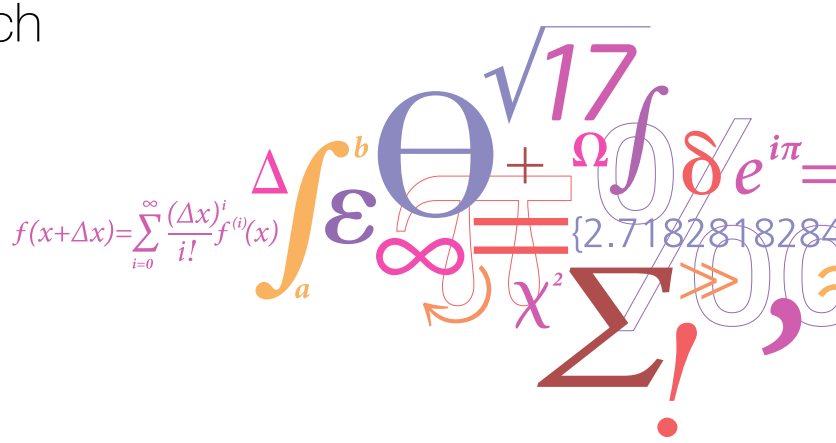
University Timetabling

Curriculum-based Course Timetabling
using ALNS and Tabu Search



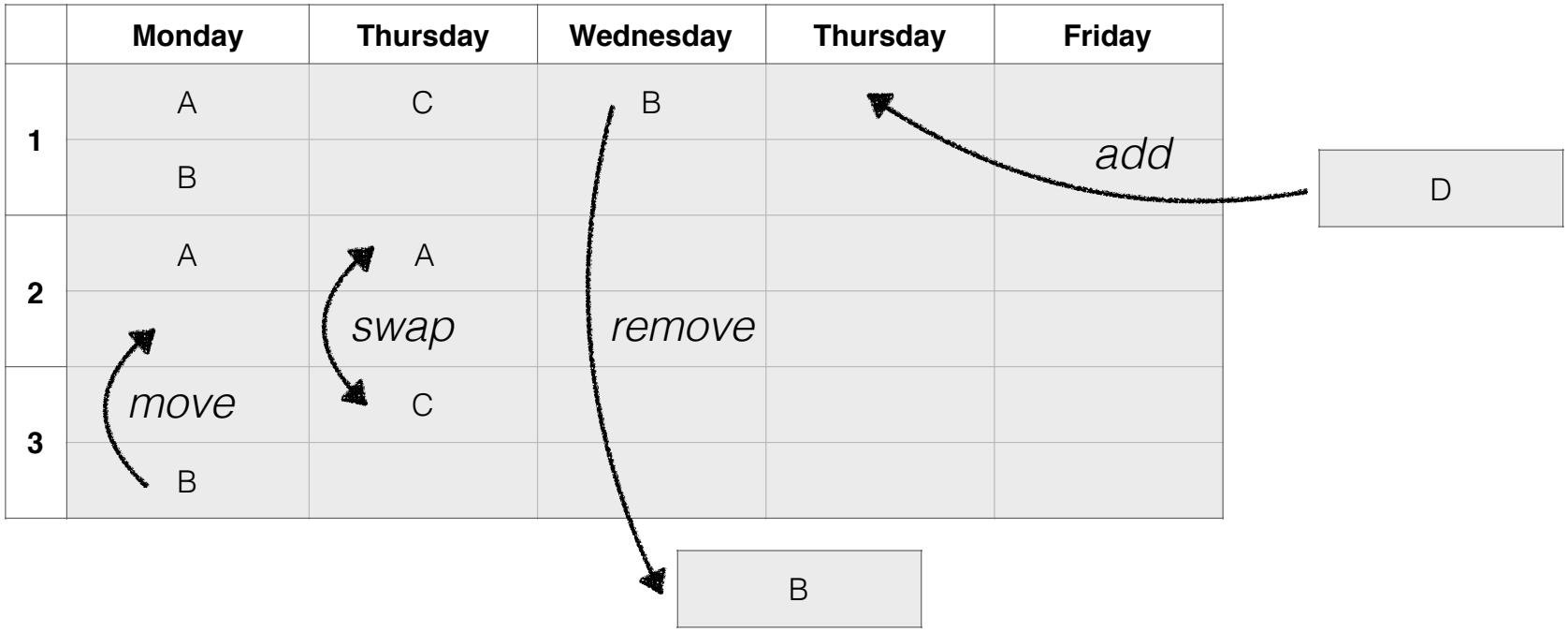
DTU Compute
Department of Applied Mathematics and Computer Science

The Meta Heuristics

- Tabu Search: *good for large neighborhood*

- ALNS: *good for highly constrained problem*

Operations


Monday
Thursday
Wednesday
Thursday
Friday
1
2
3
A
B
C
A
B
A
C
B
move
swap
remove
add
D
B

Initialization

all missing courses in random order

B	A	B	D	A	C	B

all available slots in random order

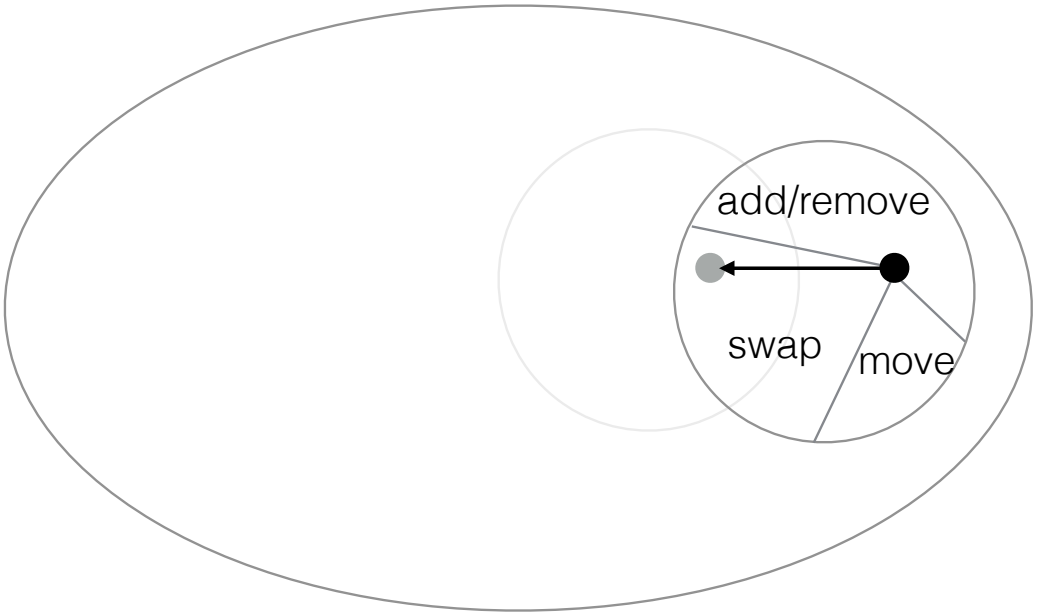
Mon	Tue	Mon	Wen	Fri	Mon	Tue
1	2	2	2	1	2	1
R1	R2	R2	R2	R2	R1	R1

for all c in courses **do**
  for i from 1 to LENGTH($slots$) **do**
    $(t, r) \leftarrow$ POPRIGHT($slots$)

    ▷ add (c, t, r) to solution if it improves the objective
    $\Delta \leftarrow$ SIMULATEADD(c, t, r)
    if $\Delta < 0$ **then**
      MUTATEADD(c, t, r) ▷ Use slot
      break
    else
      PUSHLEFT((t, r) on $slots$) ▷ The slot is still avaliable

Tabu Search

Tabu Search



Algorithm 4 Generalization of the Tabu search algorithm

```
1  function TabuSearch(solution_init)
2      s_global ← solution_init                          ▷ Globally best solution
3      s_local ← solution_init                           ▷ Current solution
4      tabu ← LimitedSet()                               ▷ May have infinite space
5
6      repeat
7          Δ, move ← LocalSearch(s_local, tabu)          ▷ Find best move ∉ tabu
8          if Δ < 0 then
9              s_local ← Apply(move on s_local)
10             Add(Opposite(move) on tabu)
11         else
12             if s_global hasn't been updated for awhile then
13                 s_local ← Intensify(s_global)         ▷ Intensification is optional
14             s_local ← Divserify(s_local)              ▷ Diversification is optional
15         if Cost(s_local) < Cost(s_global) then
16             s_global ← s_local
17     until no more time
18     return s_global
```

Tabu Search

name	type	description
diversification	integer	How many $(course, time, room)$ combinations should be removed. May be zero to disable diversification.
intensification	integer	Same as in the generalized tabu search.
tabu limit	integer	This parameter controls the tabu limit of all the tabu lists.
allow swap	{always, dynamic, never}	If *always* the swap neighborhood is always checked. If *never* the swap neighborhood is never checked. Additionally if *dynamic*, the swap neighborhood is only checked if none of the other operations could reduce the objective.

Table 3: Parameters for specialized Tabu search

Tabu Search

name	type	description
diversification	integer	How many $(course, time, room)$ combinations should be removed. May be zero to disable diversification.
intensification	integer	Same as in the generalized tabu search.
tabu limit	integer	This parameter controls the tabu limit of all the tabu lists.
allow swap	{always, dynamic, never}	If *always* the swap neighborhood is always checked. If *never* the swap neighborhood is never checked. Additionally if *dynamic*, the swap neighborhood is only checked if none of the other operations could reduce the objective.

Table 3: Parameters for specialized Tabu search

parameter	search space	value
allow swap	$\{\text{never}, \text{always}, \text{dynamic}\}$	dynamic
tabu limit	$\{10, 20, 40, \infty\}$	40
intensification	$\{2, 10, \infty\}$	10
diversification	$\{0, 1, 5\}$	5

Table 9: Best Tabu search parameters with $\mu = 4.139$ and $\sigma = 0.122$

Tabu Search

		intensification		
		2	10	∞
diversification	0	(4.90, 0.52)	(5.36, 0.37)	(5.53, 0.71)
	1	(5.07, 0.97)	(5.34, 0.17)	(5.29, 0.63)
	5	(5.19, 0.41)	(4.14, 0.12)	(5.06, 0.77)

Table 7: Shows (μ, σ) with `allow_swap=dynamic` and `tabu_limit=40` fixed

		tabu_limit			
		10	20	40	∞
allow_swap	never	(5.53, 0.70)	(5.91, 0.89)	(6.52, 0.60)	(5.93, 0.43)
	always	(9.12, 0.18)	(8.77, 0.26)	(9.16, 0.87)	(8.82, 0.44)
	dynamic	(5.52, 0.69)	(5.10, 0.59)	(4.14, 0.12)	(5.45, 0.27)

Table 8: Shows (μ, σ) with `diversification=5` and `intensification=10` fixed

ALNS

ALNS

Destroy

- Fully Random - *Remove random combinations*
- Curriculum - *Only sample from a random curriculum*
- Day - *Only sample from a random day*
- Course - *Only sample from a random course*

Repair

- Very Greedy - *Insert missing courses at the first placement with $\Delta < 0$.*
- Best Placement - *For each course sort placement by Δ, then add the requested amount.*

ALNS

In this case there are no invalid destroy or repair methods.

$$\Psi = \max\{w_{global}, w_{current}\}$$

Algorithm 6 Generalization of the ALNS search algorithm

```
1  function AlnsSearch(solution_init)
2      s_global ← solution_init                          ▷ Globally best solution
3      s_local ← solution_init                           ▷ Current solution
4      p⁺, p⁻ ← vector of 1s
5
6      repeat
7          d ← SampleFunction(p⁻)
8          r ← SampleFunction(p⁺)
9          s_local ← Repair(Destroy(s_local, d), r)
10
11         Ψ ← max{w_global, w_current, w_accept, w_reject}
12         p_d⁻ ← λp_d⁻ + (1 − λ)Ψ
13         p_d⁺ ← λp_d⁺ + (1 − λ)Ψ
14
15         if Cost(s_local) < Cost(s_global) then
16             s_global ← s_local
17     until no more time
18     return s_global
```

ALNS

name	type	description
λ	ratio $\in [0, 1]$	remember parameter used in the moving average update of the probabilities.
w_{global}	positive integer	reward for a globally better solution
$w_{current}$	positive integer	reword for a locally better solution
$remove$	positive integer	number of courses removed in each destroy function

Table 5: Parameters for generalized ALNS search

parameter	search space	value
λ	$\{0.9, 0.95, 0.99\}$	0.99
w_{global}	$\{5, 10, 20\}$	10
$w_{current}$	$\{1, 3, 4, 10\}$	10
remove	$\{1, 3, 5\}$	1

Table 12: Best ALNS parameters with $\mu = 0.3502$ and $\sigma = 0.0594$

ALNS

		remove		
		1	3	5
update_lambda	0.9	(0.62, 0.04)	(0.58, 0.02)	(0.99, 0.04)
	0.95	(0.46, 0.13)	(1.09, 0.05)	(1.22, 0.05)
	0.99	(0.35, 0.06)	(1.86, 0.04)	(1.72, 0.15)

Table 10: Shows (μ, σ) with `w_global=10` and `w_current=10` fixed

		w_current			
		1	3	5	10
w_global	5	(0.45, 0.15)	(0.55, 0.14)	(0.67, 0.04)	(0.39, 0.09)
	10	(0.42, 0.12)	(0.58, 0.01)	(0.55, 0.06)	(0.35, 0.06)
	20	(0.52, 0.18)	(0.40, 0.11)	(0.43, 0.11)	(0.58, 0.10)

Table 11: Shows (μ, σ) with `update_lambda=0.99` and `remove=1` fixed

Compareing

		Tabu	ALNS
train	1	(3.00, 0.84)	(0.22, 0.18)
	3	(3.07, 0.08)	(0.23, 0.15)
	5	(0.14, 0.04)	(0.07, 0.05)
	7	(6.33, 0.22)	(0.20, 0.11)
	9	(3.19, 0.24)	(0.13, 0.09)
	11	(12.80, 2.45)	(0.80, 0.81)
	13	(3.90, 0.17)	(0.11, 0.11)
test	2	(3.17, 0.27)	(0.09, 0.11)
	4	(4.88, 0.35)	(0.09, 0.08)
	6	(5.06, 0.35)	(0.05, 0.05)
	8	(5.19, 0.43)	(0.12, 0.07)
	10	(6.22, 0.39)	(0.14, 0.07)
	12	(0.92, 0.10)	(0.06, 0.06)
all train		(4.63, 0.46)	(0.25, 0.14)
all test		(4.24, 0.15)	(0.09, 0.04)

Table 13: Test and train results over 5 runs using best parameters

Conclusion

- Dynamic neighborhood is a good strategy.
- The ALNS sub functions should have approximately same speed.
- It is a long line of good choices, that makes a good solution.
- It was more important to search wide and suboptimal than narrow and optimal.