Non-Dominated Sorting

Maxim Buzdalov

ITMO University

11.03.2015

Preface

If you were me, what option do you prefer to get from home to university?

1. By foot: 0 roubles, $\approx$ 9 hours
2. By car: $\approx$ 100 roubles, $[1; 1.5]$ hours depending on traffic
3. By train: $\approx$ 2h 10m, 110 roubles
4. By train and subway: $\approx$ 2h, 90 roubles

Preface

If you were me, what option do you prefer to get from home to university?

1. By foot: 0 roubles, $\approx$ 9 hours
2. By car: $\approx$ 100 roubles, $[1; 1.5]$ hours depending on traffic
3. By train: $\approx$ 2h 10m, 110 roubles
4. By train and subway: $\approx$ 2h, 90 roubles

Definitely not #3!

Pareto-optimality

- We are solving optimization problem
  - Assume there are K criteria which are equally important
  - We want to minimize each of them, but they depend on each other
- Solutions:
  - $P = (P_1, P_2, \ldots, P_K)$
  - $Q = (Q_1, Q_2, \ldots, Q_K)$
- $P \prec Q$ (P dominates Q) iff:
  - $\forall i, 1 \le i \le K : P_i \le Q_i$
  - $\exists i, 1 \le i \le K : P_i < Q_i$
- What to do if there are 1 000 000 solutions?

Plan

Non-dominated sorting: what is it?

Offline algorithm

- History
- Algorithm
- Analysis

Online algorithm

- Algorithm
- Analysis

Open questions

Randomized search heuristics

- Sometimes the (optimization) problem is too hard to solve exactly
  - $>99\%$ of problems from real world
- Sometimes it's hard to solve a problem good enough
  - simple/complex heuristics don't wish to work
- What to do?
  - one way: look how existing optimizers work and do the same
  - randomness is a good way to overcome the curse of the human factor
  - randomized search heuristics

Randomized search heuristics: what are they?

- Natural evolution: an optimizer which produced all living forms
  - genetic algorithms
  - evolution strategies
- "Swarm intelligence"
  - ant colony optimization
  - particle swarm optimization
  - "cuckoo search" and the other "zoological" algorithms
- Physics
  - simulated annealing
  - "intelligent water drops"

Randomized search heuristics: Principles

- It is hard to solve the problem, but it is often much easier to estimate the quality of a solution
  - compare P and NP problem classes
  - at least it is easy to tell if A is better than B
- It is not very complex to make small and large changes to solutions
  - and do it in randomized ways
- Motto: "change solutions until they are good enough"

What to do with multiple objectives?

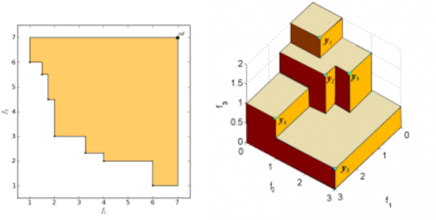
- Multi-objective optimization
  - it's generally impossible to find a single "best" solution
  - need to sample many enough Pareto-optimal solutions
- Most randomized search heuristics are designed to optimize only one objective

What to do with multiple objectives?

- First idea: Reduce the multiple objective problem to a single objective problem
- Early years:
  - weighed sum: optimize $\alpha_1 X_1 + \alpha_2 X_2 + \ldots + \alpha_k X_K$
  - lexicographic comparison: first compare X_1s then X_2s then ...
  - suffer from small diversity and sensitivity to scale/order

What to do with multiple objectives?

- Hypervolume indicator
  - The total volume of solutions which are dominated by the current solution set
  - Reflects both quality and diversity
  - $O(N \log N)$ in the two-dimensional case, NP-hard in general



Picture source: Dortmund University

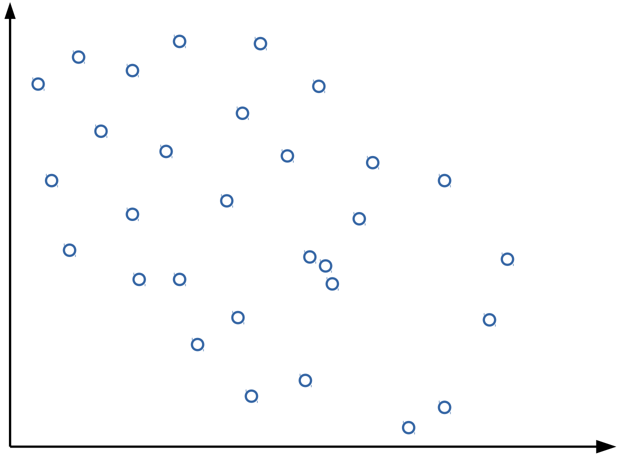
Pareto-oriented algorithms

- Some algorithms work with Pareto-optimal solutions only
  - given: a set of solutions S
  - returns: a subset of solutions $P \subset S$ which are not dominated by any solution from S
- Some algorithms want to know some information about non-optimal solutions as well
  - reason 1: information theory. By ignoring non-optimal solutions one gets less information per query
  - reason 2: good solutions can be constructed by recombination of not-so-good solutions

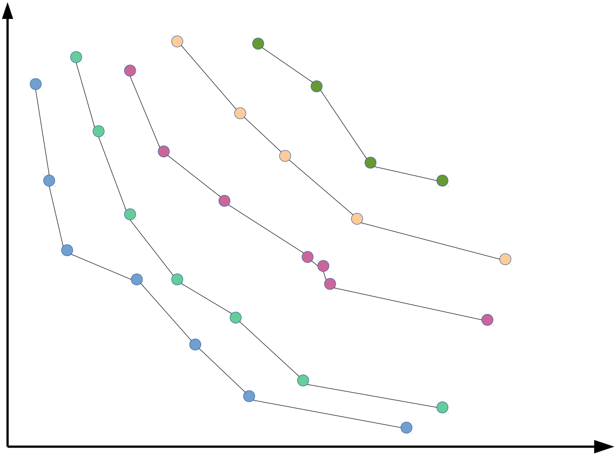
Non-dominated sorting

- Rank 0: those who are not dominated by any solution
- Rank 1: those who are not dominated by any solution except if rank $= 0$
- Rank 2: those who are not dominated by any solution except if rank ≤ 1
- $\ldots$

Non-dominated sorting

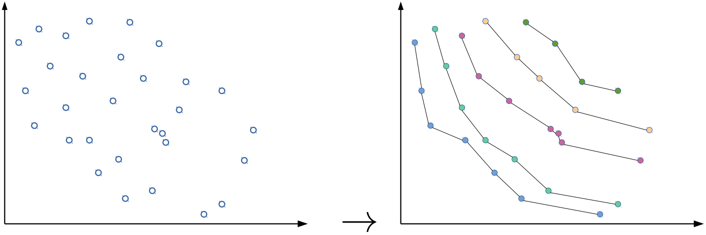


Non-dominated sorting



Part 1. Offline algorithm

- (From now on: solutions → points)
- Given N points, each of dimension K
- For each point, determine its rank



Some history

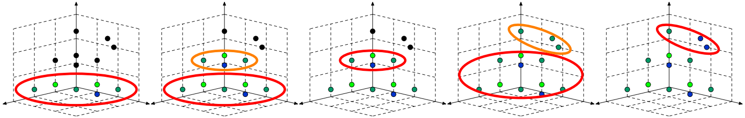
- 1975: Kung et al.
  - Finding rank-0 in $O(N \log^{K-1} N)$
- 2001: Deb et al. (part of NSGA-II)
  - Running time: $\Theta(N^2 K)$
- 2003: M. Jensen
  - $O(N \log^{K-1} N)$
  - No two objective values may coincide
- 2013 (!): Fortin et al.
  - Works in all cases
  - $O(N \log^{K-1} N)$ in average, $O(N^2 K)$ proven
- 2014: M. Buzdalov
  - $O(N \log^{K-1} N)$ in all cases

Divide-and-conquer approach

- S: points, $K \geq 3$: dimension
- If $|S| = 2$, just compare points
- Find a median M of the K-th objective over S
- $S = S_L \cup S_M \cup S_H$, where
  - S_L: points with $X_K < M$
  - S_M: points with $X_K = M$
  - S_H: points with $X_K > M$
- No point from S_M dominates any point from S_L
  - thus ranks of S_L don't depend on S_M
  - S_M, however, depends on S_L

Divide-and-conquer approach

- IdeaA(S, K):
  - $S = S_L \cup S_M \cup S_H$
  - IdeaA(S_L, K)
  - Update ranks of S_M using S_L
  - IdeaA(S_M, $K - 1$)
  - Update ranks of S_H using $S_L \cup S_M$
  - IdeaA(S_H, K)
- Both "update ranks" need $K - 1$ objectives

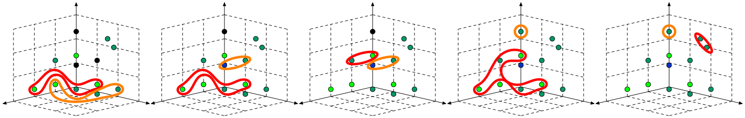


How to update ranks of B using A?

- Consider again $K \geq 3$
- Ranks of A are final
- $a \in A$ dominates $b \in B$ in objectives $> K$
- If $|A| = 1$ or $|B| = 1$, do full comparison
- Find a median M of K-th objective of $A \cup B$
- $A = A_L \cup A_M \cup A_H$
- $B = B_L \cup B_M \cup B_H$
- No need to:
  - update ranks of B_L using A_M or A_H
  - update ranks of B_M using A_H

Divide-and-conquer: again!

- Idea2(A, B, K):
  - $A = A_L \cup A_M \cup A_H$
  - $B = B_L \cup B_M \cup B_H$
  - Idea2(A_L, B_L, K)
  - Idea2(A_L, B_M, $K - 1$)
  - Idea2(A_M, B_M, $K - 1$)
  - Idea2($A_L \cup A_M$, B_H, $K - 1$)
  - Idea2(A_H, B_H, K)



Running time estimation

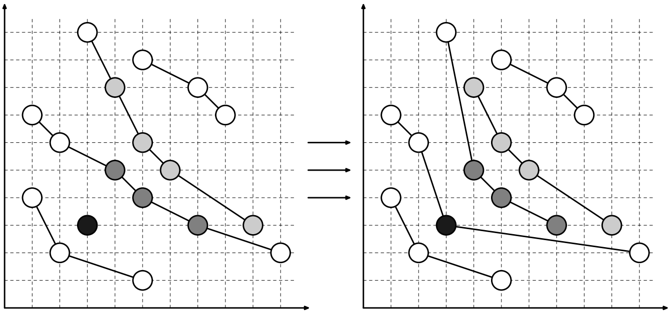
- Base cases: $K = 2$
  - Implemented using a sweep-line approach
  - $\text{Idea1}(N, 2) \rightarrow O(N \log N)$
  - $\text{Idea2}(A, B, 2) \rightarrow O((A + B) \log A)$
- Hypothesis
  - $T_{\text{Idea2}}(A, B, K) = O((A + B) \log^{K-1}(A + B))$
  - $T_{\text{Idea1}}(N, K) = O(N \log^{K-1} N)$
- Inductive proof
  - $S = S_L \cup S_M \cup S_H$
  - $2|S_L| \le |S|$, $2|S_H| \le |S|$
  - $A = A_L \cup A_M \cup A_H$, $B = B_L \cup B_M \cup B_H$
  - $2|A_L + B_L| \le |A + B|$, $2|A_H + B_H| \le |A + B|$
  - For K, calls of smaller levels are always $O(T \log^{K-2} T)$

Part 2. Online algorithm

- A set of points S which support queries:
  - `Add` a point P to S
  - `Find` the rank of a point P if it is inserted
  - `Get` a random point from S with its rank
  - `Delete` a point with the worst rank
- Straightforward: use $O(N \log^{K-1} N)$ algo
  - Add: $O(N \log^{K-1} N)$
  - Find: $O(N)$
  - Get: $O(1)$
  - Delete: $O(1)$
- ENLU approach (Deb et al, 2014): $O(N\sqrt{N}K)$ in average, $O(N^2K)$ worst
- How to be faster?

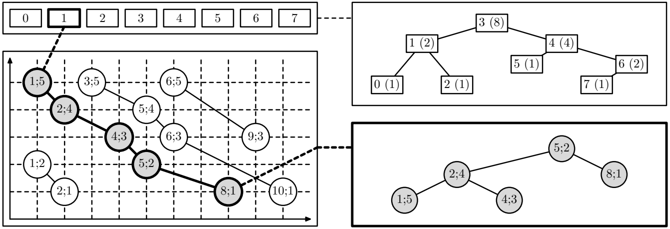
Incremental approach

- Layer: a set of points with the same rank
- Idea: during addition layers exchange a small number of contiguous fragments



Data structure: Tree of trees

- Right now we have an algorithm for $K = 2$ only
- The high-level tree: nodes are layers as low-level trees
- A lower-level tree: nodes are points from a layer



Runtime analysis

- Let there be N points and M layers
- Add: at most $O(N)$, more precisely:

$$O\left(M \log\left(1 + \log\frac{N}{M}\right) + \log M \log\frac{N}{\log M}\right)$$

- Find: $O\left(\log M \log \frac{N}{\log M}\right)$
- Get: $O(\log N)$
- Delete: $O(\log N)$

Open questions

- Offline algorithm
  - What is the theoretical lower bound on the complexity?
  - What is the two-parameter complexity of the proposed algorithm?
  - How to make it faster?
- Online algorithm
  - Extend to $K > 2$
  - What will be the running time?
    - current guess: $O(N \log^{K-2} N)$ for insertion
  - Learn how to compute statistics (e.g. densities of all sorts)
  - Make it faster