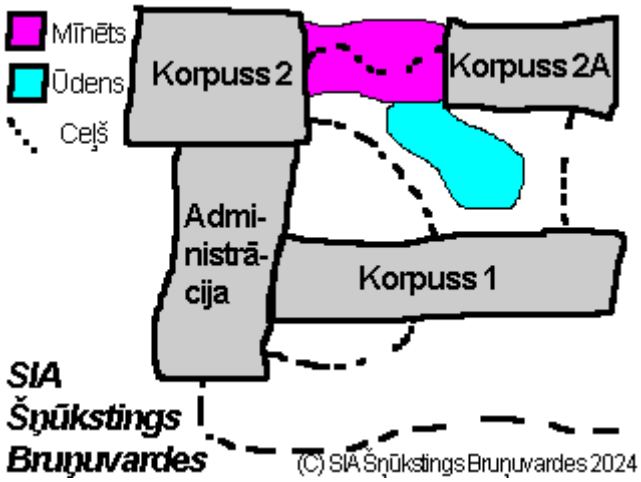
Lielais praktiskais darbs

Uzdevuma formulējums

SIA Šņūkstings Bruņuvardes ir bruņuvaržu ražošanas uzņēmums. Uzņēmumā strādā tikai šņūksti. Katru dienu tajā strādājošie šņūksti pabaro bruņuvardes, bet viņiem nav skaidrs kāds ir vislabākais grafiks bruņuvaržu barošanai, tāpēc viņiem ir nepieciešama IT sistēma, kas spēs optimizēt barošanas grafikus. Papildus šņūkstiem jāpaspēj pusdienās iesņūkstēties.



Uzņēmuma teritorijā ir iespējams nokļūt tikai caur administrācijas ēku. Tajā arī atrodas ēdnīca. Tālāk ir iespējams no administrācijas ēkas nokļūt uz korpusu 1. No korpusa 1 ir iespējams nokļūt uz korpusu 2 un korpusu 2a. Ir iespējams arī nokļūt no korpusa 2 uz korpusu 2a, bet ceļš starp tiem ir mīnēts.

Bruņuvardes tiek uzglabātas bruņuvaržu krātiņos. Katram krātiņam ir numurs. Krātiņi ar secīgiem numuriem arī fiziski atradīsies blakus. Vienā krātiņā ir viena bruņuvarde. Katrā korpusā ir zināms daudzums krātiņu. Administrācijas ēkā nav krātiņu.

Ir jāatrod grafiks, kas būs bruņuvaržu barošanas secība. Tas sastāvēs no ierakstiem, katrā ierakstā būs norādīts kāda darbība šņūkstam ir jāveic, kurā korpusā tā ir jāveic un kurā krātiņā tā ir jāveic. Grafikam jābūt saliktam tādam, ka ja šņūksts seko šim grafikam, viņš būs paveicis savu darbu iespējami īsākā laikā.

Domēna modelis

Korpusi tiek modelēti kā ieraksts, kas sastāv no korpusa identifikatora, nosaukuma un skaita, cik daudzi bruņuvaržu krātiņi tajā atrodas. Papildus tiek glabāti attālumi starp korpusiem un norādīts vai ceļš starp korpusiem ir mīnēts.

Aktivitātes tiek modelētas kā ieraksts, kas sastāv no darbības veida, korpuss, kurā jāveic aktivitāte un krātiņa, kurā jāveic tā darbība.

Šņūksti tiek modelēti kā ieraksts, kas sastāv no identifikatora, šņūksta vārda un saraksta ar laikiem, kuros šņūksts nebūs pieejams darbību veikšanai.

Šņūkstu grafiks sastāv no saraksta ar aktivitātēm.

Risinājums sastāv no šņūkstu saraksta, kas tajā dienā strādā, un saraksta ar viņu grafikiem.

Izmaksu funkcija

Izmaksas tiek aprēķinātas ņemot katra šņūksta grafiku, un tad ejot tam cauri un rēķinot kurā laikā katram šņūkstam beigsies darbs. Papildus tiek ievākti vēl dažādi mērījumi. Kopā tiek aprēķināts:

Pilnais nostrādātais laiks, kuru ir nostrādājis šņūksts.
Virslaiks, kuru nostrādāji šņūksts. Pilnā darba dienā šņūksts nostrādās 2000 laika vienības, ja šņūksts nostrādās vairāk par šo skaitu, tad tās tiks uzskatītas par pārstrādi un pieskaitīsies virslaikam.
Zemlaiks, kuru nostrādājis šņūksts. Ja šņūksts beidz darbu ātrāk par 2000 laika vienībām, tad starpība tiks uzskatīta par zemstrādi un pieskaitīsies pie zemlaika.
Aizliegtais laiks, kuru ir nostrādājis šņūksts. Šņūksti normālā dienā dienā var veikt 2000 laika vienības normālstrādes un 2000 vienības virsstrādes. Ja šņūksts vairāk laika strādās, tad to darbu uzskatīs par fiziski neiespējamu un pieskaitīs aizliegtajam laikam.
Novirze no pusdienlaika. Šņūksti parasti iet iešņūkstēties 1000 laika vienībās no darba dienas sākuma. Ja šņūksts ies iešņūkstēties ātrāk vai vēlāk, tad šī novirze tiks pieskaitīta novirzei no pusdienlaika.
Mīnu lauku šķērsojumi, kurus ir veicis šņūksts pārvietojoties no viena uz otru korpusu.
Nesakārtotība ir skaits, cik daudzi krātiņi tika apmeklēti ārpus to secības, t.i. kuri fiziski neatrodas blakus viens otram.

Mīkstās izmaksas tiek aprēķinātas ar šādu formulu:

$$2 \cdot virslaiks + zemlaiks + pusdienlaika\,novirze + 10 \cdot mīnu\,lauku\,šķērsojumi + nesakārtotība$$

Cietās izmaksas tiek aprēķinātas ar šādu formulu: $aizliegtais\,laiks$

Kopējās izmaksas tiek aprēķinātas ar šādu formulu:

$$mīkstās\,izmaksas + 5 \cdot cietās\,izmaksas$$

Apkārtnes funkcija

Apkārtnes funkcija vispirms nejauši izvēlēsies divus šņūkstu grafikus. Šie grafiki var būt atšķirīgi, bet tas arī var būt tas pats grafiks. Pēc tam funkcija no pirmā grafika nejauši izvēlēsies vienu aktivitāti un izņems to no grafika. Tad tā ievietos to otrajā grafikā, nejauši izvēlētā vietā.

Funkcija pārvietos aktivitātes starp dažādiem grafikiem, tikai ja šī aktivitāte nav iešņūkstēšanās, jo katrs sņūksts iešņūkstēsies tikai vienu reizi dienā.

Risinājuma arhitektūra

Risinājums tika veidots Erlang valodā, jo es tajā neko iepriekš nebiju mēģinājis programmēt un tāpēc man šķita ka šāds vieglais projekts varētu būt labs veids kā to iemācīties.

Programma ir sadalīta 3 moduļos: domēns, optimizators un serveris.

Domēna modulis satur visas domēna definīcijas, sākotnējā risinājuma ģeneratoru, risinājuma apkārtnes funkciju un risinājuma novērtēšanas funkcijas.

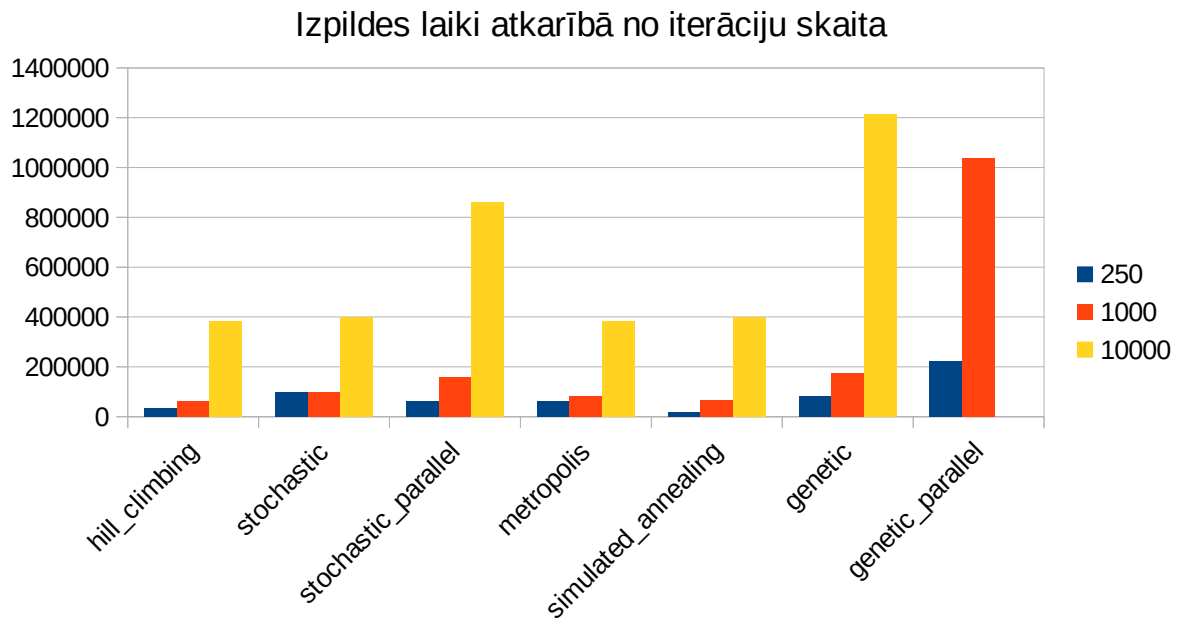
Optimizatora modulis satur optimizācijas darbu pārraugu/pārvaldnieku/supervizora procesu, kas pieņem jaunus optimizācijas darbu pieprasījumus, glabā rezultātus tiem darbiem kas ir beigušies un uzrauga nepabeigtos optimizācijas darbus. Papildus modulis satur pašus optimizācijas algoritmus, kas implementēti kā atsevišķi procesi.

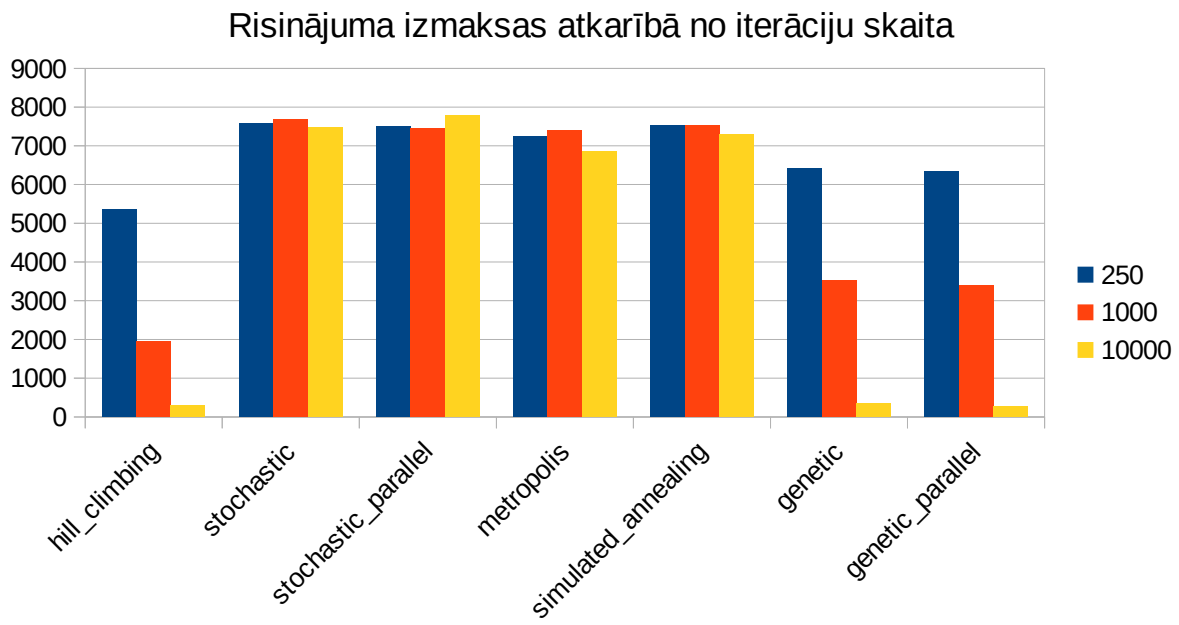
Servera modulis satur serveri un risinājuma HTTP saskarni. Papildus ir viena indeksa HTML lapa ar ļoti daudz JavaScript koda un viena HTTP 404 kļūdas paziņojuma lapa. Risinājums Erlang stilā neizmanto nekādu ievada validāciju, visam nepareizajam ievadam ir ļauts atteicēt serveri.

Testēšana

Testēšanā izmantoti šādi optimizācijas algoritmi:

Kalnā kāpējs, kas katrā iterācijā no risinājuma izveidos jaunu risinājumu un paturēs šo risinājumu nākamajām iterācijām ja tas ir labāks par iepriekšējo. Beigās izdod palikušo risinājumu.
Stohastiskais, kas ir tā kā kalnā kāpējs, tikai vienmēr paturēs jauno risinājumu nākamajām iterācijām. Papildus tas atcerēsies vislabāko risinājumu un to beigās izdos.
Stohastiskais paralēlais, kas ir tas pats stohastiskais, tikai atsevišķos procesos uzģenerēs 4 risinājumus, nevis tikai vienu.
Metropolisa, kas izmanto Metropolisa algoritmu.
Simulētās apsaldēšanas, kas izmanto *Simulated Annealing* algoritmu.
Ģenētiskais, kas sākumā no sākuma risinājuma uzģenerēs 32 citus, tad tas katrā iterācijā paturēs 8 labākos risinājumus un no katra no tiem uzģenerēs 4 jaunus risinājumus. Beigās izdod labāko atlikušo risinājumu.
Ģenētiskais paralēlais, kas strādā tā pat kā parastais ģenētiskais, tikai tas jauno risinājumu ģenerāciju izpildīs 8 atsevišķos procesos.





Savādi ka paralēlajiem algoritmiem sanāca tā ka tie ir lēnāki. Ģenētiskais paralēlais vispār bija tik lēns ka man vajadzēja tam izdzēst no grafika vienu mērījumu, jo tas neietilpa grafikā.

Vienīgā atšķirība no parastajām versijām ir tāda ka paralēlie algoritmi palaiž jaunus procesus. Varbūt ja manam datoram vai nu būtu vairāk procesora kodoli, vai arī mana apkārtnes un izmaksas funkcijas būtu lēnākas, tad varbūt procesa izveidošanas izmaksas atsvērtu paralelizācijā ietaupīto laiku.

Github

https://github.com/racenis/DatZ6103--MAZAIS-un-LIELAIS-praktiskais-darbs

Dockerhub

https://hub.docker.com/r/racenis/sia-snukstings-brunuvardes