A Novel Metaheuristic Search Technique

Iterative Treatment Learning

Jeremy Greenwald

Portland State

February 14, 2007

Outline

1. Preface
    - Research Motivation
    - Problem Statement and Goals

Outline

1. Preface
    - Research Motivation
    - Problem Statement and Goals
2. Treatment Learning

Outline

1. Preface
   - Research Motivation
   - Problem Statement and Goals
2. Treatment Learning
3. Metaheuristic Search
   - Simulated Annealing
   - Evolutionary Algorithms

Outline

1. Preface
    - Research Motivation
    - Problem Statement and Goals
2. Treatment Learning
3. Metaheuristic Search
    - Simulated Annealing
    - Evolutionary Algorithms
4. Search Based SE

Outline

Outline

1. Preface
    - Research Motivation
    - Problem Statement and Goals
2. Treatment Learning
3. Metaheuristic Search
    - Simulated Annealing
    - Evolutionary Algorithms
4. Search Based SE
5. ITL
6. Experiments
    - Requirements Engineering
    - Property Verification

Outline

1. Preface
   - Research Motivation
   - Problem Statement and Goals
2. Treatment Learning
3. Metaheuristic Search
   - Simulated Annealing
   - Evolutionary Algorithms
4. Search Based SE
5. ITL
6. Experiments
   - Requirements Engineering
   - Property Verification
7. Conclusions and Future Work

Research Motivation

Can we get more out of the models that are generated when using model-based development (MBD)? Does search-based software engineering (SBSE) offer a methodology for increasing the effectiveness of these models?

Problem Statement

ITL is a new SB technique that we would like to

- establish as metaheuristic search technique with unique properties
- apply to models representing different stages in the software life cycle

Goals

My goals are

1. set ITL in a metaheuristic search context
2. improve the performance of ITL by developing and tuning *extreme sampling*
3. apply ITL to
   1. early life stage models - requirements engineering
   2. late life stage models - testing

1. Preface
    - Research Motivation
    - Problem Statement and Goals
2. Treatment Learning
3. Metaheuristic Search
    - Simulated Annealing
    - Evolutionary Algorithms
4. Search Based SE
5. ITL
6. Experiments
    - Requirements Engineering
    - Property Verification
7. Conclusions and Future Work

What is Treatment Learning?

Key elements are

- lift, ordering heuristic
- minimum best support, over-fitting avoidance
- small treatment effect, solution form

What is Treatment Learning?

Treatments are conjunctions of range restrictions or assignments to some of the independent variables. Lifts are the ratio of the normalized weighted sums between treated instances and entire training set.

$$\begin{aligned} lift &= \frac{treated\ average}{baseline} \\ &= \frac{\sum_{i=0}^{n}(weight_i * treated_i)}{\sum_{i=0}^{n}(weight_i * prop_i)} \end{aligned}$$

Why Do We Prefer Treatment Learning?

- Theories returned by machine learners are supposed to *generalize* from training data
- Explanatory, as opposed to performance, systems should offer insight to human users about hidden relationships in the data
- Treatment learners offer theories that reveal these relations in a **comprehensible** form

Example Treatment

Training on Boston housing data, tar3 finds:

$$\boxed{\begin{aligned} & rm \geq 6.6 \wedge ptratio \leq 15.9 \Rightarrow \\ & \qquad \mathbf{P}(high) = 97\%,\ \mathbf{P}(medhigh) = 3\% \end{aligned}}$$

where the baseline distribution was
29% *high*, 29% *medhigh*, 21% *medlow*, 21% *low*
and each instance has 13 independent attributes

This treatment is small, comprehensible, and valuable.

Example Treatment

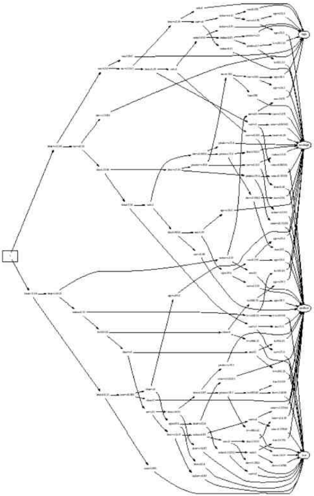
Training on Boston housing data, tar3 finds:

$$\boxed{\begin{array}{l} rm \geq 6.6 \land ptratio \leq 15.9 \Rightarrow \\ \qquad \mathbf{P}(high) = 97\%,\ \mathbf{P}(medhigh) = 3\% \end{array}}$$

where the baseline distribution was
29% *high*, 29% *medhigh*, 21% *medlow*, 21% *low*
and each instance has 13 independent attributes

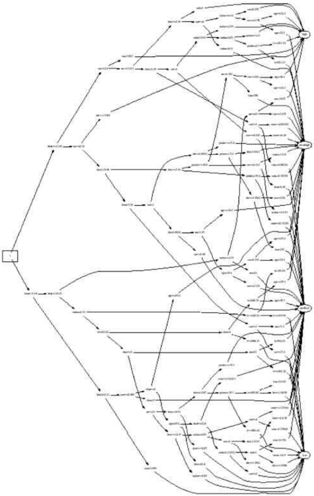
This treatment is small, comprehensible, and valuable. But what does a theory look like from another well-regarded data miner . . .

Example Decision Tree



This decision tree, while accurate, does not lend itself to easy interpretation by human experts. If the point of an explanatory system is to reveal relationships, what does this tree reveal?

Example Decision Tree



This decision tree, while accurate, does not lend itself to easy interpretation by human experts. If the point of an explanatory system is to reveal relationships, what does this tree reveal?

Anything?

1. Preface
    - Research Motivation
    - Problem Statement and Goals
2. Treatment Learning
3. Metaheuristic Search
    - Simulated Annealing
    - Evolutionary Algorithms
4. Search Based SE
5. ITL
6. Experiments
    - Requirements Engineering
    - Property Verification
7. Conclusions and Future Work

What is Metaheuristic Search

Metaheuristic search techniques are used in optimization problems that can't be solved using analytic or complete methods.

They strive to find an acceptable *near-optimal* solution, without any mathematical guarantee that a better solution wouldn't be found if the search is re-initialized.

Metaheuristic Search

Three of the most common metaheuristic search technique

- simulated annealing

Metaheuristic Search

Three of the most common metaheuristic search technique

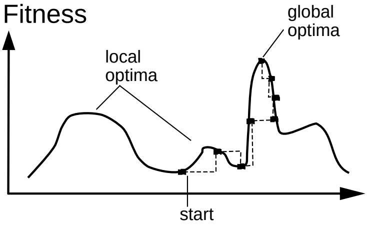
- simulated annealing
- evolutionary algorithms

Metaheuristic Search

Three of the most common metaheuristic search technique

- simulated annealing
- evolutionary algorithms
- tabu search

Simulated annealing



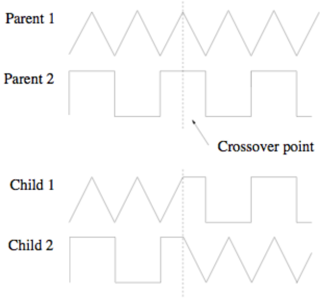
Hill climber that tries to not get stuck in local optima.

Probability of making a bad move $\propto e^{-(\nabla E/T)}$ and "cooling" schedule controls the value of the temperature.

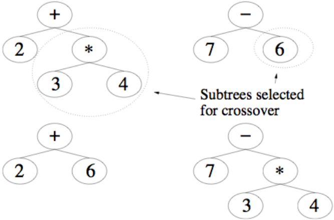
Evolutionary Algorithms

genetic algorithms (GA) and genetic programming (GP)

GA crossover event



GP crossover event



Rethinking Software Engineering

Key steps to reformulating research goal as numeric optimization (Clarke, Harman, Jones et. al.)

- representation of candidate solutions
- objective function
- transformation operators

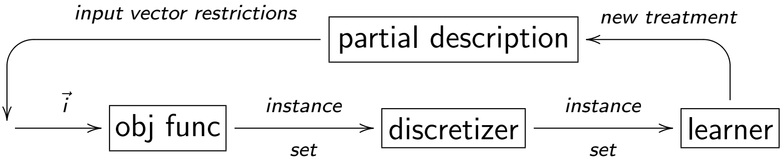
Is SBSE a Useful Idea?

According to a review of literature from 1992 to 2003 evolutionary algorithms were used to solve problems from different life stages of the software life cycle (Rela)

life stage	number of publications
planning	19
design	40
implementation	10
testing	54
total	123

1. Preface
   - Research Motivation
   - Problem Statement and Goals
2. Treatment Learning
3. Metaheuristic Search
   - Simulated Annealing
   - Evolutionary Algorithms
4. Search Based SE
5. **ITL**
6. Experiments
   - Requirements Engineering
   - Property Verification
7. Conclusions and Future Work

ITL



Each trip through the cycle is an *iteration* and each iteration adds the new treatment to the input vector restrictions so that a smaller hypervolume is sampled from the next iteration.

Partial Descriptions

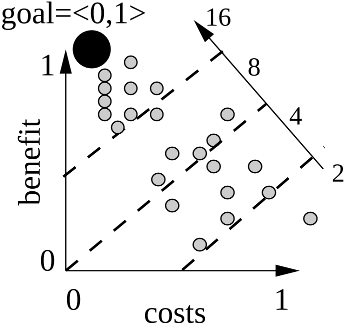
Recall that treatments only comment on a few attributes and these comments are not always assignments.

Partial Descriptions

Recall that treatments only comment on a few attributes and these comments are not always assignments.
The candidate solution describes a portion of the input space rather than a single point.

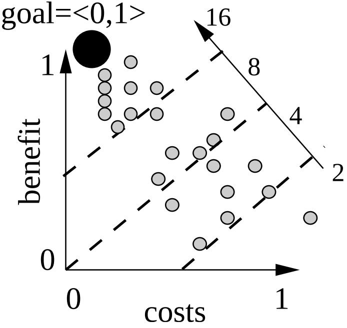
Discretizer

Previous work used a diagonal striping discretizer

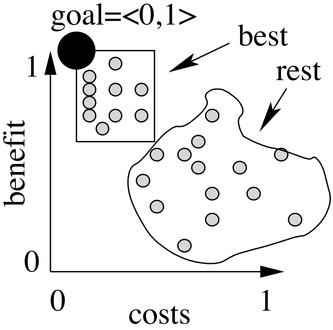


Discretizer

Previous work used a diagonal striping discretizer

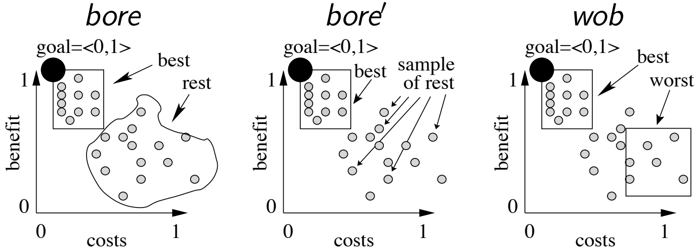


This work uses *extreme sampling*



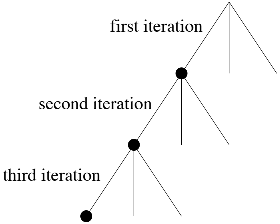
Extreme Sampling

Different versions of extreme sampling



Search strategy

ITL search strategy is a greedy forward select



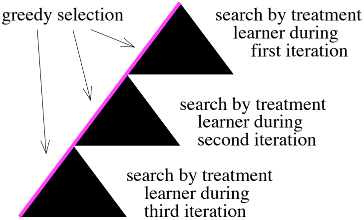
that is raises the question . . .

Search strategy

Why does it work?

Search strategy

Why does it work?
Because inside each iteration, treatment learner doing bushier search



Search strategy

Of course there are lots of improvements that could be made, but . . .

Search strategy

Of course there are lots of improvements that could be made, but . . .

Let's validate the simple method, before developing more complex strategies.
We will see that empirically this simple search works quite well anyway.

Requirements Engineering

Early life cycle work

Defect Detection and Prevention

DDP models have

- weighted objectives
- risks to objectives
- mitigations to reduce risks, but cost money

to find what “design” maximizes the objectives achieved, while costing the least.

DDP was developed at JPL and has proven its effectiveness over the last seven years on dozens of JPL projects.

DDP has a built-in simulated annealer to find a near-optimal design.

The Models

Studied three DDP models, *aero*, *cob*, and *holo*. Previously only *aero* had been studied previously, using diagonal striping.

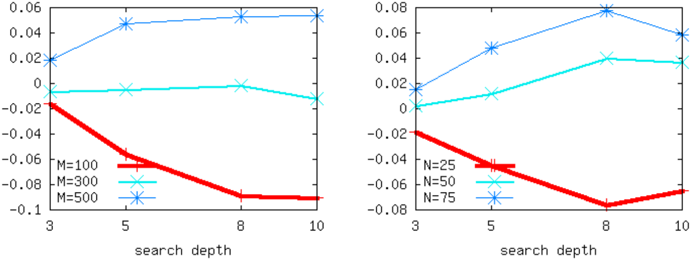
Question 1: Extreme sampling tuning

Extreme sampling has two parameters that must be set by the user.

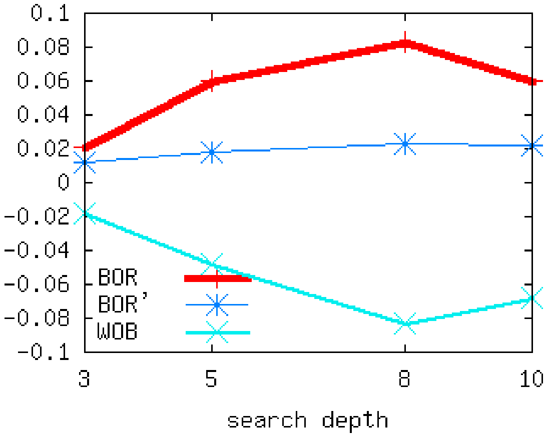
- M, batch size
- N, how are instances assigned to the good and bad class

And which of the three versions (*bore*, *bore′*, *wob*) is preferred

Answer 1


0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
M=100
M=300
M=500
3
5
8
10
search depth
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
N=25
N=50
N=75
3
5
8
10
search depth

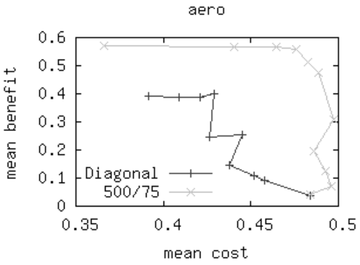
Answer 1

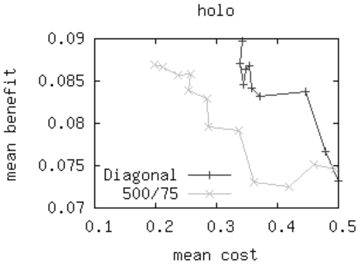

0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
-0.1
3
5
8
10
search depth
BOR
BOR'
WOB

Question 2: *bore* vs. diagonal

Now that we have picked the highest performing version of extreme sampling that we investigated, is our new discretizer better than our old one?

Answer 2



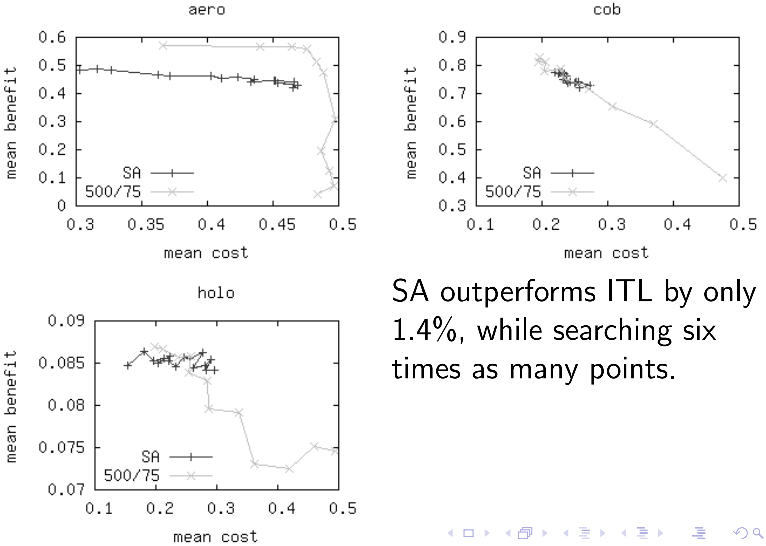


bore outperforms diagonal by 61% while using only 25% of the number of data points.

Question 3: *bore* vs. SA

Our new discretizer increases the performance of ITL, but how does ITL (with *bore*) compare to other metaheuristic search techniques?
Let's compare ITL to the built-in simulated annealer in DDP.

Answer 3



SA outperforms ITL by only 1.4%, while searching six times as many points.

Conclusions from DDP

The experiments with DDP and ITL show

- how M/N effect performance of extreme sampling, *bore*-500/75 the best investigated

Conclusions from DDP

The experiments with DDP and ITL show

- how M/N effect performance of extreme sampling, *bore*-500/75 the best investigated
- *bore* better than diagonal
    - faster
    - higher quality solutions

Conclusions from DDP

The experiments with DDP and ITL show

- how M/N effect performance of extreme sampling, *bore*-500/75 the best investigated
- *bore* better than diagonal
    - faster
    - higher quality solutions
- ITL faster than simulated annealer
    - faster
    - same quality solutions

Property Verification

Late life cycle work

SPY

Motivation for SPY development was to create an environment to validate temporal properties in models with real-valued inputs.
The presence of real-valued inputs makes validating these types of models difficult with complete model checking techniques. Two common ways to use model checkers with real-valued models

- bounded checking
- abstraction

SPY

Motivation for SPY development was to create an environment to validate temporal properties in models with real-valued inputs.

The presence of real-valued inputs makes validating these types of models difficult with complete model checking techniques. Two common ways to use model checkers with real-valued models

- bounded checking
- abstraction

A complement to, not a replacement of, model checkers.

SPY

The SPY framework checks these types of models by conducting an ITL search. Hence, like all metaheuristic methods, it has no completeness guarantee, but has no restrictions on model input types.
Desired behavior is determined by the development what is called the *worth* function.

NASA flight models

We investigated three NASA flight models developed by NASA contractors that were translated from Simulink by a translation framework developed by our UMN collaborators

- Dual FGS
- Altitude Switch
- Voting Sensor

There were a total of ten properties we checked in these models.

NASA flight models

We investigated three NASA flight models developed by NASA contractors that were translated from Simulink by a translation framework developed by our UMN collaborators

- Dual FGS
- Altitude Switch
- Voting Sensor

There were a total of ten properties we checked in these models.

In addition, we baselined SPY against Reactis, a popular commercial product we the same capabilities and a similar methodology.

Question 1: temporal properties

Can SPY find temporal property violations in models with real-valued inputs.

Answer 1

SPY found the same property violations as our baseline tool. Some violations were injected into the property formulations to ensure that every model had at least one violated property.

Problems with Flight Models

The flight models checked by SPY were temporal models, which raised issues not seen with the non-temporal benefit/cost models.

First attempt to learn on the values of each discrete time step for every input variable usually defeated the learners ability to find useful treatments.

When it did find treatments, it was realized their form was not useful.

Problems with Flight Models

We ended up parameterizing the input variables as probability and step functions, and the presented the learner with the function parameters.

This led to less than useful range restrictions.

Solution

Investigate less complex models, that have more controllable inputs.

We choose two different biomathematical models which were temporal, but whose input variables remained fixed throughout the simulation.

Competitive Exclusion

A system of two interacting species

$$\frac{du_1}{d\tau} = u_1(1 - u_1 - a_{12}u_2)$$
$$\frac{du_2}{d\tau} = \rho u_2(1 - u_2 - a_{21}u_1)$$

Five independent variables, ρ, u_1^0, u_2^0, a_{12}, and a_{21}.

Competitive Exclusion

Symbolic analysis shows that system always approaches an equilibrium state and that if
$0 \leq a_{12} \leq 1$ and $0 \leq a_{21} \leq 1$
both species have a non-zero population at the equilibrium state.

Question 2: Can SPY find the correct ranges

We want SPY to find ranges on the relevant independent variables (a_{12} and a_{21}) that lead to the desired model output behavior.
We also want SPY to ignore irrelevant independent variables (ρ, u_1^0, and u_2^0).

Answer 2

Relevant variables

name	lower bounds	upper bounds	speed
a_{12}	0.0(10)	0.2(5), 0.6(2), 0.8(2)	1.8
a_{21}	0.0(10)	0.2(3), 0.3, 0.4(4), 0.8(2)	1.2

Irrelevant variables

name	lower bounds	upper bounds	speed
ρ	0.0(10)	15.0(10)	1.0
u_1^0	0.04(10)	2.04(10)	1.0
u_2^0	0.04(10)	2.04(10)	1.0

Animal Neurons

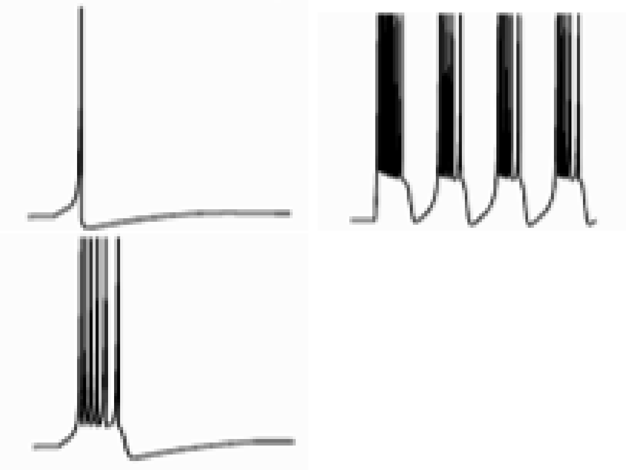
A computationally model that can simulate the behavior of real neurons

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I$$
$$\frac{du}{dt} = a(bv - u)$$
$$\text{if } v \geq 30mV, \text{then } v \leftarrow c \text{ and } u \leftarrow u + d$$

Four independent variables, a, b, c, and d.

Animal Neurons

Three example behaviors

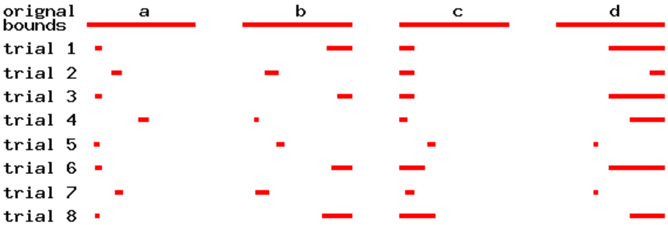


Question 3: Can SPY find these selected behaviors

After developing three worth functions, one each for the three different example behaviors, can SPY find restrictions to the independent variables that confine the model output behavior.

Answer 3

Tonic bursting



8 of 10 trials were successful, most successful objective function.

Conclusions from SPY

SPY experiments with flight and biomathematical models show

- SPY execution model is effective
- translation preserved model semantics
- SPY found property violations that were
    - presented in the model
    - injected in the property formulations
- potentially useful range restrictions found in the biomathematical models

Conclusions from SPY

SPY experiments with flight and biomathematical models show

- SPY execution model is effective
- translation preserved model semantics
- SPY found property violations that were
  - presented in the model
  - injected in the property formulations
- potentially useful range restrictions found in the biomathematical models
- time-dependent independent variables have to be parameterized functions
- difficult to optimize discrete/non-continuous functions

Conclusions

We found that

- *bore*-500/75 are the best setting to extreme sampling
- produces stable, low variance solutions
- performs better with *bore* than with diagonal discretizer
- performs as well, but faster than, original simulated annealer
- boolean/non-continuous functions hard to optimize

Conclusions

We found that

- *bore*-500/75 are the best setting to extreme sampling
- produces stable, low variance solutions
- performs better with *bore* than with diagonal discretizer
- performs as well, but faster than, original simulated annealer
- boolean/non-continuous functions hard to optimize
- used ITL on an early stage activity, requirements engineering
- used SPY on a late stage activity, testing

Future Work

- a smarter search strategy
- ability to optimize discrete or non-continuous functions

Questions? Comments?