Traveling Tournament Problem

Simulated Annealing

Philippe Tanghe Li Quan

30 maart 2011

Inhoudsopgave

Inhoudsopgave

Inleiding

- Aris Anagnostopoulos, Laurent Dominique Michel, Pascal Van Hentenryck en Yannis Vergados.
  A simulated annealing approach to the traveling tournament problem. (2006)

Inleiding

- Aris Anagnostopoulos, Laurent Dominique Michel, Pascal Van Hentenryck en Yannis Vergados.
  A simulated annealing approach to the traveling tournament problem. (2006)
- Pascal Van Hentenryck en Yannis Vergados.
  A Traveling Tournament Scheduling: A Systematic Evaluation of Simulated Annealing. (2006)

Inleiding

- Aris Anagnostopoulos, Laurent Dominique Michel, Pascal Van Hentenryck en Yannis Vergados.
  A simulated annealing approach to the traveling tournament problem. (2006)
- Pascal Van Hentenryck en Yannis Vergados.
  A Traveling Tournament Scheduling: A Systematic Evaluation of Simulated Annealing. (2006)

- Traveling Tournament Simulated Annealing Algorithm (TTSA)
  - SA: goede metaheuristiek voor TTP
  - duidelijk en concreet
  - P. Van Hentenryck is een Belg

Inhoudsopgave

Basisalgoritme SA

```
find random schedule S;
bestSoFar ← cost(S);
phase ← 0;
while phase ≤ maxP do
    counter ← 0;
    while counter ≤ maxC do
        select a random move m from neighborhood(S);
        let S′ be the schedule obtained from S with m;
        Δ ← cost(S′) - cost(S);
        if Δ < 0 then
            accept ← true;
        else
            accept ← true with probability exp(−Δ/T);
        end
        if accept then
            S ← S′;
            if cost(S′) < bestSoFar then
                counter ← 0; phase ← 0;
                bestSoFar ← cost(S′);
            else
                counter++;
            end
        end
        phase++;
        T ← T · β;
    end
end
```

Eigenschappen TTSA

- hard en soft constraints

Eigenschappen TTSA

- hard en soft constraints
- neighborhood van grootte $\mathcal{O}(n^3)$

Eigenschappen TTSA

- hard en soft constraints
- neighborhood van grootte $\mathcal{O}(n^3)$
- strategic oscillation

Eigenschappen TTSA

- hard en soft constraints
- neighborhood van grootte $\mathcal{O}(n^3)$
- strategic oscillation
- reheats

Hard en soft constraints

Voorstelling schedule

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

- T teams; R rounds
- + home; - away

- constraints
  - hard: double round-robin
  - soft: atmost & norepeat

Local search

- initieel random schedule

Local search

- initieel random schedule
  - eenvoudige recursieve backtrack search

Local search

- initieel random schedule
  - eenvoudige recursieve backtrack search
- kies S' in neighborhood van S

Local search

- initieel random schedule
  - eenvoudige recursieve backtrack search
- kies S' in neighborhood van S
  - SwapHomes(S, T_i, T_j)
  - SwapRounds(S, r_k, r_l)
  - SwapTeams(S, T_i, T_j)
  - PartialSwapRounds(S, T_i, r_k, r_l)
  - PartialSwapTeams(S, T_i, T_j, r_k)

Local search

- initieel random schedule
  - eenvoudige recursieve backtrack search
- kies S' in neighborhood van S
  - SwapHomes(S, T_i, T_j)
  - SwapRounds(S, r_k, r_l)
  - SwapTeams(S, T_i, T_j)
  - PartialSwapRounds(S, T_i, r_k, r_l)
  - PartialSwapTeams(S, T_i, T_j, r_k)
- aanvaard of verwerp S'

Voorbeeld PartialSwapTeams

PartialSwapTeams(S, T_2, T_4, r_9)

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

Voorbeeld PartialSwapTeams

PartialSwapTeams(S, T_2, T_4, r_9)

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

Objectieffunctie

Objectieffunctie

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

Objectieffunctie

Objectieffunctie

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

- $cost(S)$: afstandskost

Objectieffunctie

Objectieffunctie

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

- $cost(S)$: afstandskost
- $nbv(S)$: # violations (soft) constraints

Objectieffunctie

Objectieffunctie

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

- $cost(S)$: afstandskost
- $nbv(S)$: $\#$ violations (soft) constraints
- w: gewichtsfactor

Objectieffunctie

Objectieffunctie

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

- $cost(S)$: afstandskost
- $nbv(S)$: # violations (soft) constraints
- w: gewichtsfactor
- $f(v) = 1 + (\sqrt{v} \ln v)/\lambda$

Objectieffunctie

Objectieffunctie

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

- $cost(S)$: afstandskost
- $nbv(S)$: # violations (soft) constraints
- w: gewichtsfactor
- $f(v) = 1 + (\sqrt{v} \ln v)/\lambda$
  - sublineair, eerste violation kostelijker ($f(1) = 1$)

Objectieffunctie

Objectieffunctie

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

- $cost(S)$: afstandskost
- $nbv(S)$: # violations (soft) constraints
- w: gewichtsfactor
- $f(v) = 1 + (\sqrt{v} \ln v)/\lambda$
  - sublineair, eerste violation kostelijker ($f(1) = 1$)
  - ($\lambda = 2$ voor kleine n; $\lambda = 1$ voor grote n)

Uitbreidingen

- strategic oscillation

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

Uitbreidingen

- strategic oscillation

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

  - gewichtsfactor w variëren
  - feasible $w \leftarrow w/\theta$; infeasible $w \leftarrow w \cdot \delta$

Uitbreidingen

- strategic oscillation

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

  - gewichtsfactor w variëren
  - feasible $w \leftarrow w/\theta$; infeasible $w \leftarrow w \cdot \delta$
- reheating

Uitbreidingen

- strategic oscillation

$$C(S) = \begin{cases} cost(S) & \text{als } S \text{ feasible is,} \\ \sqrt{cost(S)^2 + [w \cdot f(nbv(S))]^2} & \text{anders.} \end{cases}$$

  - gewichtsfactor w variëren
  - feasible $w \leftarrow w/\theta$; infeasible $w \leftarrow w \cdot \delta$
- reheating
  - lokale minima op lage temperaturen
  - eenvoudig reheating schema

Inhoudsopgave

Implementatie

MATLAB

- + snelle implementatie
- + matrix- en vectoroperaties
- + snel testen
- – performantie (?)

Inhoudsopgave

Experimenten

- random schedule generatie
- parameters

Random schedule

eenvoudig recursieve backtracking?

```
RandomSchedule() {
  Q ← {⟨t, w⟩ | t ∈ Teams & w ∈ Weeks};
  GenerateSchedule(Q, S);
  return S;
}
bool GenerateSchedule(Q, S) {
  if Q = ∅ then return true; end if
  select ⟨t, w⟩ ∈ Q such that ∀⟨t′, w′⟩ ∈ Q : ⟨t′, w′⟩ ≥ ⟨t, w⟩;
  Choices ← {1, −1, ..., t − 1, −(t − 1), t + 1, −(t + 1), ..., n, −n};
  forall o ∈ Choices in random order do
    if ⟨o, w⟩ ∉ Q then
      S[t, w] ← o;
      if o > 0 then
        S[o, w] ← −t;
      else
        S[−o, w] ← t;
      end if
      if GenerateSchedule(Q \ {⟨t, w⟩, ⟨|o|, w⟩}, S) then
        return true;
      end if
    end if
  end forall
  return false;
}
```

Random schedule

kleine fout in algoritme

```
RANDOMSCHEDULE() {
  Q ← {⟨t, w⟩ | t ∈ Teams & w ∈ Weeks};
  GENERATESCHEDULE(Q, S);
  return S;
}
bool GENERATESCHEDULE(Q, S) {
  if Q = ∅ then return true; end if
  select ⟨t, w⟩ ∈ Q such that ∀⟨t′, w′⟩ ∈ Q : ⟨t′, w′⟩ ≥ ⟨t, w⟩;
  Choices ← {1, −1, . . . , t − 1, −(t − 1), t + 1, −(t + 1), . . . , n, −n};
  forall o ∈ Choices in random order do
    if ⟨o, w⟩ ∉ Q then
      S[t, w] ← o;
      if o > 0 then
        S[o, w] ← −t;
      else
        S[−o, w] ← t;
      end if
      if GENERATESCHEDULE(Q \ {⟨t, w⟩, ⟨|o|, w⟩}, S) then
        return true;
      end if
    end if
  end forall
  return false;
}
```

Random schedule

```
RANDOMSCHEDULE() {
  Q ← {⟨t, w⟩ | t ∈ Teams & w ∈ Weeks};
  GENERATESCHEDULE(Q, S);
  return S;
}
bool GENERATESCHEDULE(Q, S) {
  if Q = ∅ then return true; end if
  select ⟨t, w⟩ ∈ Q such that ∀⟨t', w'⟩ ∈ Q : ⟨t', w'⟩ ≥ ⟨t, w⟩;
  Choices ← {1, −1, ..., t − 1, −(t − 1), t + 1, −(t + 1), ..., n, −n};
  forall o ∈ Choices in random order do
    if ⟨o, w⟩ ∉ Q then
      S[t, w] ← o;
      if o > 0 then
        S[o, w] ← −t;
      else
        S[−o, w] ← t;
      end if
      if GENERATESCHEDULE(Q \ {⟨t, w⟩, ⟨|o|, w⟩}, S) then
        return true;
      end if
    end if
  end forall
  return false;
}
```

selected opponent o
not already assigned
in week w

Triviaal tegenvoorbeeld ($n = 2$)

- $Q = \{(1,1); (1,2); (2,1); (2,2)\}$
- $(t, w) = (1, 1)$
- $Choices = \{2, -2\}$
- $o = 2$
- $(o, w) = (2, 1) \in Q$

Random schedule

```
RandomSchedule() {
  Q ← {⟨t, w⟩ | t ∈ Teams & w ∈ Weeks};
  GenerateSchedule(Q, S);
  return S;
}
bool GenerateSchedule(Q, S) {
  if Q = ∅ then return true; end if
  select ⟨t, w⟩ ∈ Q such that ∀⟨t′, w′⟩ ∈ Q : ⟨t′, w′⟩ ≥ ⟨t, w⟩;
  Choices ← {1, −1, ..., t − 1, −(t − 1), t + 1, −(t + 1), ..., n, −n};
  forall o ∈ Choices in random order do
    if ⟨o, w⟩ ∉ Q then
      S[t, w] ← o;
      if o > 0 then
        S[o, w] ← −t;
      else
        S[−o, w] ← t;
      end if
      if GenerateSchedule(Q \ {⟨t, w⟩, ⟨|o|, w⟩}, S) then
        return true;
      end if
    end if
  end forall
  return false;
}
```

selected opponent o not already assigned in week w

Triviaal tegenvoorbeeld ($n = 2$)

- $Q = \{(1,1); (1,2); (2,1); (2,2)\}$
- $(t, w) = (1, 1)$
- $Choices = \{2, -2\}$
- $o = 2$
- $(o, w) = (2, 1) \in Q$

maar o is beschikbaar

Random schedule

```
RandomSchedule() {
  Q ← {⟨t, w⟩ | t ∈ Teams & w ∈ Weeks};
  GenerateSchedule(Q, S);
  return S;
}
bool GenerateSchedule(Q, S) {
  if Q = ∅ then return true; end if
  select ⟨t, w⟩ ∈ Q such that ∀⟨t′, w′⟩ ∈ Q : ⟨t′, w′⟩ ≥ ⟨t, w⟩;
  Choices ← {1, −1, . . . , t − 1, −(t − 1), t + 1, −(t + 1), . . . , n, −n};
  forall o ∈ Choices in random order do
    if ⟨o, w⟩ ∉ Q then
      S[t, w] ← o;
      if o > 0 then
        S[o, w] ← −t;
      else
        S[−o, w] ← t;
      end if
      if GenerateSchedule(Q \ {⟨t, w⟩, ⟨|o|, w⟩}, S) then
        return true;
      end if
    end if
  end forall
  return false;
}
```

selected opponent o not already assigned in week w

+ t versus o not already assigned

Triviaal tegenvoorbeeld ($n = 2$)

- $Q = \{(1,1); (1,2); (2,1); (2,2)\}$
- $(t, w) = (1, 1)$
- $Choices = \{2, -2\}$
- $o = 2$
- $(o, w) = (2, 1) \in Q$

anders kan team t meerdere keren zelfde wedstrijd tegen o spelen

Random schedule

scaleerbaarheid:

- goed voor NL4–8
- ok voor NL10–12
- traag voor NL14–16

bottleneck:

- shuffle-algoritme

  (`randperm` $\mathcal{O}(n \log n)$ vs $\mathcal{O}(n)$)
- #backtracks

Random schedule

scaleerbaarheid:

- goed voor NL4–8
- ok voor NL10–12
- traag voor NL14–16

bottleneck:

- shuffle-algoritme

  (`randperm` $\mathcal{O}(n \log n)$ vs $\mathcal{O}(n)$)
- #backtracks

n	min (s)	avg (s)	max (s)	std (s)
4	0.004	0.006	0.009	0.002
6	0.011	0.017	0.044	0.010
8	0.020	1.889	18.558	5.857
10	0.031	14.271	112.291	35.549
12	0.055	8.795	51.114	16.583
14	0.089	96.782	612.953	195.414
16	1.088	286.021	923.226	360.467

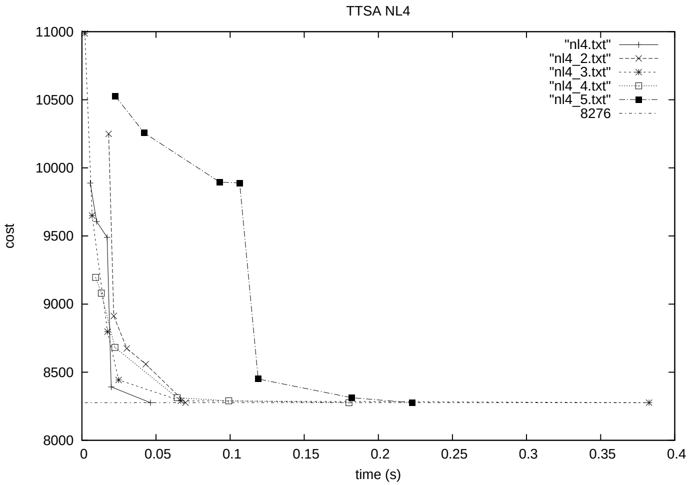
Tabel: Tijd nodig om een random schedule te maken via een recursief backtrack algoritme ($N = 10$).

Parameters

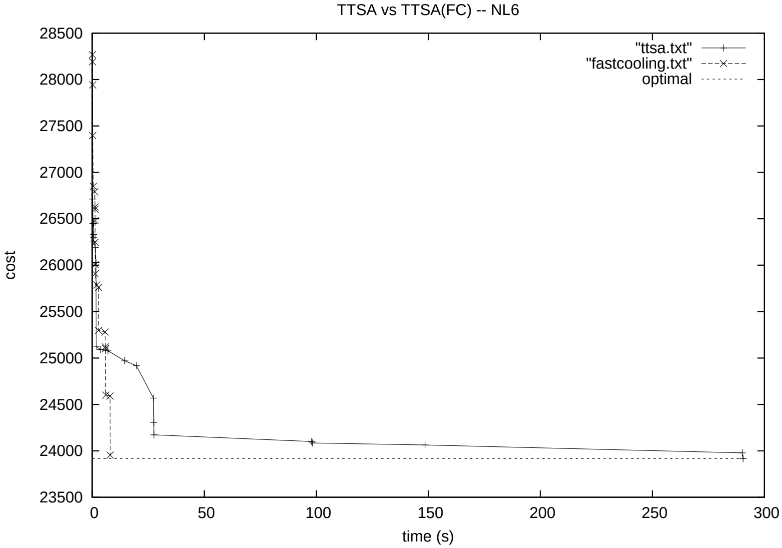
parameters empirisch bepalen

- TTSA
  - traag $\Rightarrow$ weinig experimenten
- TTSA (Fast Cooling)
  - beperkte tijd $\Rightarrow$ meer experimenten
  - goede resultaten

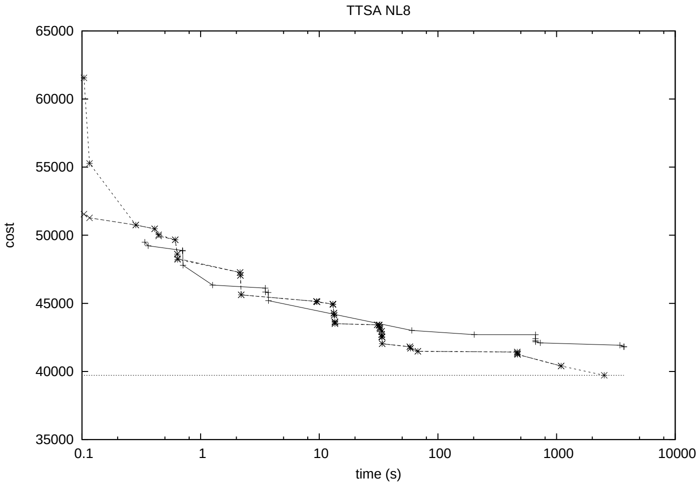
	T_0	β	$maxC$	$maxP$	$maxR$	δ	θ	w_0
TTSA	400–700	0.9999	4000–10000	7100	10–50	1.04	1.04	4000–60000
TTSA(FC)	400–600	0.99	100–500	100–500	1–5	1.04	1.04	4000–10000



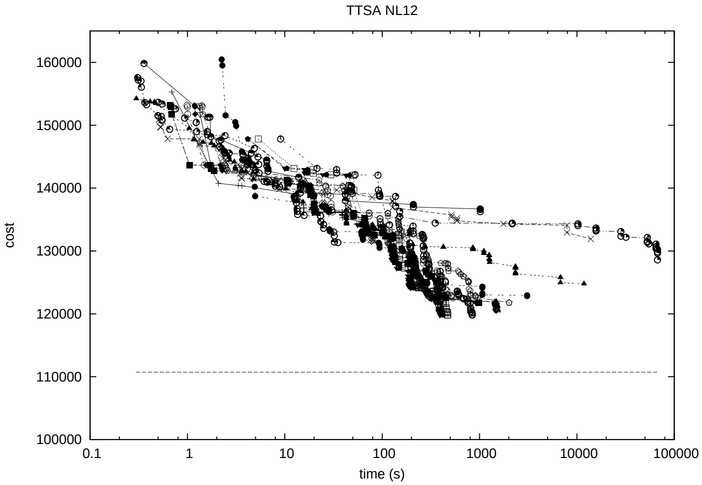
Figuur: TTSA NL4.



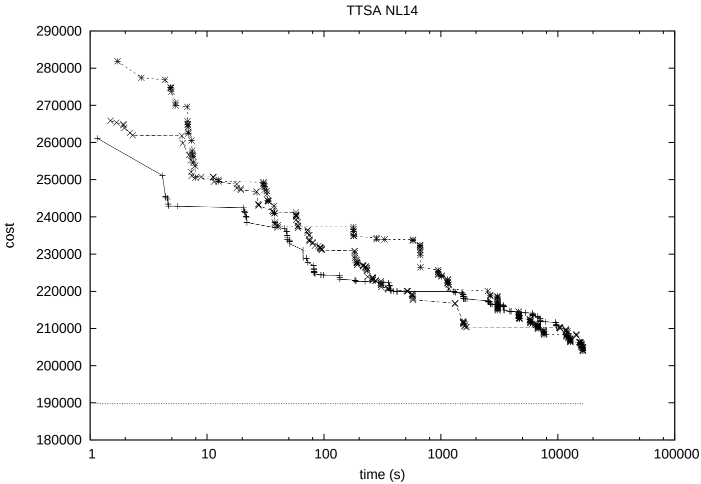
Figuur: TTSA vs TTSA(FC) NL6.



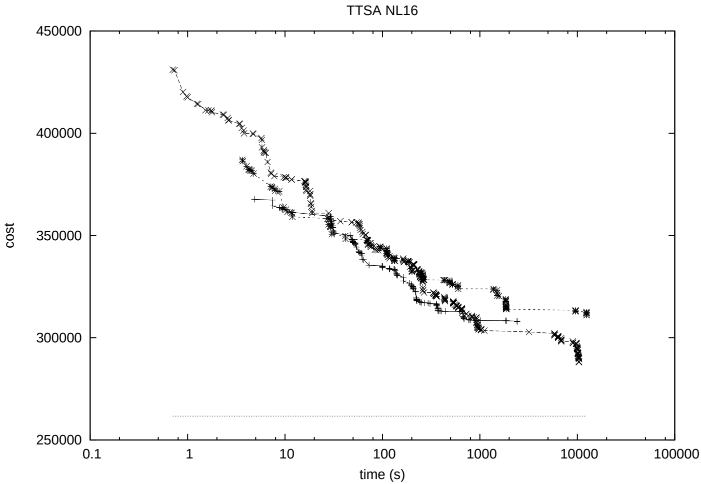
Figuur: TTSA NL8 (logaritmische tijdsas).



Figuur: TTSA(FC) NL12 (logaritmische tijdsas).



Figuur: TTSA(FC) NL14 (logaritmische tijdsas).



Figuur: TTSA(FC) NL16 (logaritmische tijdsas).

Inhoudsopgave

Mogelijke verbeteringen

- beter algoritme initieel schedule

Mogelijke verbeteringen

- beter algoritme initieel schedule
  - randomized hill-climbing: Dinitz en Stinson. *A Hill-climbing Algorithm for the Construction of One-Factorizations and Room Squares*. (1987)

Mogelijke verbeteringen

- beter algoritme initieel schedule
  - randomized hill-climbing: Dinitz en Stinson. *A Hill-climbing Algorithm for the Construction of One-Factorizations and Room Squares*. (1987)
  - constraint programming?

Mogelijke verbeteringen

- beter algoritme initieel schedule
  - randomized hill-climbing: Dinitz en Stinson. *A Hill-climbing Algorithm for the Construction of One-Factorizations and Room Squares*. (1987)
  - constraint programming?
- parameters TTSA

Mogelijke verbeteringen

- beter algoritme initieel schedule
  - randomized hill-climbing: Dinitz en Stinson. *A Hill-climbing Algorithm for the Construction of One-Factorizations and Room Squares*. (1987)
  - constraint programming?
- parameters TTSA
  - random restart

Mogelijke verbeteringen

- beter algoritme initieel schedule
  - randomized hill-climbing: Dinitz en Stinson. *A Hill-climbing Algorithm for the Construction of One-Factorizations and Room Squares*. (1987)
  - constraint programming?
- parameters TTSA
  - random restart
  - neighborhood

Mogelijke verbeteringen

- beter algoritme initieel schedule
  - randomized hill-climbing: Dinitz en Stinson. *A Hill-climbing Algorithm for the Construction of One-Factorizations and Room Squares*. (1987)
  - constraint programming?
- parameters TTSA
  - random restart
  - neighborhood
  - ...

Mogelijke verbeteringen (2)

- uitbreidingen naar niet NL-instanties.
  Van Hentenryck en Vergados.
  A Traveling Tournament Scheduling: A Systematic Evaluation of Simulated Annealing. (2006)

Mogelijke verbeteringen (2)

- uitbreidingen naar niet NL-instanties.
  Van Hentenryck en Vergados.
  A Traveling Tournament Scheduling: A Systematic Evaluation of Simulated Annealing. (2006)
- hybride algoritme, e.g. population-based SA.
  Van Hentenryck en Vergados.
  Population-based simulated annealing for traveling tournaments. (2007)

Mogelijke verbeteringen (2)

- uitbreidingen naar niet NL-instanties.
  Van Hentenryck en Vergados.
  A Traveling Tournament Scheduling: A Systematic Evaluation of Simulated Annealing. (2006)
- hybride algoritme, e.g. population-based SA.
  Van Hentenryck en Vergados.
  Population-based simulated annealing for traveling tournaments. (2007)
- ...

Inhoudsopgave

Conclusie

- matige resultaten TTSA

Conclusie

- matige resultaten TTSA
- goede resultaten TTSA(FC)

Conclusie

- matige resultaten TTSA
- goede resultaten TTSA(FC)
- empirisch bepalen parameters

Conclusie

n	cost	best (2002)	TTSA (2003)	best (2010)
4	*8276*	8276	8276	8276
6	*23916*	23916	23916	23916
8	*39721*	39721	39721	39721
10	63667	61608	59583	59436
12	*118499*	118955	112800	110729
14	*203979*	205894	190368	188728
16	288089	281660	267194	261687