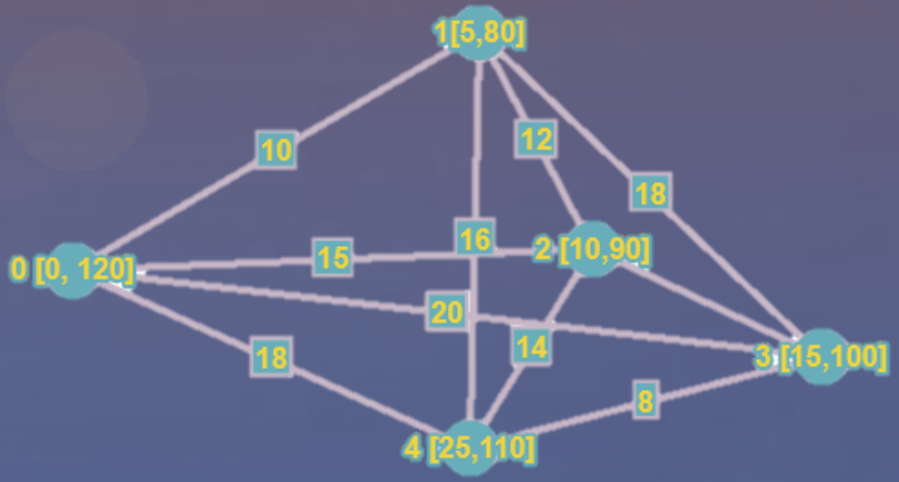
Traveling Salesman Problem with Time Windows

Mihaela Filipović 82/2020
Lazar Bećarević 93/2020

Računarska inteligencija 2024/2025

Definicija problema

- Osnovni TSP problem
- TSP sa vremenskim prozorima
- NP težak



Cilj formalno

- Skup gradova {0,1,2,...,n −1}, gde je grad 0 depo (polazna tačka).
- Matrica puta T[i][j].
- Vremenski prozor [ei,li] (najraniji i najkasniji dolazak) i trajanje usluge.
- Cilj je minimalno ukupno vreme putovanja uz poštovanje vremenskih prozora

Odnosno da minimizujemo:

$$\min_X \sum_{i=1}^{n-1} T(x_i, x_{i+1}) + T(x_n, x_1)$$

pod uslovom: $e_i \leq t_i \leq l_i \quad \text{za svaki grad } i$

Tehnike rešavanja

- Brute force
  - DFS
  - Odsecanje (pruning)
- Simulirano kaljenje (SA)
  - Heuristika za izgradnju početnog rešenja
  - Neighborhood moves
  - Temperatura

Implementacija

- Format ulaznih podataka:

```python
class TSPTWInstance:
    """Time-windowed TSP instance with n nodes (node 0 = depot)."""
    T: List[List[int]]  # n x n travel time matrix
    e: List[int]        # earliest service times
    l: List[int]        # latest service times (due dates)
    s: List[int]        # service durations
    n: int              # number of nodes
```

- Deljene funkcije:
  - **schedule_route** – proverava izvodljivost (u odnosu na dolaske u gradovima) i izračunava trošak
  - **compute_forward_backward** – izračunava granice izvodljivosti (najraniji /najkasniji odlazak)
  - **forward_precheck_partial** – brzo izračunava prihvatljivost rešenja

Brute force

- DFS

```
def dfs(curr: int, t_start_curr: int, travel_cost_so_far: int,
        visited_mask: int, route: List[int], times: List[int])
```

- Odsecanje (pruning)
  - Ako je nemoguće stići na vreme
  - Optimizacija "donje granice"

```
# Lower bound pruning
# Build list of unvisited
unvisited = [i for i in range(1, n) if not (visited_mask & (1 << i))]
# quick bound
bound = travel_cost_so_far + lb_travel(curr, unvisited)
if bound >= best_cost:
    stats["pruned_lb"] += 1
    return
```

Vremenska Složenost:
Najgori slučaj: O(n!) -
Sa odsecanjem: O(b^n) gde je b < n (faktor grananja)
Praktično zapažanje: Izvodljivo za n ≤ 12-15

Simulirano kaljenje

- Greedy feasible insertion
  - gradi početnu rutu
- Move (relocate, swap, two_opt)
- Temperature

[0, 1, 3, 4, 2, 0] → [0, 1, 2, 3, 4, 0]

temp

[0, 1, 2, 3, 4, 0]

- Auto kalibracija temperature:

```
# Pojednostavljena logika:
sample_deltas = []  # pozitivna povećanja troška
for 200 nasumičnih poteza:
   if potez povećava trošak:
      sample_deltas.append(povećanje_troška)

prosečno_povećanje = mean(sample_deltas)
T0 = -prosečno_povećanje / ln(0.75)
```

- Move (relocate, swap, two_opt):

```python
def move_relocate(route: List[int], i: int, j: int) -> Tuple[List[int], int, int]
    if i == 0 or i == len(route)-1:
        return route, i, j
    r = route[:]
    node = r.pop(i)
    j = min(j, len(r))
    r.insert(j, node)
    return r, min(i,j), max(i,j)
```

```python
def move_swap(route: List[int], i: int, j: int) -> Tuple[List[int], int, int]:
    if i == 0 or j == 0 or i == len(route)-1 or j == len(route)-1:
        return route, i, j
    if i == j: return route, i, j
    r = route[:]
    r[i], r[j] = r[j], r[i]
    return r, min(i,j), max(i,j)

def move_two_opt(route: List[int], i: int, j: int) -> Tuple[List[int], int, int]:
    if i < 1: i = 1
    if j > len(route)-2: j = len(route)-2
    if i >= j: return route, i, j
    r = route[:i] + list(reversed(route[i:j+1])) + route[j+1:]
    return r, i, j
```

- SA solver:

```
def sa_solve(inst: TSPTWInstance,
             route0: Optional[List[int]] = None,
             T0: Optional[float] = None,
             alpha: float = 0.98,
             iters_per_temp: int = 200,
             min_T: float = 1e-3,
             max_iter_no_improve: int = 5000,
             time_limit_sec: Optional[float] = None,
             rng_seed: int = 1234,
             tw_soft: bool = False,
             beta: float = 0.0) -> Dict:
```

- Kriterijum prihvatanja:

```
delta = novi_trosak - trenutni_trosak

if delta <= 0:
    # Poboljšanje → uvek prihvati
    prihvati = True
else:
    # Gore rešenje → prihvati sa verovatnoćom
    verovatnoca = exp(-delta / T)
    if random() < verovatnoca:
        prihvati = True
```

- Povratne vrednosti:

```
return {
    "route": best_route,
    "cost": best_cost,
    "times": best_times,
    "tardiness_sum": best_td,
    "accepted_cost": curr_cost,
    "iterations": total_iters,
    "final_T": Tcur,
```

Testirani slučajevi

- Grafovi različitih veličina, težina i vremenskih prozora
- Granični slučajevi:
  - Trivijalno rešenje
  - Široki vremenski prozori (TSP)
  - Jako uski vremenski prozori
  - Obavezno čekanje
  - Bez prihvatljivog rešenja
  - Asimetrična vremena puta (T[i][j] != T[j][i])...

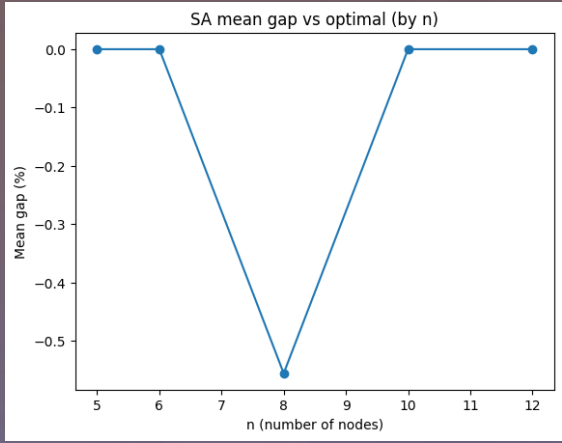
Benchmarking i vizualizacija

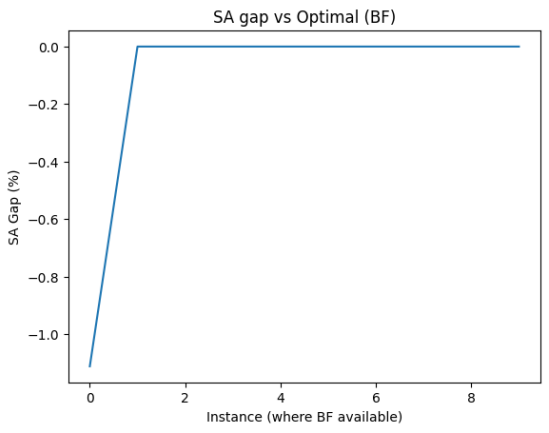
- Benchmark daje dve ključne metrike:
  - GAP (mera razlike između SA rešenja i optimalnog rešenja)
  - Speedup (ubrzanje SA u odnosu na brute force metodu)

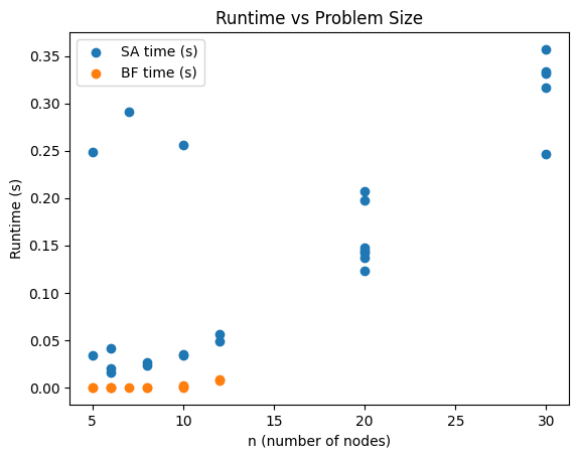
Benchmarking i vizualizacija

suite	n	bf_cost	bf_time_sec	bf_ok	bf_pruned_tv	bf_pruned_ll	sa_cost	sa_time_sec	gap_pct
n08_easy	8	90	0.001	True	5	246	89	0.104	-1.111
n08_easy	8	78	0.001	True	0	242	78	0.085	0
n10_medium	10	104	0.004	True	1	1119	104	0.123	0
n10_medium	10	116	0.007	True	195	2003	116	0.136	0
n12_easy	12	126	0.028	True	498	8964	126	0.174	0
n12_easy	12	138	0.029	True	431	9370	138	0.206	0
n20_easy	20	871	60	True	20263436	5366620	822	0.661	-5.626
n20_easy	20	671	60.001	True	21934899	3495029	598	0.711	-10.879
n20_medium	20	730	60	True	32883302	513733	730	0.41	0
n20_medium	20	907	60	True	34596903	122307	907	0.461	0
n20_medium	20	1092	60	True	35584319	13399	1092	0.491	0
n20_medium	20	837	60	True	30529582	713838	837	0.505	0
n20_medium	20	1097	60.002	True	33637060	43624	1092	0.483	-0.456
n30_medium	30			False			1333	0.879	
n30_medium	30			False			1281	1.128	

- Pomoću generisanih metrika dobijamo:
  - Gap u zavisnosti od veličine problema (gap_by_n).
  - Distribucija gap-ova po nivou težine (gap_box).
  - Poređenje vremena izvršenja (runtime)
  - Performansni profil (profile)


SA mean gap vs optimal (by n)
Mean gap (%)
n (number of nodes)


SA gap vs Optimal (BF)
SA Gap (%)
Instance (where BF available)


Runtime vs Problem Size
SA time (s)
BF time (s)
Runtime (s)
n (number of nodes)


Performance profile (quality)
Fraction of instances ≤ τ
Quality ratio τ = cost(SA)/cost(OPT)

Zaključak

- Rezultati pokazuju da je simulirano kaljenje znatno brže i skalabilnije za malo veće instance, dok brute force pruža egzaktna rešenja za manje dimenzije problema.
- Ipak za SA može biti potrebno puno vremena za veoma velike instance problema, a uspeh algoritma zavisi od pravilnog podešavanja parametara.

Potencijalna poboljšanja

- Adaptivno smanjenje temperature (umesto linearnog)
- Paralelizacija
- Hibridni pristup ili promena heuristike

HVALA NA PAŽNJI!