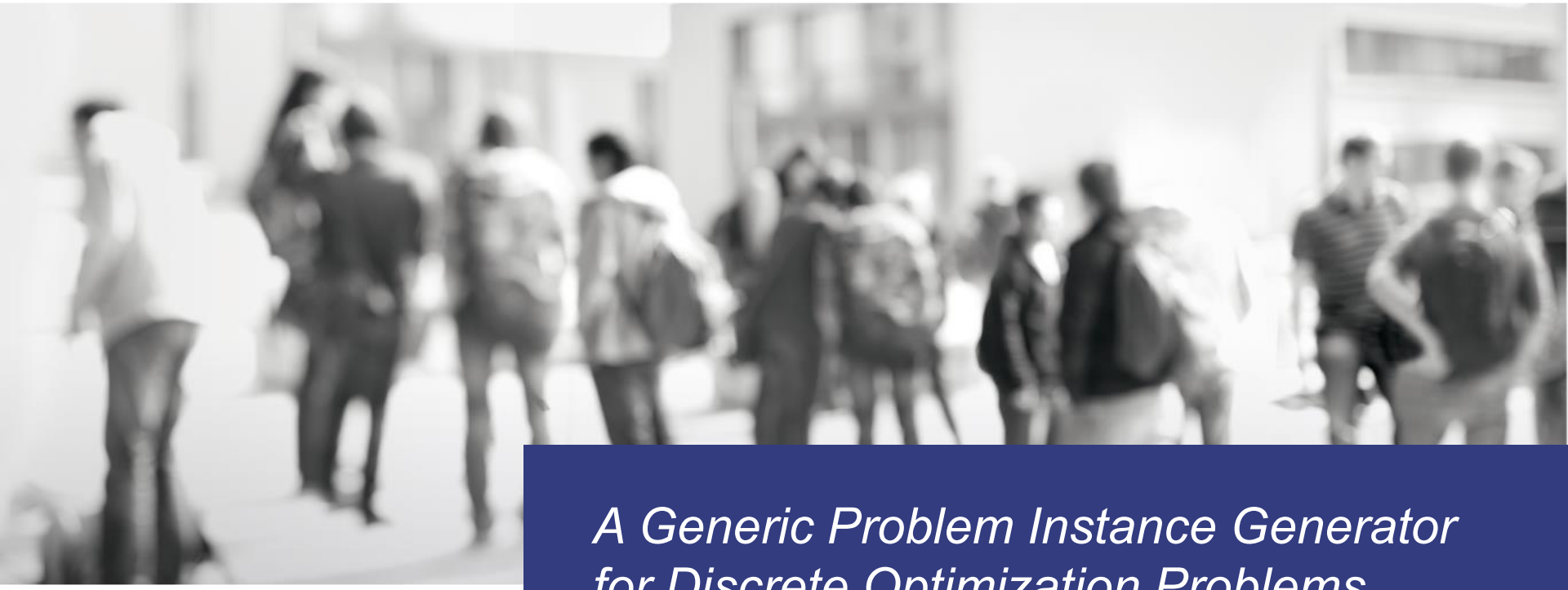
Hochschule Zittau/Görlitz
UNIVERSITY OF APPLIED SCIENCES

合肥學院
HEFEI UNIVERSITY



A Generic Problem Instance Generator for Discrete Optimization Problems

Markus Ullrich, Thomas Weise, Abhishek Awasthi and Jörg Lässig

Motivation

- Suite of Benchmark Problems for Black-Box Discrete Optimization
  - Comparing Optimization Algorithms
  - Using Specific Problem Instances
- Necessity to generate new instances
  - Previously as „hard“ classified instances now solved to optimality[1]
  - Insufficient „real“ data available

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html

Instance Format Example

i	x_1	x_2	…	x_n
„id“	„integer“	„integer“		„$x_1 + x_2$“
1	353	12		365
2	121	67		188
…	…	…		…
M	982	86		1068

- Typically one or multiple matrices / CSV file(s)
- Data types per column
- Attributes might depend on each other

Instance Generation Requirements

- Common Format:
  - Different data types (columns)
  - Multiple data entries (rows)
  - Dependencies
- Blueprint for Generation:
  - Human-readable
  - Re-usable
  - Reproducible results
  - Generic and easy to extend

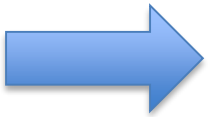
Instance Generator: General Configuration

Component	Example	Default	Optional
„version“	„v0.1“	„latest“	yes
„seed“	-82651876583214	<random value>	yes
„rows“	100000	-	no
„no_duplicates“	true	false	yes
„attributes“	[…]	-	no
„constraints“	[…]	-	no

- „seed“ allows for reproducability
- „version“ only necessary for old features

no_duplicates Example

```
"rows": 8,
"no_duplicates": true,
...
{   "name": "x",
    "type": "integer",
    "min": 0,
    "max": 1,
    "output": true }
{   "name": „y",
    ... }
{   "name": „z",
    ... }
```



```
x;y;z
1;1;0
1;1;1
1;0;1
0;0;1
0;0;0
0;1;0
0;1;1
1;0;0
```

- *y* and *z* are configured like *x*
- „rows“ > 8 is not valid

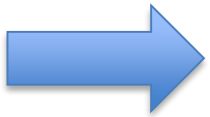
Instance Generator: Attribute Configuration

Component	Example	Default	Optional
„name“	„x“	-	no
„type“	„double“	-	no
„use_all_values“	true	false	yes
„output“	true	true	yes
„output_probability“	0.3	-	yes
„seed“	35654876534	<global seed>	yes

- „use_all_values“ is a special constraint
- „output_probability“ for varying row length

use_all_values Example

```
"rows": 10,
...
"name": "x",
"type": "integer",
"min": 0,
"max": 9,
"use_all_values": true,
"output": true
...
```



```
x;y;z
3;0;0
7;0;0
1;0;1
2;0;0
5;1;1
8;0;0
0;1;0
4;0;1
6;1;1
9;1;0
```

- Every integer value within the provided boundaries is generated.
- „rows“ < 10 is not valid

Instance Generator: Attribute Type Configurations

Component	Example	Used for Type
„min“ and „max“	10000.0	„integer“ and „double“
„default“ and „value“	500.0	
„expression“	„5 * y“	
„start“, „increment“ or „min-“ and „maxIncrement“	100.0	„id“
„mean“ and „standard_deviation“	10.0	„Gaussian“
„true“	0.3	„boolean“

- Any complex „expression“ is also possible, e.g.: $\frac{x_1 * x_2}{y_1}$
- Requires additional „helper“ attributes

Instance Generator: Constraint Configuration

- Constraints define relations between attributes
- For relations, a left and right part can be defined
- Both parts can be any type of expression

Component	Example Values	Type
„relation“	„<“, „>“, „=“, „<=“, „>=“, „!=“	-
„left“ and „right“	300, 12.5	„value“
	„x“, „y“, „x1“, „y2“	„attribute“
	„avg(x1)“, „min(y2)“, „abs(j)“	Aggregate „expression“
	„x1 / 2“, „y1 + y2“, „100 - j“	Arithmetic „expression“

Constraint Validation

1. Check for no circular dependencies

For every row:

2. Check Boundaries (can fail on the first row)

   For every constraint:

   a. Evaluate „left“ and „right“
   b. Check if expression holds

3. Check for no duplicates

   a. Compare every previous row with the current

Circular Dependency Check

- Topological node sorting method developed by A.B. Kahn [1]:

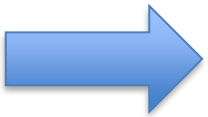
```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edge
while S is non-empty do
        remove a node n from S
        add n to tail of L
        for each node m with an edge e from n to m do
                remove edge e from the graph
                if m has no other incoming edges then
                        insert m into S
if graph has edges then
        return error (graph has at least one cycle)
Else
        return L (a topologically sorted order)
```

[1] https://doi.org/10.1145/368996.369025

Demo: TSP and Max-SAT

TSP Example Configuration

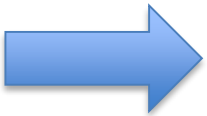
```
{   "no_duplicates": true,
    ...
    "attributes": [
        {   "name": "id",
            "type": "id",
            "start": 1,
            "increment": 1,
            "output": true },
        {   "name": "x",
            "type": "double",
            "min": 0.0,
            "max": 10000.0,
            "output": true },
        {   "name": "y",
            ... }],
    "constraints": [
]}
```



```
id;x;y
1;9815.99;6830.20
2;3021.45;4173.25
3;4185.60;8340.02
4;7201.43;6169.73
5;8069.81;4958.80
6;6009.31;2321.92
...
```

Max-SAT Example Configuration

```
{   "no_duplicates": true,
    "separator": " ",
    "comment_prefix": "c",
    "alternative_header": "p cnf 4 100",
    ...
    "attributes": [
        {   "name": "i",
            "type": "integer",
            "min": -4,
            "max": 4,
            "output": true },
        {   "name": „j",
            ... },
        {   "name": "k",
            ...
            "output_probability": 0.3 },
        {   "name": "zero",
            "type": "integer",
            "value": 0 }
    ],
```



```
p cnf 4 100
-1 3 0
4 -2 0
-2 1 -3 0
...
```

```
"constraints": [
        {   "name": "i!=j",
            "left": {
                "type": "expression",
                "value": "abs(i)" },
            "relation": "!=",
            "right": {
                "type": "expression",
                "value": "abs(j)" }},
        {   "name": "k!=j",
            ... },
        {   "name": "k!=i",
            ... },
        {   "name": "no_i_zero",
            "left": {
                "type": "attribute",
                "value": "i" },
            "relation": "!=",
            "right": {
                "type": "integer",
                "value": 0 }},
        {   "name": "no_j_zero",
            ... },
        {   "name": "no_k_zero",
            ... }]}
```

Load Allocation Example

Given any number of customers and suppliers, let:

n = total number of customers,

τ_i = total time for which customer i needs power,

d_i = power demand for $\tau_i, \tau_i \geq 0 \; \forall \; i$,

E_i = earliest possible power feeding time for customer i,

L_i = latest possible power feeding time, $L_i - E_i \geq \tau_i$,

t = time of day, based on 15 min intervals,

P_t = power available at time $t, P_t \geq 0 \; \forall \; t$,

α^t = penalty for not meeting the total demand at time t,

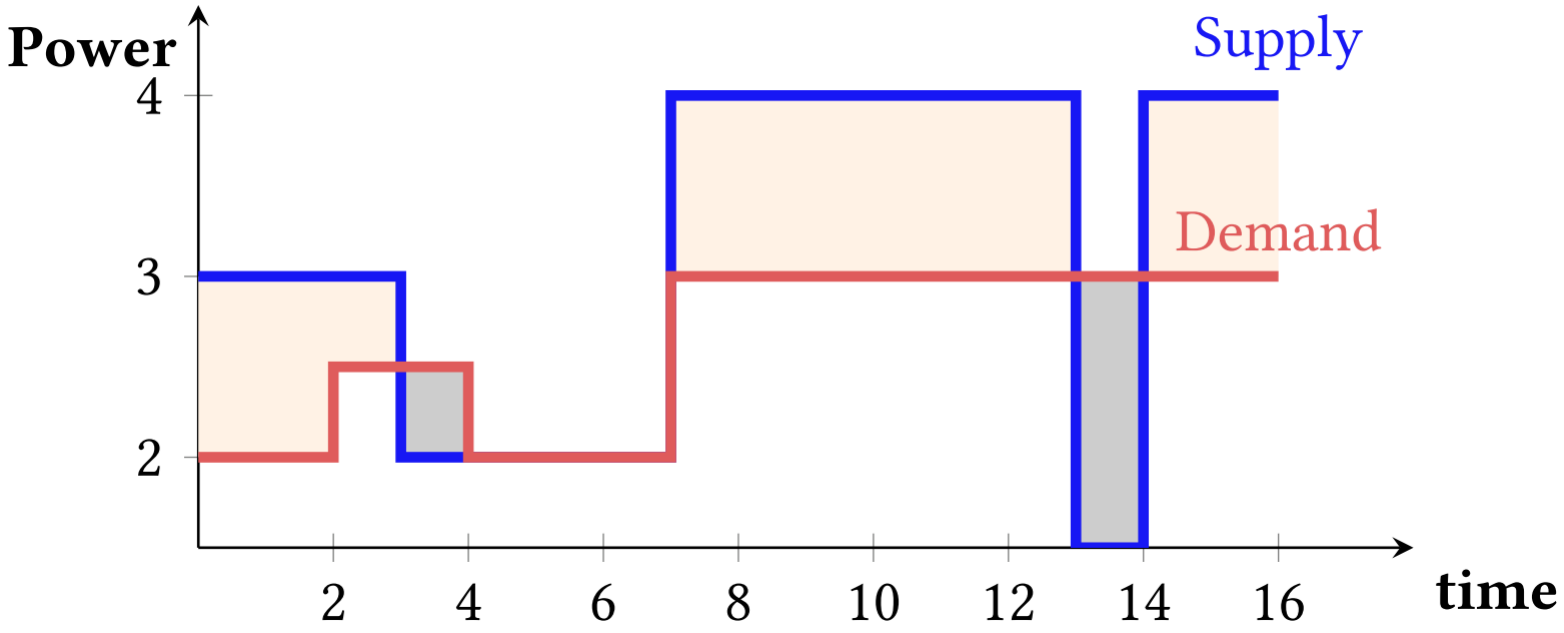
β_i = penalty for exc. power consumption of customer i,

θ_i^t = power available at time $t, P_t \geq 0 \; \forall \; t$,

Load Allocation Example

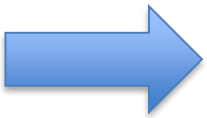
Objective function:

$$\min\left\{\sum_t \alpha^t \cdot \max\left\{0, P_t - \sum_i \theta_i^t\right\} + \sum_i \sum_t \beta_i \cdot \max\left\{0, \theta_i^t - P_t\right\}\right\}$$



Customer Example Configuration

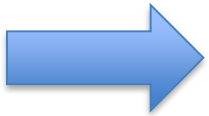
```
{   "rows": 10,
    "no_duplicates": true,
    ...
    "parameters": [

    ],
    "attributes": [
        {   "name": "n",
            "type": "id",
            "start": 1,
            "increment": 1,
            "output": true
        },
        {   "name": "E_i",
            "type": "integer",
            "min": 0,
            "max": 70,
            "output": true },
        {   "name": "L_i",
            "min": 20,
            "max": 95,
            ... },
        {   "name": "tau_i",
            "min": 2,
            "max": 20,
            ... },
        {   "name": "d_i",
            "min": 1,
            "max": 10,
            ... },
        {   "name": "beta_i",
            "min": 1,
            "max": 10,
            ... }
    ],
    "constraints": [
        {   "name": "L_i >= E_i + tau_i",
            "left": {
                "type": "attribute",
                "value": "L_i" },
            "relation": ">=",
            "right": {
                "type": "expression",
                "value": "E_i + tau_i" }
}]}
```



```
n;E_i;L_i;tau_i;d_i;beta_i
1;70;81;5;7;3
2;4;40;10;6;4
3;20;67;7;10;3
```

Supplier Example Configuration

```
{   "rows": 10,
    "no_duplicates": true,
    ...
    "attributes": [
        {   "name": "t",
            "type": "id",
            "start": 0,
            "minIncrement": 2,
            "stop": 95,
            "output": true },
        {   "name": "P_t",
            "type": "integer",
            "min": 0,
            "max": 10,
            "output": true },
        {   "name": "alpha_t",
            "min": 1,
            "max": 10,
            ... }],
    "constraints": []
}
```



```
t;P_t;alpha_t
0;0;2
9;7;8
16;2;9
24;6;1
30;1;10
37;6;8
42;5;10
55;8;6
72;7;2
95;1;5
```

Conclusion

- Generator Properties:
  - Straightforward blueprint creation
  - Flexible, re-usable, reproducable results
  - Creates random instances:

    → Unknown quality characteristics
- Future Work:
  - Complex expressions and constraints
  - CLI for bulk generation support
  - Any suggestions are welcome

Thank you for your attention!

Any Questions?

https://github.com/mullrichHSZG/BBDOB_problem-generator