GRASP

in aproksimativna rešitev za TSP

Seminarska naloga pri predmetu
Izbrana poglavja iz optimizacije

Leon Lampret

Fakulteta za Matematiko in Fiziko,
Univerza v Ljubljani,
Oddelek za Matematiko

19. maj 2011

GRASP

Metahevristika je algoritemski način reševanja kombinatoričnega optimizacijskega problema, pri katerem na začetku izberemo množico kandidatov za rešitev, in jo iterativno izboljšujemo (glede na neko vnaprej izbrano funkcijo zaželjenosti), ter po dovolj korakih vrnemo najboljši element iz te množice.

Metahevristike torej vrnejo približne rešitve, a veliko hitreje kot eksaktni postopki.

GRASP ("greedy randomized adaptive search procedure") je metahevristika, ki sestoji iz dveh faz: *"greedy randomized construction"* in *"local search"*.

V prvi fazi na pameten način (odvisno od problema) izberemo izmed vseh možnih rešitev CL ("candidate list") množico začetnih približkov RCL ("restricted candidates list"). To storimo deloma deterministično in deloma stohastično, da zagotovimo, da so začetni približki obetavni, a dovolj razpršeni po celotni množici CL, da bo druga faza pregledala čimvečji del CL.

V drugi fazi za vsako izmed teh rešitev $s \in \text{RCL}$ pregledamo elemente $s' \in \text{CL}$ v njeni okolici (kaj je okolica je od problema in načina reševanja odvisno). Če najdemo boljšo rešitev s', jo dodamo v RCL ter s odstranimo. To ponavljamo dokler zaustavitveni pogoj (npr. št. iteracij, zahtevana natančnost) ni izpolnjen.

Input: $\mathcal{O}$; objekt, ki ga preučujemo, npr. graf, matroid, itd.
Algorithm:
ugotovi, kaj je množica vseh možnih rešitev $\mathtt{CL} \subseteq 2^{\mathcal{O}}$;
(ta množica ni eksplicitno podana v programu, ker je prevelika)
določi, kaj je funkcija zaželjenosti $f : \mathtt{CL} \to \mathbb{R}$
določi množico začetnih približkov $\mathtt{RCL} \subseteq \mathtt{CL}$
določi kaj pomeni okolica elementa $\mathtt{s} \in \mathtt{RCL}$ v množici CL
ponavljaj dokler ni zadoščeno zaustavitvenemu pogoju:
    naključno izberi $\mathtt{s} \in \mathtt{RCL}$
    preglej vse elemente $\mathtt{s'} \in \mathtt{CL}$ v okolici s
    če najdeš s', ki je bolj zaželjen, tj. $f(\mathtt{s'}) > f(\mathtt{s})$,
    ga zamenjaj z s, tj. $\mathtt{RCL} := (\mathtt{RCL} - \{\mathtt{s}\}) \cup \{\mathtt{s'}\}$
Return: najboljši element (glede na f) v množici RCL.

TSP

Problem trgovskega potnika ("travelling salesman problem"/TSP) se glasi:

formulacija v vsakdanjem jeziku: danih je n mest in razdalja med poljubnim parom mest (od mesta do mesta lahko potujemo po zgolj eni poti). Najdi najkrajšo pot, ki se začne in konča v istem mestu ter obišče vsako mesto natanko enkrat!

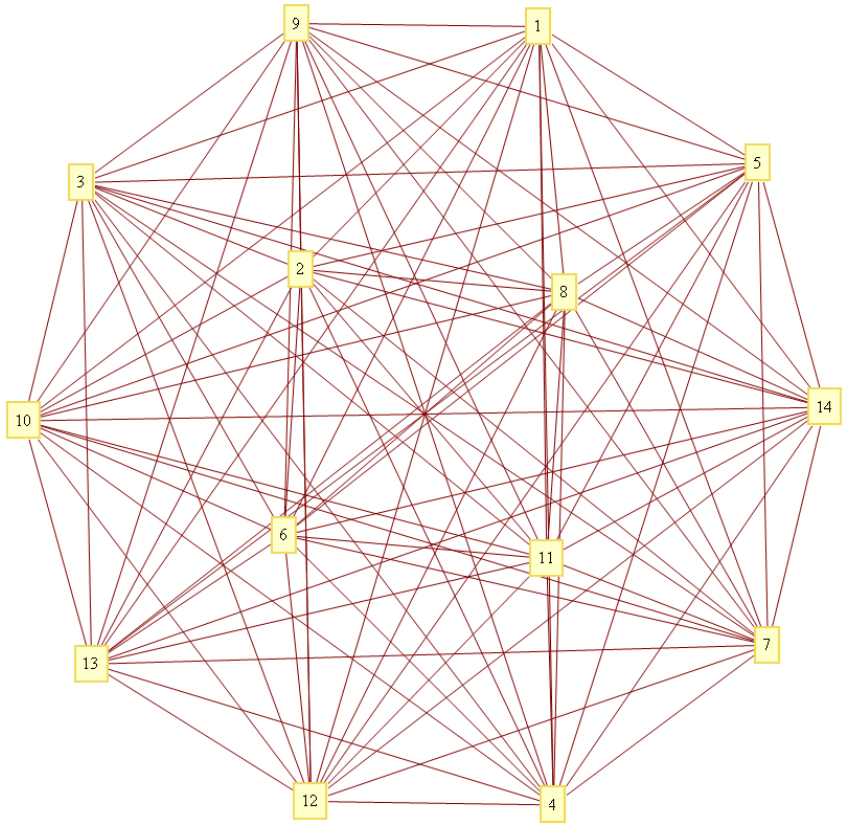
formulacija v matematičnem jeziku: v (neusmerjenem enostavnem) polnem grafu K_n z uteženimi povezavami (pozitivne vrednosti) najdi najkrajši cikel, ki vsebuje vsa vozlišča! Ciklom, ki vsebujejo vsa vozlišča grafa, pravimo *Hamiltonovi cikli*.

RAČUNSKA ZAHTEVNOST:

Če so vozlišča (mesta) označena z $1, \ldots, n$, lahko vsak Hamiltonov cikel v K_n predstavimo z zaporedjem $(1, *, \ldots, *)$ števil $\{2, \ldots, n\}$. Pri tem vsaki zaporedji oblike $(1, v_2, v_3 \ldots, v_{n-1}, v_n)$ in $(1, v_n, v_{n-1}, \ldots, v_2, v_1)$ predstavljata isti cikel. Zato je število vseh možnih Hamiltonovih ciklov v K_n enako

$$h_n = \frac{(n-1)!}{2}.$$

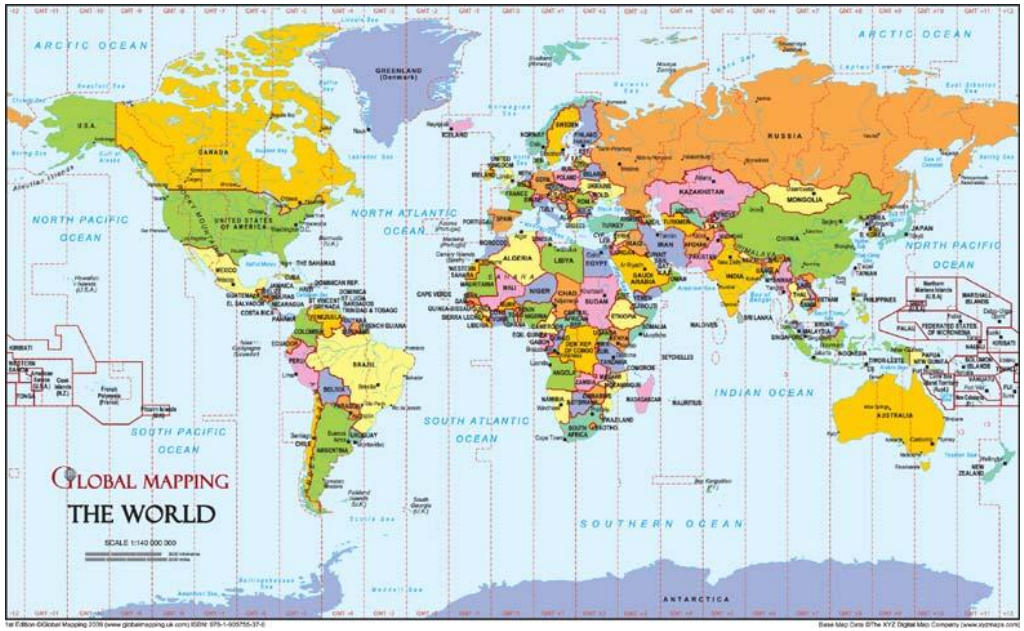
Ko $n \in \mathbb{N}$ raste, postaja h_n neobvladljivo velik, kar pomeni, da vse eksaktne metode iskanja optimalnega cikla niso primerne.



Da dobimo občutek o računski kompleksnosti problema TSP, omenimo dva zgleda. Graf na zgornji sliki vsebuje 14 vozlišč, 91 povezav in preko 3 miljarde Hamiltonovih ciklov.

Kot drugi zgled omenimo, da če bi želeli obiskati prestolnice vseh 203 držav na svetu (vsako natanko enkrat), bi to lahko storili na več kot 10^{379} načinov (toliko je torej vseh možnih Hamiltonovih ciklov na polnem grafu vseh svetovnih prestolnic).

Za primerjavo, število vseh atomov našega planeta je približno $9 \cdot 10^{49}$, število vseh zvezd v vidnem vesolju (groba ocena) pa 10^{22}.



TSP via GRASP

Vhodni podatki: g, *iter*, k.

g graf z vozlišči $\{1, \ldots, n\}$. Podan je v obliki incidenčne matrike, tj. $g_{i,j}$ = razdalja med mestoma i in j. *iter* je število dovoljenih iteracij algoritma, k pa število začetnih približkov, torej $|\mathrm{RCL}|$. Mn. vseh rešitev CL je v našem primeru mn. vseh Hamiltonovih poti v g. Za te se odločimo, da jih predstavimo kot zaporedja števil $(l, 1, v_2, \ldots, v_n)$, $v_i \in \{2, \ldots, n\}$, kjer je l dolžina cikla $(1, v_2, \ldots, v_n)$ v grafu g. Torej sproti hranimo vrednost (dolžino) cikla, da ni treba vsakič znova računati, in tudi, da lahko cikli postanejo med seboj primerljiva podatkovna struktura. Seveda med algoritmom množica CL nikoli ni zapisana v računalniku, saj je prevelika.

Algoritem:

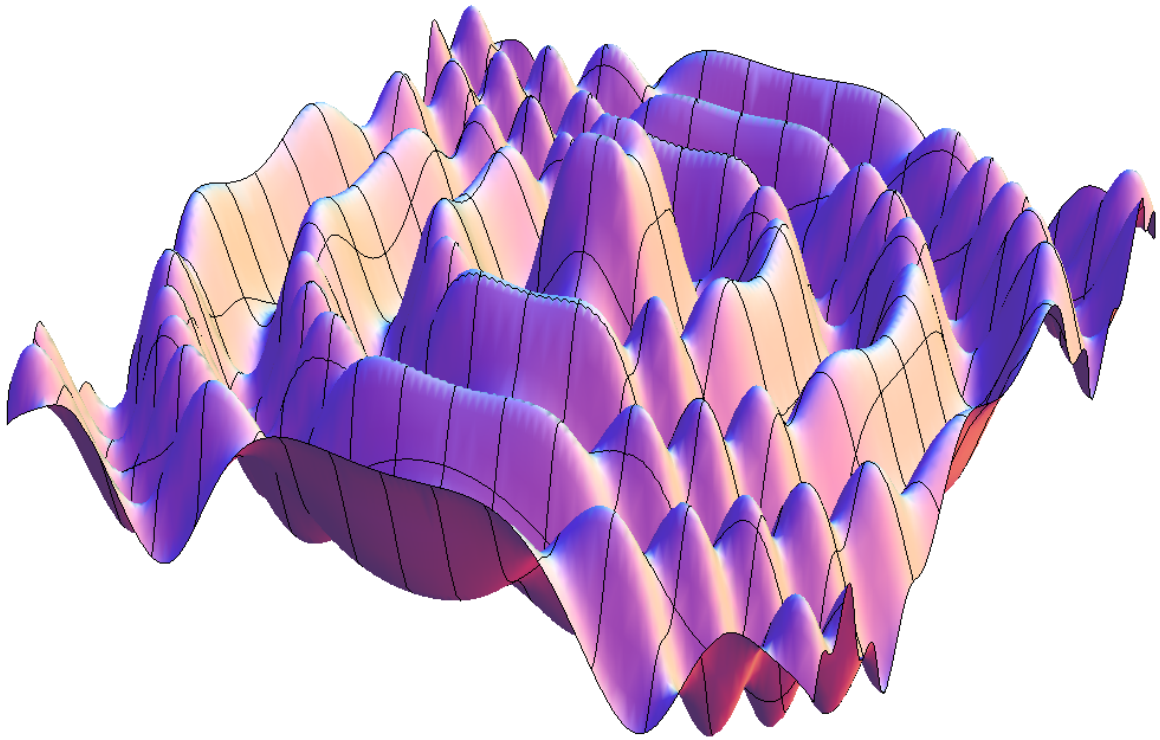
Greedy randomized construction: Vsak začetni približek $t = (l, 1, v_2, \ldots, v_n) \in \text{RCL}$ (t...”tour”) konstruiramo tako, da določimo $v_1 := 1$; nato iterativno ($i = 2, \ldots, n$) za v_i izberemo naključno izmed $\lfloor \frac{n}{5} \rfloor$ najbližjih vozlišč do v_{i-1}, ki še niso v t. Ko je $(1, v_2, \ldots, v_n)$ izbran, določimo še l, kot dolžino tega cikla. Take cikle t konstruiramo toliko časa, da RCL napolnimo (dokler $|\text{RCL}| < k$). Ker moramo elemente v RCL znati primerjati po dolžini, mora biti RCL urejena podatkovna struktura (v našem primeru se odločimo za **sorted set** v Javi).

Local search: Elementi v RCL so urejeni po dolžini (primerjamo jih po 0-ti koordinati l). Naključno izberemo $t \in \text{RCL}$, toda z linearno padajočo verjetnostjo: najverjetneje izberemo cikel na vrhu RCL (najkrajši), malo manj verjetneje drugega (drugi najkrajši), še manj verjetneje tretjega, itd. Okolico cikla t definiramo kot množico vseh ciklov $t' \in \text{CL}$, ki jih dobimo iz t, tako da mu zamenjamo 2 vozlišči: $v_i \leftrightarrow v_j$. V algoritmu torej izberemo t' tako, da naključno zamenjamo dve vozlišči cikla t. Nato preverimo, če smo s tem dobili krajši cikel. V primeru da ja, t odstranimo iz RCL, ter t' dodamo v RCL. Ta postopek ponavljamo tolikokrat, kolikor je predpisano število ponovitev (*iter*).

Izhodni podatek: Na koncu vrnemo zgornji (najkrajši) element v RCL (tj. približek najkrajšega Hamiltonovega cikla v g).

tipičen problem algoritma

Predstavljajmo si vse možne Hamiltonove cikle v našem grafu (torej mn. CL) kot (diskretno) podmnožico $\mathbb{R}^2$, in naj $f : CL \to \mathbb{R}$ ciklu priredi minus njegovo dolžino.



Potem se naš optimizacijski problem glasi "najdi maksimum funkcije f". Množica RCL je podmnožica od CL, in algoritem bo v vsaki iteraciji neko točko iz RCL zamenjal s kako v njeni neposredni okolici. Torej bodo točke iz RCL skonvergirale k njihovim **lokalnim maksimumom**, in tam ostale ujete.

Pomembno je torej, da je množica RCL čimbolj razpršena, do problemov pa bo vseeno prihajalo vsakič, ko bo f imela veliko lokalnih maksimumov. Večja kot je RCL, večja bo verjetnost, da kak element pristane v bližini **globalnega maksimuma** in zato k njemu skonvergira.