

Fast Probabilistic Modeling for Combinatorial Optimization



Scott Davies

School of Computer Science

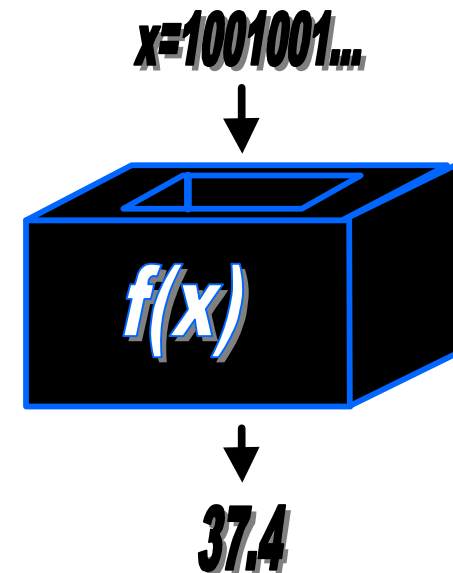
Carnegie Mellon University

(and Justsystem Pittsburgh Research Center)

Joint work with Shumeet Baluja

Combinatorial Optimization

- Maximize “evaluation function” $f(x)$
 - input: fixed-length bitstring $x = (x_1, x_2, \dots, x_n)$
 - output: real value
- x might represent:
 - job shop schedules
 - TSP tours
 - discretized numeric values
 - etc.
- Our focus: “Black Box” optimization
 - No domain-dependent heuristics



Commonly Used Approaches

- Hill-climbing, simulated annealing
 - Generate candidate solutions “neighboring” single current working solution (e.g. differing by one bit)
 - Typically make no attempt to model how particular bits affect solution quality
- Genetic algorithms
 - Attempt to implicitly capture dependency of solution quality on bit values by maintaining a population of candidate solutions
 - Use *randomized* crossover and mutation operators on population members to generate new candidate solutions

Using Explicit Probabilistic Models

- Maintain an explicit probability distribution P from which we generate new candidate solutions
 - Initialize P to uniform distribution
 - Until termination criteria met:
 - Stochastically generate K candidate solutions from P
 - Evaluate them
 - Update P to make it more likely to generate solutions “similar” to the “good” solutions
 - Return best bitstring ever evaluated
- Several different choices for what sorts of P to use and how to update it after candidate solution evaluation

Previous Probabilistic Methods

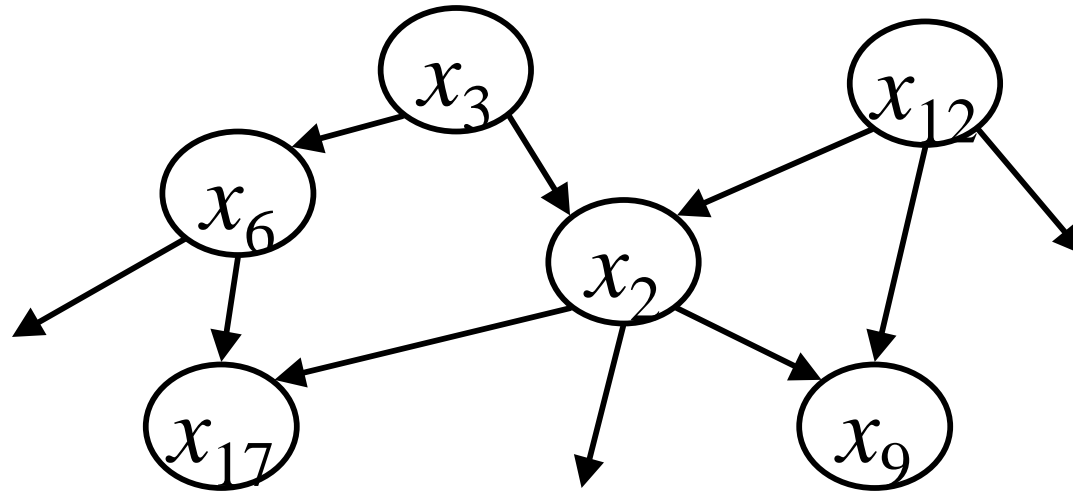
■ Population-Based Incremental Learning (PBIL) [Baluja, 1995]

- Maintains a vector of probabilities: one *independent* probability $P(x_i=1)$ for each bit x_i . (Initialized to .5)
- Until termination criteria met:
 - | Generate a population of K bitstrings from P . Evaluate them.
 - | Adjust P to make M best more likely
- Often works very well (compared to hillclimbing and GAs) despite its independence assumption

■ MIMIC [De Bonet, et al., 1997]: model best $N\%$ of bitstrings evaluated so far with a Markov Chain

- e.g.: $P(x_1, x_2, \dots, x_n) = P(x_3) * P(x_6|x_3) * P(x_2|x_6) * \dots$

Bayesian Networks



$$P(x_1, \dots, x_n) = P(x_3) * P(x_{12}) * P(x_2 | x_3, x_{12}) * P(x_6 | x_3) * \dots$$

■ Specified by:

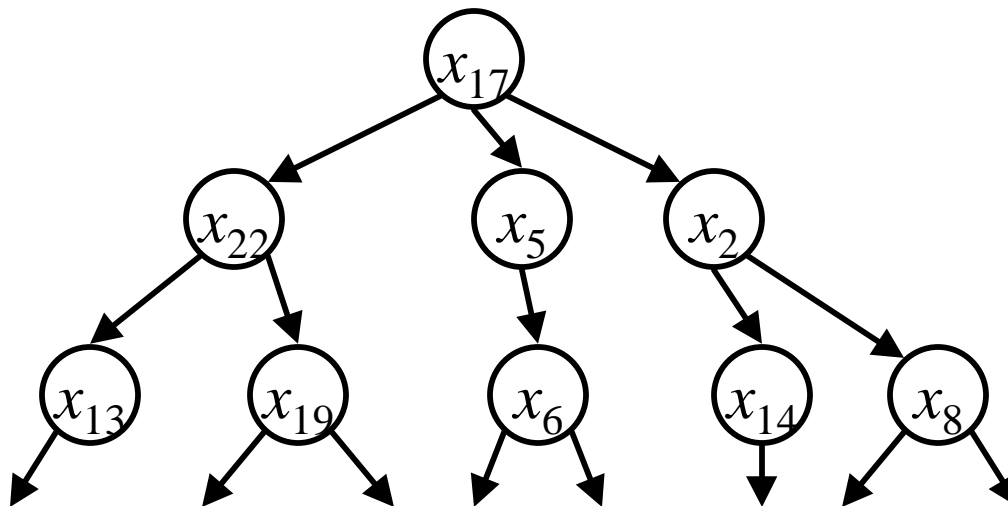
- Network structure (must be a DAG)
- Probability distribution for each variable (bit) given each possible combination of its parents' values

Optimization with Bayesian Networks

- Two operations we need to perform during optimization:
 - Given a network, generate bitstrings from the probability distribution represented by that network
 - | Trivial with Bayesian networks
 - Given a set of “good” bitstrings, find the network most likely to have generated it
 - | Given a fixed network structure and a dataset with no missing values, trivial to fill in the probability tables optimally
 - | **But what structure should we use?**
 - Finding best possible structure is NP-hard [Chickering, et al., 1995]

Single-Parent Tree-Shaped Networks

- Let's allow each bit to be conditioned on at most one other bit.



Optimal Single-Parent Tree-Shaped Networks

- To find optimal single-parent tree-shaped network, just find maximum spanning tree using $I(x_i, x_j)$ as the weight for the edge between x_i and x_j .
[Chow and Liu, 1968].
 - $I(x_i, x_j)$ = mutual information between x_i and x_j
- Can be done in $O(n^2)$ time (assuming D has already been reduced to sufficient statistics)

Optimal Dependency Trees for Combinatorial Optimization

[Baluja & Davies, 1997]

- Start with a dataset D initialized from the uniform distribution
- Until termination criteria met:
 - Build optimal dependency tree T with which to model D .
 - Generate K bitstrings from probability distribution represented by T . Evaluate them.
 - Add best M bitstrings to D after decaying the weight of all datapoints already in D by a factor α between 0 and 1.
- Return best bitstring ever evaluated.
- Running time: $O(K*n + M*n^2)$ per iteration

Graph-Coloring Example

■ Noisy Graph-Coloring example



A: graph-coloring problem

B: PBIL's "model" of problem

C: Automatically learned Markov chain [as in MIMIC]

D: Automatically learned dependency tree

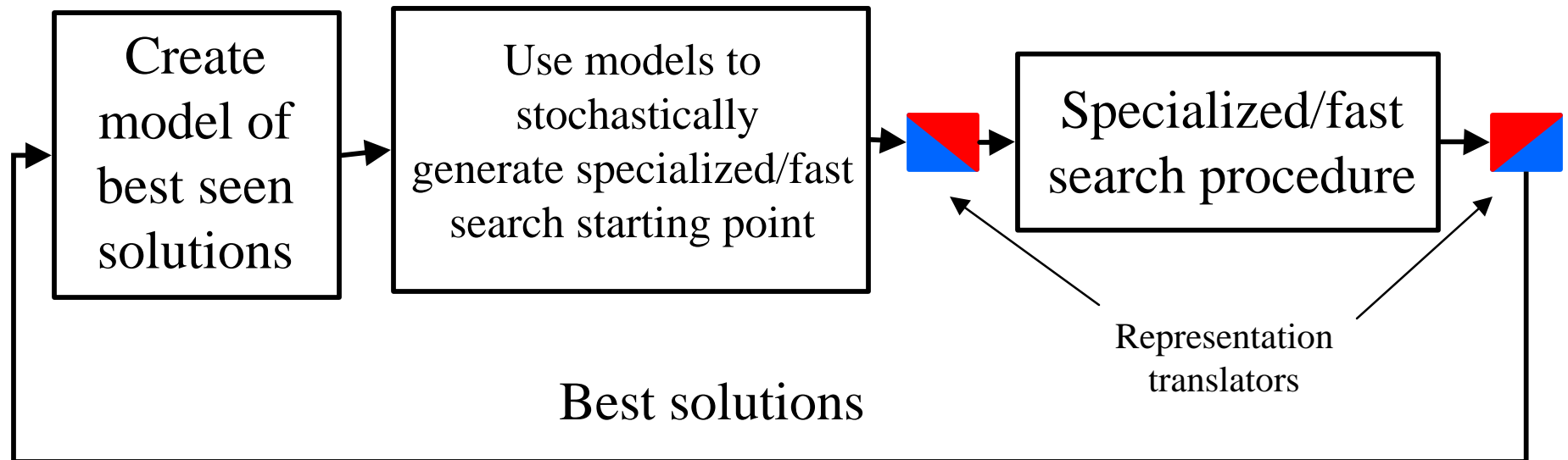
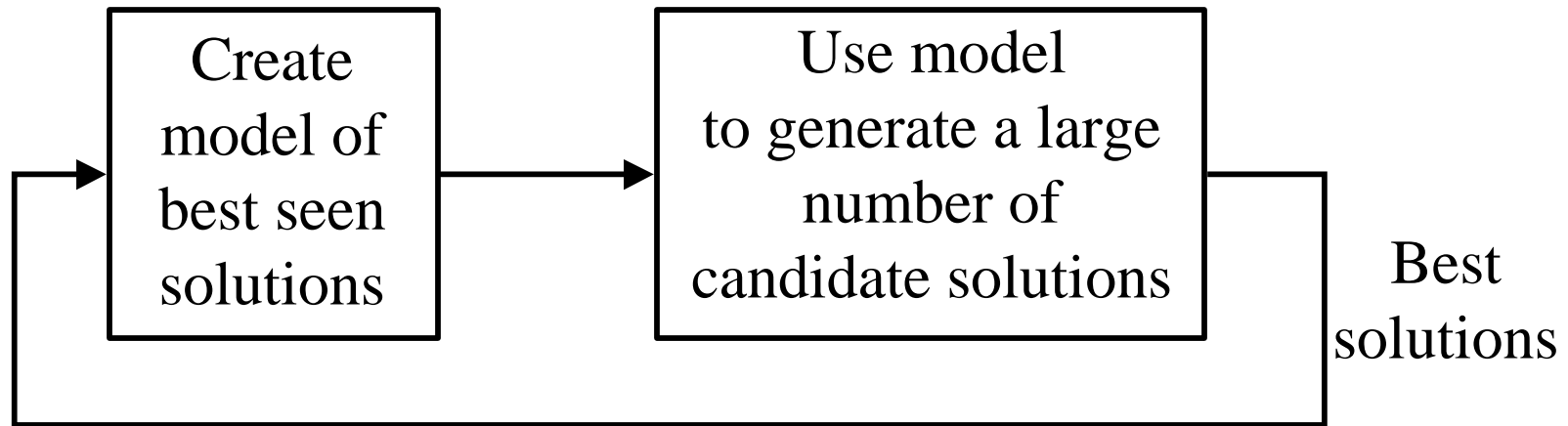
Optimal Dependency Tree Results

- Tested following optimization algorithms on variety of small problems, up to 256 bits long:
 - Optimization with optimal dependency trees
 - Optimization with chain-shaped networks (ala MIMIC)
 - PBIL
 - Hillclimbing
 - Genetic algorithms
- General trend:
 - Probabilistic methods (PBIL, chains, trees) did better than hillclimbing & GAs
 - Among probabilistic methods, $\text{PBIL} < \text{Chains} < \text{Trees}$:
more accurate models are better

Using Probabilistic Models for Intelligent Restarts

- Tree-based algorithm's $O(n^2)$ execution time per iteration very expensive for large problems
 - Even more so for algorithm based on more complicated Bayesian networks
- One possible approach: use probabilistic models to select good starting points for faster optimization algorithms, e.g. hillclimbing or PBIL
 - Can also be used with specialized domain-dependent search methods!
 - “Wrapper” approach: modeling used to “wrap around” other randomized optimization algorithms and provide them with good starting points

Straight Probabilistic Methods vs. “Wrappers”



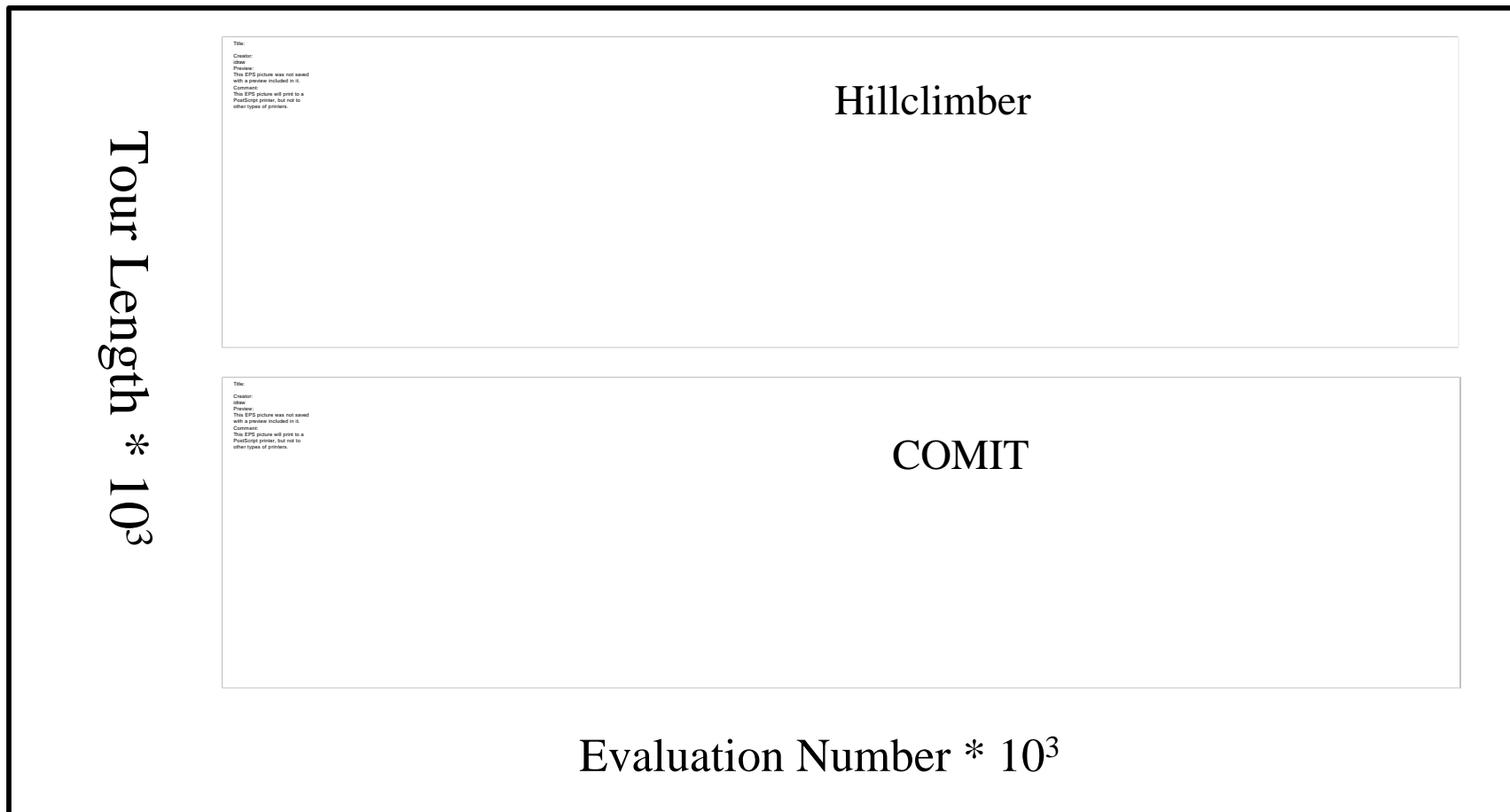
COMIT

■ Combining Optimizers with Mutual Information Trees [AAAI 1998]

- Initialize dataset D with bitstrings drawn from uniform distribution
- Until termination criteria met:
 - | Build optimal dependency tree T with which to model D .
 - | Use T to stochastically generate K bitstrings. Evaluate them.
 - | Execute a fast-search procedure, initialized with the best solutions generated from T .
 - | Replace up to M of the worst bitstrings in D with the best bitstrings found during the fast-search run just executed.
- Return best bitstring ever evaluated

COMIT w/Hillclimbing: Example of Behavior

■ TSP domain: 100 cities, 700 bits



COMIT w/Hillclimbing: Results

■ Highlighted: best COMIT better than best hillclimber with confidence > 95%

Problem	Min/Max	# bits	HC	COMIT
Knapsack 1	Max	512	3377	6684
Knapsack 2	Max	900	3488	7733
Knapsack 3	Max	1200	5280	13052
Jobshop 1	Min	500	982	970
Jobshop 2	Min	700	957	953
Jobshop 3	Min	700	1199	1195
Bin-packing 1	Min	504	158	145
Bin-packing 2	Min	800	138	111
TSP 1 (100 city)	Min	700	1573	1335
TSP 2 (200 city)	Min	1600	15100	15117
TSP 3 (150 city)	Min	1200	11247	9077
Sum. Canc.	Min	675	59	52

COMIT with PBIL

- Generate K samples from T ; evaluate them
- Initialize PBIL's \mathbf{P} vector according to the unconditional distributions contained in the best $C\%$ of these examples
- PBIL run terminated after 5000 evaluations without improvement

COMIT w/PBIL: Results

- Highlighted: better than opposing algorithm(s) with confidence > 95%

Problem	Min/Max	# bits	PBIL	COMIT+PBIL
Knapsack 1	Max	512	7823	7872
Knapsack 2	Max	900	9601	10134
Knapsack 3	Max	1200	14668	18014
Jobshop 1	Min	500	963	961
Jobshop 2	Min	700	943	940
Jobshop 3	Min	700	1176	1170
Bin-packing 1	Min	504	140	280
Bin-packing 2	Min	800	830	1000
TSP 1 (100 city)	Min	700	1331	1021
TSP 2 (200 city)	Min	1600	17148	14870
TSP 3 (150 city)	Min	1200	11035	9078
Sum. Canc.	Min	675	25	33

Conclusions

- COMIT makes probabilistic modeling applicable to much larger problems
- Led to significant improvements over baseline search algorithm in most problems tested
- Allows use of specialized search heuristics

Future Work

- Incorporating domain knowledge
- Using COMIT to combine results of multiple search algorithms
- Using more sophisticated probabilistic models?
 - Hidden variables
 - More complicated structures
 - Models other than Bayesian networks
- Applying COMIT to bigger problems

Comparison: COMIT vs. STAGE

■ STAGE (Boyan & Moore, 1998):

- +Used problem-specific knowledge to find salient features of good solutions.
- Untested with large numbers of features -- such as actual encoding of entire solution
- Value function approximator cannot fit too accurately, or plateaus will confuse hillclimbers

■ Probabilistic modeling:

- +Works directly with parameters in solution encoding.
 - +High dimensions OK
- +No domain knowledge required
- +Can encode prior knowledge about parameter dependencies
- Open question how to incorporate knowledge of salient high-level features