Generating Instances with Performance Differences for More Than Just Two Algorithms

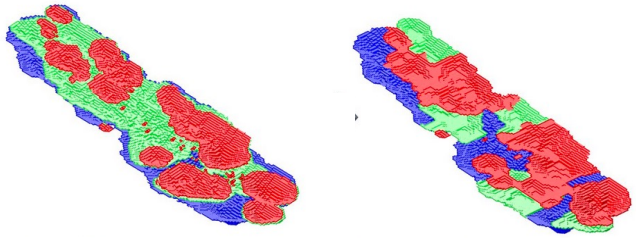
Jakob Bossek, Markus Wagner

jakob.bossek@wi.uni-muenster.de
markus.wagner@adelaide.edu.au

Code at https://github.com/jakobbossek/GECCO2021-ECPERM-ttp-evolving

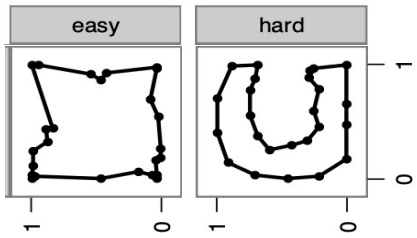
Diversity nowadays: show alternatives

Mine planning:
optimize w.r.t. known objectives (money, time, …) but then show alternative plans (e.g. sequences, …)



Inspirational image generation:
optimize quality & similarity (to a seed), but be diverse (search here via the latent space)

Algorithm understanding / algorithm tuning / algorithm portfolios / …:
generate X instances (*diverse* w.r.t. Y features) on which Z algorithms perform as *differently* as possible



Algorithm footprint

[Background 1/4] Performance Diversity of instances for the Travelling Salesperson Problem

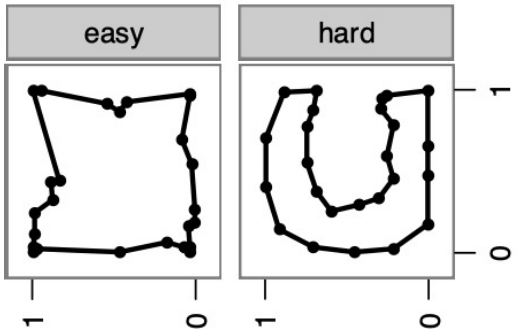
fundamental combinatorial problem: find the shortest tour across n cities

➔ In ~2010: we want to construct a set of TSP instances on which the performance of one algorithm varies

Examples:

- Diverse set where a certain algorithm is performing badly (high approximation ratio) $\alpha_A(I) = A(I)/OPT(I)$
- Diverse set where two solvers are performing differently (again: use performance ratios).

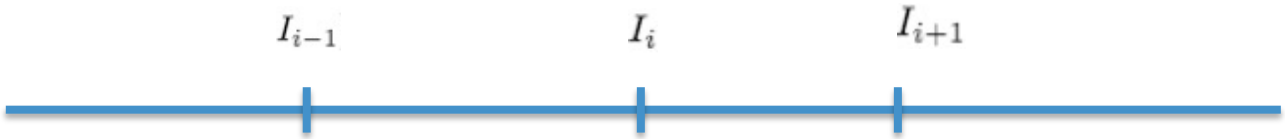
Here: 25 cities



[Background 2/4] Single-feature diversity measure

(also shows: going from 2 to 3+ can sometimes be challenging)

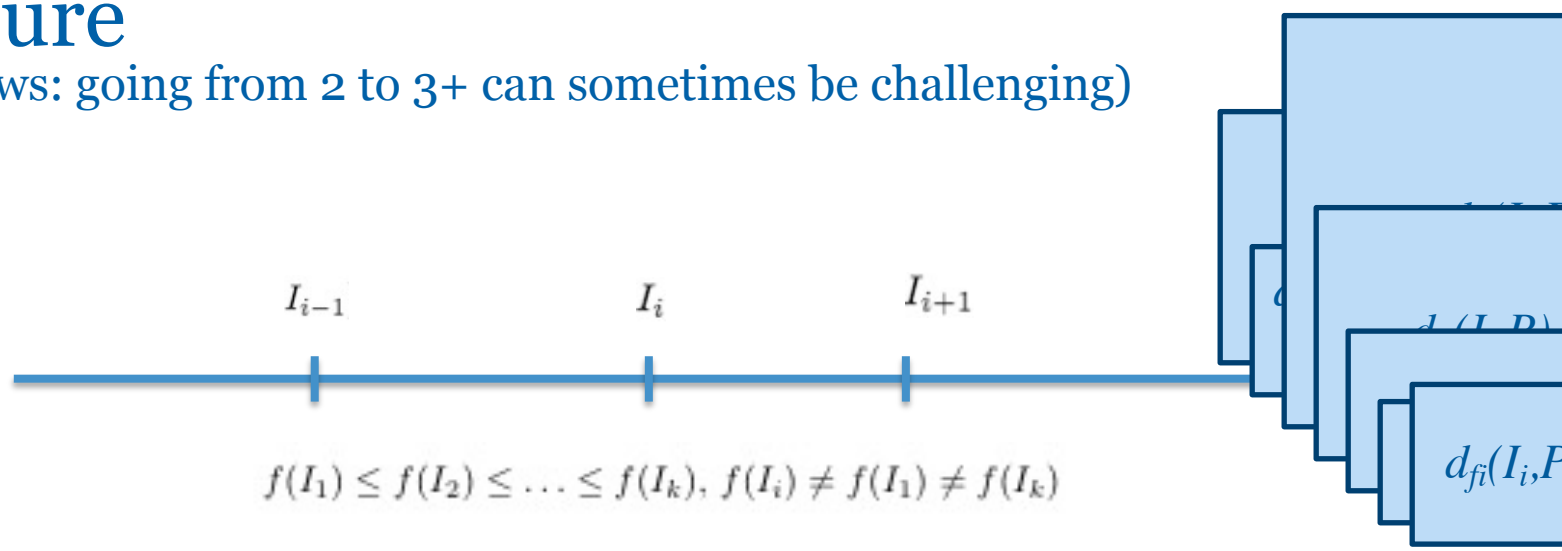
Feature f:



$$f(I_1) \leq f(I_2) \leq \ldots \leq f(I_k),\ f(I_i) \neq f(I_1) \neq f(I_k)$$

[Background 2/4] Single-feature diversity measure

(also shows: going from 2 to 3+ can sometimes be challenging)



Feature f:

$$f(I_1) \leq f(I_2) \leq \ldots \leq f(I_k),\ f(I_i) \neq f(I_1) \neq f(I_k)$$

"Diversity" of a single solution:

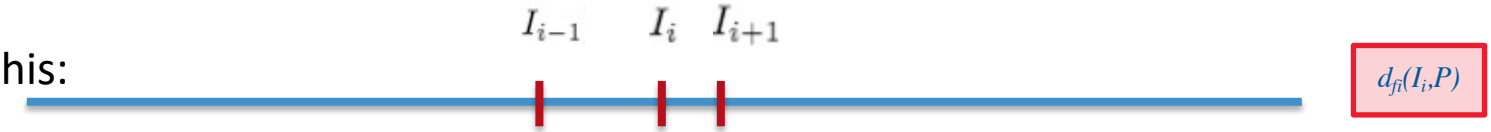
$$d_{f_i}(I_i, P) = (f(I_i) - f(I_{i-1})) \times (f(I_{i+1}) - f(I_i))$$

Diversity of a population:

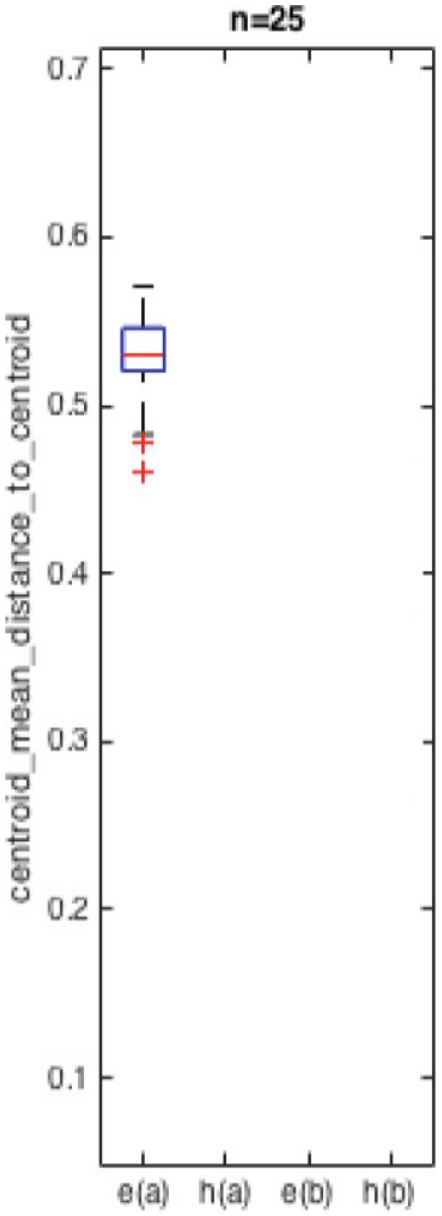
$$d'(I, P) = \sum_{i=1}^{k} (w_i \times d_{f_i}(I, P))$$

Maximum: if solutions are equally spaced out, as this is then the sum of squares

Compare with this:



[Background 3/4] Features of "easy/hard" TSP instances for ***2-opt*** (with and without diversity optimization)



Feature values of evolved instances:
From left to right:

1. Easy instances / only using **α**
2. Hard instances / only using **α**

} Conclude that this feature is important for instance difficulty (you might expect training an algorithm selector on this data)

3. Easy instances / feature diversity (**α** as quality constraint)
4. Hard instances / feature diversity (**α** as quality constraint)

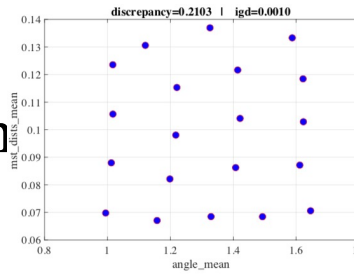
Works for other features, too… but not for all.

Long story short (here): old conclusions drawn re importance (for difficulty) invalidated by including feature diversity

[Background 4/4] … diversity optimization is a hot field

Our focus is typically on problem formulations…

- Feature diversity w.r.t. 2+ features (GECCO'19 BPA nomination)
  ➔ going from 1 to 2+ features has been challenging without favouring one feature (or value range or linear combination or …) over another



- Lots of theoretical runtime analyses by Frank Neumann et al. (GECCO'21 BPA nomination)
- More material: IEEE CEC 2021 tutorial
  https://cs.adelaide.edu.au/~optlog/EvolutionaryDiversityOptimisationTutorialCEC2021.php

➔ **The OPEN QUESTION this present paper aims to answer:**

How to evolve instances on which 3+ algorithms perform as differently as possible?

For example: I_1: $A_1 > A_2 > A_3$ I_2: $A_3 > A_1 > A_3$ I_3: $A_3 > A_2 > A_1$

Approach

What shall our
fitness function F be?

Algorithm 1: Outline of instance evolving (1 + 1)-EA.

```
input: Fitness function F
Initialize instance I randomly;
while budget not depleted do                ▷ Often time-limit
    Generate I′ by applying mutation to I;
    if F(I′) ≥ F(I) then
        Replace I with I′;
return I;
```

IDEA 1/3: Pairwise approach

Consider all N(N-1) ordered pairs of algorithms and evolve for each pair in individual runs.

➔

- Easy to formulate with existing tech.
- But: a run ignores all other N-2 algorithms, e.g., a resulting instance might be easy for all of these.

Approach

What shall our fitness function F be?

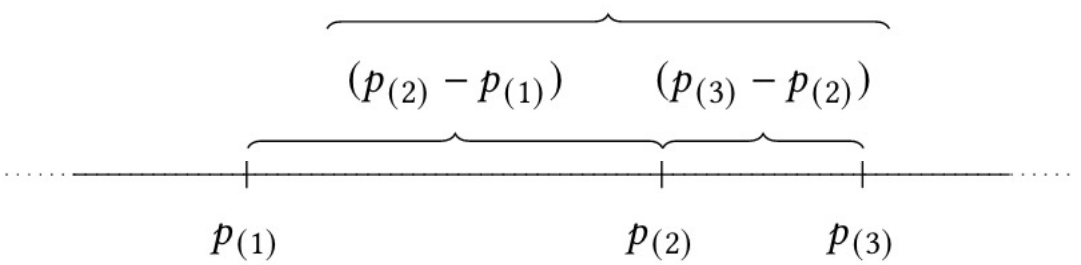
Algorithm 1: Outline of instance evolving (1 + 1)-EA.

```
input: Fitness function F
Initialize instance I randomly;
while budget not depleted do                ▷ Often time-limit
    Generate I' by applying mutation to I;
    if F(I') ≥ F(I) then
        Replace I with I';
return I;
```

IDEA 2/3: No order

Sort performances p_i and then maximise the crowding distance.

$$F(p_1, p_2, p_3) = (p_{(2)} - p_{(1)}) \cdot (p_{(3)} - p_{(2)})$$



➔

- Uses established technology (see [Background 3/5]).
- We need to get lucky to hit a desired permutation.

Approach

What shall our fitness function F be?

Algorithm 1: Outline of instance evolving (1 + 1)-EA.

```
input: Fitness function F
Initialize instance I randomly;
while budget not depleted do                ▷ Often time-limit
    Generate I′ by applying mutation to I;
    if F(I′) ≥ F(I) then
        Replace I with I′;
return I;
```

IDEA 3/3: Explicit ranking

Use a three-tuple to implement two "phases":

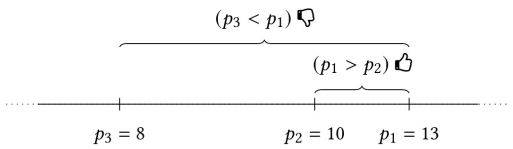
1. Phase: match the desired ranking
2. Phase: maximise the performance difference

Some details:

Let $p_1,\ldots,p_N$ be the performance values of the algorithms, and let π be the desired ranking.

Good directions: $G=\{(i,i+1)|p_{\pi(i)} \geq p_{\pi(i)}\}$

Bad directions: $B=\{(i,i+1)|p_{\pi(i)} < p_{\pi(i)}\}$



→ Maximise lexicographically: $F(p_1,\ldots,p_N; \pi) = (\,|G|, f_B, f_G\,)$

where f_B is the sum of the distances in the bad pairs, and f_G is the sum of the distances in the good pairs.*

Local sensitivity is necessary!

Case Study

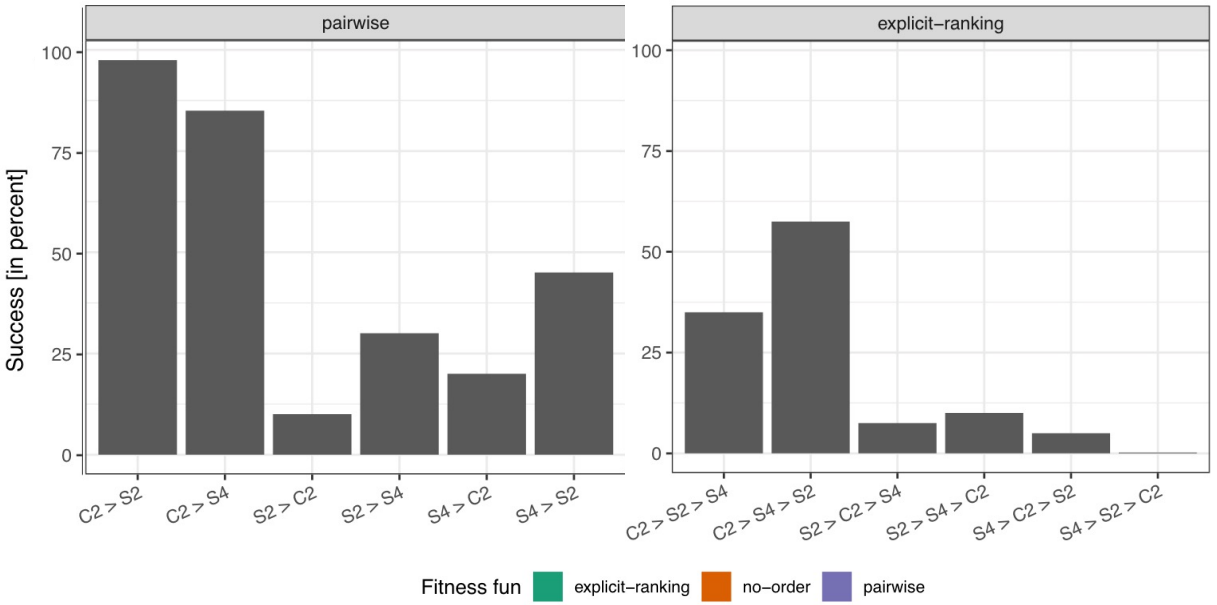
Target problem: Travelling Thief Problem
(partly “find a permutation for the travelling” and partly “find a bitstring for a knapsack”)

Target algorithms:
- S2 (“simple”): targets bitstring
- S4 (“simple”): targets permutation
- C2 (“complex”): targets bitstring & permutation alternatingly

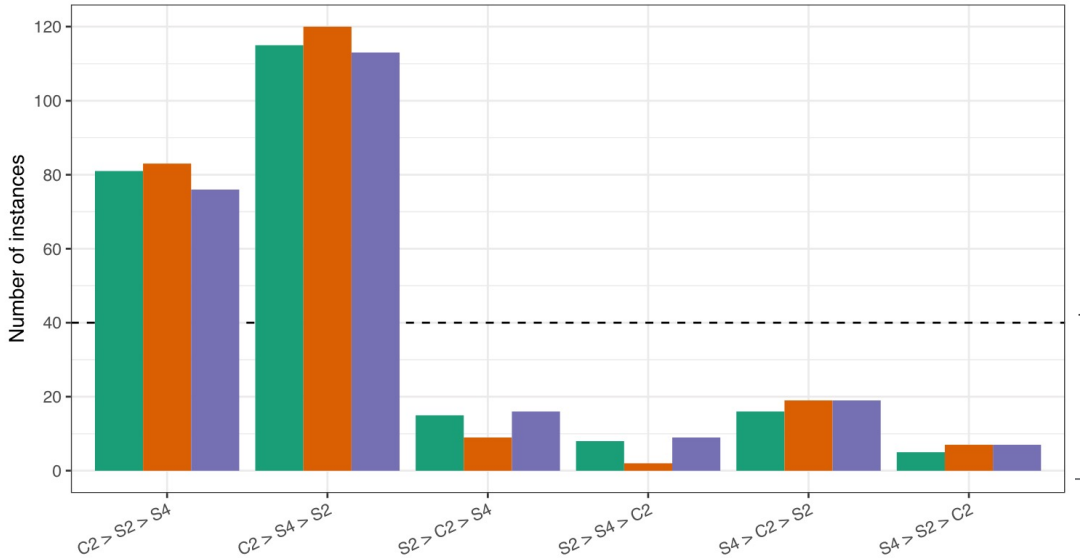
Instance evolution:
- simple (1+1)EA with disruptive operators
- instances with 200 nodes and with 200, …, 2000 items
- performance on an instance:
  - During evolution: median of 5 runs
  - After evolution: run all three algorithms 30 times

Results: desired vs. actual rankings

% of successful jobs

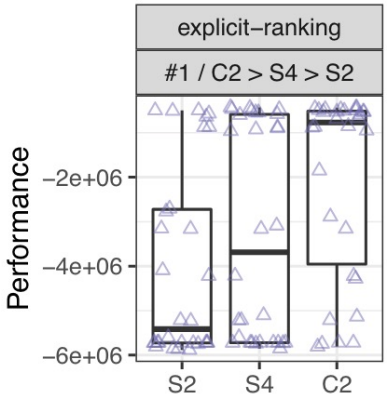


# of instances evolved

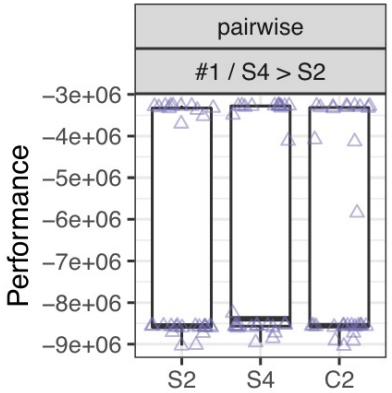


Results: Investigating issues

1) General noise…
(30*3=90 triangles show the raw performance)



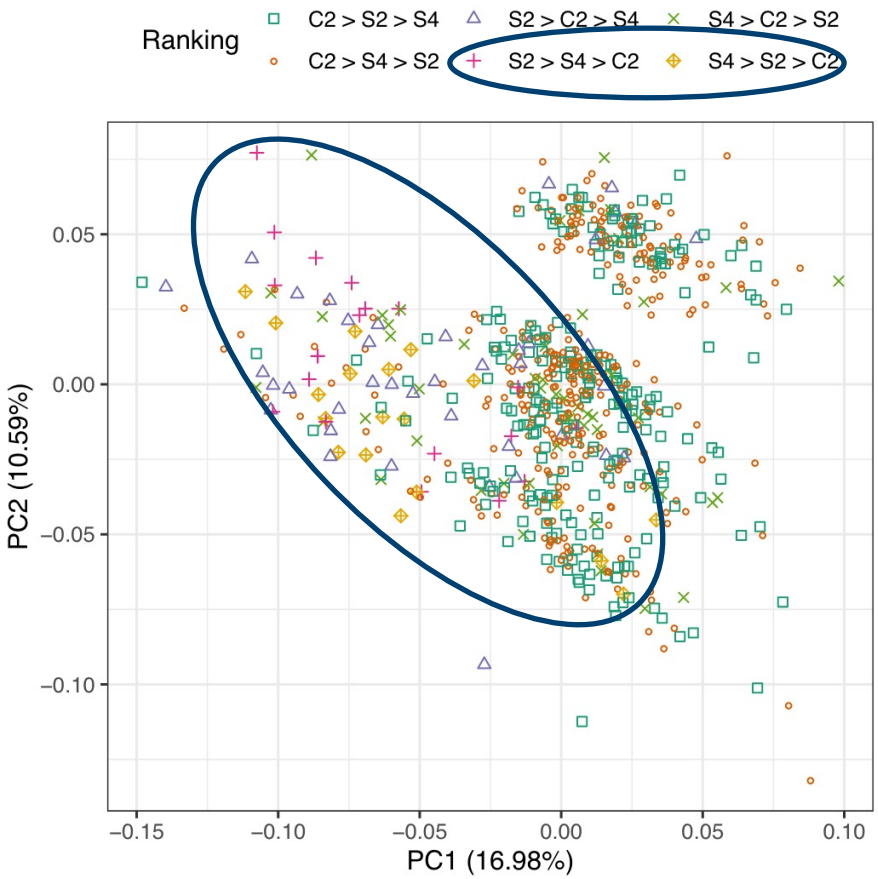
… sometimes bi-modal behaviour



→ Address 1) and 2) by using more repetitions during evolution… but this costs computation time.

2) The complex C2 heuristic is “more powerful” than the other two and tends to dominate.

Results: Properties of instances (1/2)
PCA in the "instance feature"-space



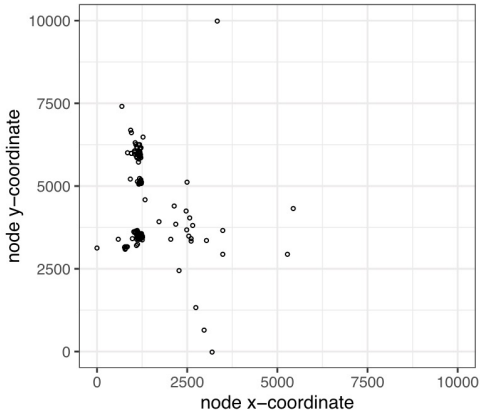
→ For some rankings: instances appear in distinct parts of the "instance feature"-space. But remember [Background 3/5]: we need to be careful when drawing conclusions from such observations, as we have *only* optimised performance rankings.

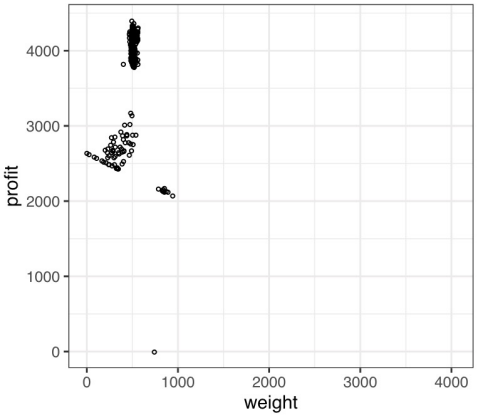
Results: Properties of instances (2/2)
Two Examples

TSP: node coordinates

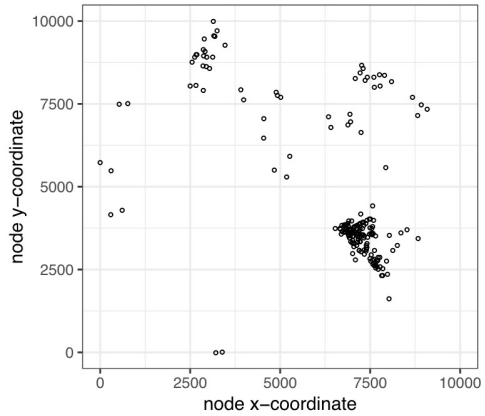
knapsack items: weight & profit

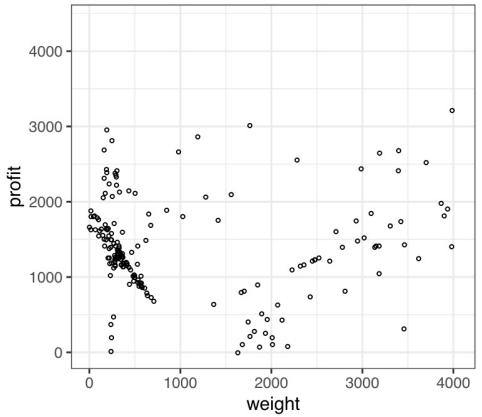
S2 > S4 > C2





C2 > S4 > S2





Summary: Diversity — when optimisation is not the goal

10 years ago: “evolve easy/hard instances for one algorithm”

Now: *”evolve solutions (== ‘actual’ solutions, instances, …) that are diverse in multi-dimensional spaces (‘actual’ feature space, performance space, …), possibly subject to minimum quality constraints”*

What is next???

- Optimisation: study custom variation operators [we learned that we have to be disruptive in the encoding space]
- Your solvers: how to change your state-of-the-art solvers to compute diverse sets of solutions (instead of a single one)? [beneficial to the end user, but maybe also during search?]
- Your domains: can ‘diversity’ be helpful to you, if so, why and how?

Email: jakob.bossek@wi.uni-muenster.de markus.wagner@adelaide.edu.au

Papers + code:
http://www.jakobbossek.de/
https://cs.adelaide.edu.au/~markus/publications.html
https://cs.adelaide.edu.au/~frank/publications.html