(b) 1500 points.

MINLP results to date

- Test set: 8 small general MINLP instances from minlplib2 [3].

Name	#Vars	#BinVars	#IntVars	#Cons
eg_all_s	8	0	7	28
eg_disc2_s	8	0	3	28
gear3	8	0	4	4
m7_ar4_1	112	0	42	269
m7_ar5_1	112	0	42	269
nvs01	3	0	2	3
o7_ar2_1	112	0	42	269
o7_ar3_1	112	0	42	269

- IPOPT runtime = maximum 50 seconds
- Kept track of first 100 nodes in B&B tree
- 5.1 integer-feasible solutions found on avg

Conclusions

- GO results are promising
  - Soln quality good
  - Soln speed very good for nonlinear constraints
- Future work:
  - GO parameter optimization
  - Incorporation of new heuristics for robustness and quick first incumbent
  - Improved integer branching

Looking for a good post-doc

- Topic: concurrent optimization

- About Ottawa, Canada:
  - Canada's capital
  - Many fine museums, outdoor festivals
  - Canoeing, kayaking, hiking, camping, skiing
  - Close(ish) to Montreal
  - English/French bilingual

- *Must like snow*