ავტორის სტილი დაცულია

ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

ნათელა ანანიაშვილი

ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი
კომპიუტერულ მეცნიერებათა დეპარტამენტი

უწყვეტი და დისკრეტული ოპტიმიზაციის ზოგიერთი ალგორითმის მოდიფიკაცია მათი შინაარსობრივი და ვიზუალური ანალიზის საფუძველზე

ს ა დ ო ქ ტ ო რ ო დ ი ს ე რ ტ ა ც ი ა

ხელმძღვანელები:

სადოქტორო პროგრამის ხელმძღვანელი:
თსუ პროფესორი
ფიზიკა-მათემატიკის მეცნიერებათა დოქტორი გია სირბილაძე

სამეცნიერო ხელმძღვანელი:
თსუ პროფესორი
ფიზიკა-მათემატიკის მეცნიერებათა დოქტორი კობა გელაშვილი

თბილისი 2018

Ivane Javakhishvili Tbilisi Statate University

Natela Ananiashvili

Faculty of Exact and Natural Sciences
Departament of Computer Science

Modification of Some Continuous and Discreet Optimization Algorithms Based on Their Content and Visual Analysis

PhD Thesis

Supervisors:

Supervisor of the Doctoral Program
Professor, TSU
Doctor of science in Physics and Mathematics — Gia Sirbiladze

Supervisor of the Doctoral Program
Professor, TSU
Doctor of science in Physics and Mathematics — Koba Gelashvili

Tbilisi 2018

ანოტაცია

ნაშრომის ძირითადი მიზანია გლობალური ოპტიმიზაციის ამოცანების ზუსტი და მიახლოებითი ამონახსნებისათვის ახალი გაუმჯობესებული ალგორითმების შექმნა და თანამედროვე პროგრამული ინსტრუმენტების გამოყენებით მოქნილი პროგრამული რეალიზაცია.

ნაშრომში განიხილება მძიმე ბირთვის მეთოდის ახალი უწყვეტი დაგვიანებული მოდელი (Gelashvili, et al., 2013). ”მძიმე ბირთვი” წარმოადგენს გლუვი ფუნქციის მინიმიზაციის

$$f(x) \to min, \quad x \in R_n, \qquad (1)$$

ამოხსნის კარგად ცნობილ და საკმაოდ ეფექტურ მეთოდს. ეს მეთოდი თავისი შინაარსით მრავალფუნქციურია: იგი გამოიყენება როგორც ”პირველივე” ლოკალური მინიმალის სწრაფი მიღწევისთვის, ასევე გლობალური (ან კარგი ლოკალური) მინიმალის განსაზღვრისთვის. მძიმე ბირთვის თემატიკა მრავალმხრივ ინტერესს იწვევს. იგი საინტერესოა მათემატიკური მოდელირების თვალსაზრისით, რადგან დინამიკური პროცესის წონასწორული მდგომარეობისკენ სწრაფვის იდეის რეალიზება შესაძლოა განხორციელდეს ერთმანეთისგან საგრძნობლად განსხვავებული მათემატიკური მოდელების სახით (Brown, et al., 1950; Brown, 1951; Polyak, 1987; Eiselt, et al., 2000; Soo Y. Chang, et al., 1989). ამჟამად, უპირობო ოპტიმიზაციის საკმაოდ ზოგად შემთხვევებს (მაგ. გლუვი, ამოზნექილი, კვაზიამოზნექილი მიზნის ფუნქციები) სისტემატურად ეძღვნება სამეცნიერო და პრაქტიკული თვალსაზრისით აქტუალური სამეცნიერო ნაშრომები (Goudou, et al., 2009; Hajba, 2011; Cabot A., 2009; Bhaya, 2010; Paul-Emile, December 2009), რომლებშიც მათემატიკურ მოდელს წარმოადგენს პირველი ან მეორე რიგის ჩვეულებრივი დიფერენციალური სისტემები. უშუალოდ ექსპერიმენტებით შეგვიძლია დავრწმუნდეთ, რომ მძიმე ბირთვის მეთოდი წარმატებით შეიძლება იქნას გამოყენებული არაგლუვი ფუნქციების მინიმიზაციისთვის. ახალი ალგორითმის პროგრამული რეალიზაცია გაცილებით მოქნილია, აქვს პარამეტრების შერჩევის უფრო მდიდარი საშუალებები და ტესტებზე სტაბილურად გაცილებით უკეთეს შედეგებს აჩვენებს მძიმე ბირთვის სტანდარტულ მეთოდთან შედარებით.

ნაშრომში შემოთავაზებულია ოპტიმიზაციის (1) ამოცანის ამოხსნის ორი მოდიფიცირებული გენეტიკური ალგორითმი, რომლებშიც გამოყენებულია ახალი სელექციის, შეჯვარებისა და მუტაციის ოპერატორები. ეს ალგორითმები სატესტო ამოცანებზე საიმედოდ მუშაობს. აღსანიშნავია, რომ რამდენიმე ამოცანისთვის, რომლისთვისაც კლასიკური მეთოდები ვერ აღწევს შედეგს, ჩვენი ალგორითმები იღებს პასუხს. სისწრაფე სრულიად შეესაბამება გენეტიკური ალგორითმებისთვის მიღებულ (არამკაცრ) სტანდარტებს.

ნაშრომში აგრეთვე განხილულია ისეთი მნიშვნელოვანი ამოცანები, როგორიცაა: უმცირესი დაყოფისა და დაფარვის ამოცანები. როგორც ცნობილია, უმცირესი დაყოფის და დაფარვის ამოცანები მიეკუთვნებიან NP-რთული ამოცანების კლასს (Garey, et al., 1979). დღეისათვის ამ ამოცანების ზუსტი ამოხსნისათვის, გარდა მათი კერძო შემთხვევებისა არ

არსებობს ეფექტური ალგორითმი. ამოხსნის დრო დამოკიდებულია ამოცანის განზომილებაზე, ის შესაძლოა ძლიერ გაიზარდოს ამოცანის განზომილების ზრდასთან ერთად და შეუძლებელი გახდეს ოპტიმალური მნიშვნელობის მიღება რეალურ დროში. პრაქტიკაში ამ ამოცანების ამოხსნის საჭიროება ჩნდება მაგალითად ტრანსპორტის დანიშვნისას (Christofides, 1986), ენერგოსისტემის წყაროების განთავსებისას (Minieka, 1978), ინფორმაციის გადაცემისას (Berge, 1958), მომსახურების პუნქტების განთავსებისას და სხვა. ამ ამოცანების ზუსტი ამოხსნისას იყენებენ შტოების და საზღვრების მეთოდს (Christofides, 1986; Minieka, 1978; Berge, 1958; Balas E., 1980; Fisher M. L., 1990; Ananiashvili, 2014; Ananiashvili, 2015). დღეისათვის დიდი განზომილების ამოცანების მიახლოებითი ამოხსნებისათვის ხშირად იყენებენ: გენეტიკურ ალგორითმებს (Beaslay J. E., 1992; Eremeev, 2000), ნეირონულ ქსელებს (Grossman, et al., 1997) და სხვა ევრისტიულ ალგორითმებს. ევრისტიული ალგორითმები დროის მისაღებ ინტერვალში პოულობენ ოპტიმალურთან მიახლოებულ ამონახსნებს. მიახლოებითი ალგორითმები უმეტესად გულისხმობენ ნაწილობრივ გადარჩევას.

ნაშრომში შემოთავაზებულია უმცირესი დაყოფისა და დაფარვის ამოცანების ზუსტი ამოხსნის მოდიფიცირებული მეთოდი. პროგრამირების ტექნიკის არაარსებითი გართულების ხარჯზე შესაძლებელი გახდა გამოყენებული მეხსიერების, ასევე გამოთვლისათვის საჭირო დროის მიახლოებით $32 \cdot M$-ჯერ შემცირება, სადაც M დამფარავი ქვესიმრავლეების რაოდენობაა. ამ მიზნით ჯერ დაყოფის მატრიცა ჩაიწერა კომპაქტურად და შემდეგ მიღებული მატრიცის ელემენტებზე ძირითადი ოპერაციები შესრულდა ლოგიკური ოპერატორებით. ამ ამოცანების ამოსახსნელად დაიწერა პროგრამების კომპლექსი ალგორითმულ ენა C++-ზე. კომპლექსი აპრობირებულია ლიტერატურაში კარგად ცნობილ რეალურ კომბინატორული ტიპის ამოცანებზე, რომლებიც შეგვიძლია ავიღოთ სატესტო ამოცანების ცნობილი ელექტრონული ბიბლიოთეკიდან: Or-Library-დან (Beasley, 1990). დროის მისაღებ ინტერვალებში მიღებულია კარგი შედეგები დაყოფის ამოცანებისათვის და მისაღები შედეგები დაფარვის ამოცანებისათვის.

შემოთავაზებულია დაფარვის და მრავალჯერადი დაფარვის ამოცანების ამოხსნის მოდიფიცირებული გენეტიკური ალგორითმები, რომლებიც დაფუძნებულია კლასიკურ გენეტიკურ ალგორითმზე შეჯვარების და მუტაციის ახალი ოპერატორებით. შექმნილია პროგრამული კომპლექსები Matlab-ის გარემოში. ალგორითმები გასინჯულია სატესტო ამოცანებზე, მიღებულია სტაბილური შედეგები. კერძოდ, ყველა ჩვენს მიერ გასინჯული სატესტო ამოცანის ამოხსნა დასრულდა შედეგიანად, შედეგები რიგ შემთხვევებში დაემთხვა ზუსტ ამონახსნებს, დანარჩენ შემთხვევებში ახლოს არს ზუსტ ამონახსნებთან.

ნაშრომში შემოთავაზებული ალგორითმებისათვის ნაბიჯ-ნაბიჯ არის დათვლილი თითოეული ბიჯისთვის საჭირო პირობითი დრო და შემდეგ შეფასებულია მთლიანი ალგორითმის სარეალიზაციო დრო.

Annotation

The primary goal of the present work is to create new optimized algorithms for exact and approximate solutions of global optimization problems and flexible programming impelmentation with modern software tools.

A new continuous model of the heavy ball method (Gelashvili, et al., 2013) is considered. The heavy ball is a well-known and quite effective method to solve the minimization problem of the smooth function

$$f(x) \to min, \quad x \in R_n, \qquad (1)$$

This method is multifunctional in its essence used for both attaining the very "first" local minimum and identifying the global (or good local) minimum. The heavy ball is an interesting topic in many aspects. It is interesting from the perspective of mathematical modeling, since the idea of the dynamic process converging to equilibrium can be realized through quite different mathematical models (Brown, et al., 1950; Brown, 1951, Polyak, 1987; Eiselt, et al., 2000 Soo Y. Chang, et al., 1989). Today, serious scientific and practical scientific papers (Goudou, et al., 2009; Hajba, 2011; Cabot A., 2009; Bhaya, 2010; Paul-Emile, December 2009) are systematically devoted to general cases of unconstrained optimization (e.g. smooth, convex, quasi-convex objective functions), where the ordinary differential systems of the first or second order are used as mathematical models. The experiments show that the heavy ball method can be successfully used for minimization of the non-smooth functions. Programming realization of the new algorithm is much more flexible providing richer option for selecting the parameters and showing much better test results compared to the standard heavy ball method.

In the present work, two modified genetic algorithms are suggested for solving the optimization problem (1), where the new operators of selection, crossover and mutation are used. Algorithms permanently show good results on test problems.

In the present work, the minimal set partition and covering problems are also considered. As it is known, the minimal set partition and covering problems belong to the class of complex NP problems (Garey, et al., 1979). Today, the effective algorithm for exact solution of these problems, except for their private cases, does not exist. The time of solving the problem depends on the scale of the problem, which may significantly increase with the increase of the problem scale making it impossible to receive the optimum value in real time. Solution of these problems has applications, for example, in transport scheduling (Christofides, 1986), locating the sources of power systems (Minneka, 1978), data transfer (Berge, 1958), locating the service centers, etc. For exact solution of these problems the branch and bound method is used (Christofides, 1986; Minneka, 1978; Berge, 1958; Balas E., 1980; Fisher M. L., 1990; Ananiashvili, 2014; Ananiashvili, 2015). Today, genetic algorithms (Beaslay J. E., 1992; Eremeev, 2000), neuron networks (Grossman, et al., 1997) and other heuristic algorithms are often used to receive approximate solutions for the large-scale problems. Heuristic algorithms can find near-optimal solution in the acceptable interval of time. In most cases, the approximate algorithms mostly imply partial selection.

The modified method for exact solution of minimal set partition and covering problems is suggested in the work. Non-essential complication of programming technique reduced the necessary memory and decreased the computational time approximately 32-M times, where M is the number of covering subsets. To that end, the partition matrix was compactly written, and then the basic operations were performed on the matrix elements with logical operators. A complex of programs was written in algorithmic language C++ to solve these problems. The complex is tested on real combinative problems known in literature, which can be taken from the well-known electronic library of test problems: Or-Library (Beasley, 1990). Good results are obtained for set partition problems and acceptable results for set coverage problems in suitable time intervals.

Modified genetic algorithms for solving the set cover and set multi-cover problems are proposed, which are based on new operators of crossover and mutation to new algorithms. The software complexes are developed in Matlab. The algorithms are checked on test problems permanently showing good results. The conventional time required for each step of the proposed algorithms is calculated step-by-step, and then the entire time of realization of the algorithm is estimated.

შინაარსი

ცხრილების ნუსხა

ნახაზების ნუსხა

შესავალი

რეალურ სამყაროში არსებობს ბევრი მნიშვნელოვანი პრობლემა, რომლებიც საჭიროებენ ოპტიმიზირებას. პრაქტიკული და თეორიული ხასიათის ამოცანების უმეტესობა გულისხმობს საუკეთესო კონფიგურაციის ან პარამეტრების ამორჩევას, რაიმე მიზნის მისაღწევად. დღეისათვის არსებობს როგორც ამ ამოცანების, ასევე მათი ამოხსნის მეთოდების კლასიფიკაცია. ოპტიმიზაციის ამოცანები ბუნებრივად იყოფიან ორ კლასად: ამოცანები უწყვეტი ცვლადებით და ამოცანები დისკრეტული ცვლადებით, ამ უკანასკნელთ კომბინატორულ ამოცანებს უწოდებენ. უწყვეტ ამოცანებში საზოგადოდ ეძებენ მნიშვნელობებს ნამდვილ რიცხვთა სიმრავლიდან. კომბინატორულ ამოცანებში, კი რაიმე ობიექტს ან ობიექტებს რომელიმე სასრული ან შესაძლოა თვლადი სიმრავლიდან. ასეთი ობიექტები შესაძლოა იყოს: მთელი რიცხვი, სიმრავლე, გადანაცვლება ან გრაფი. დისკრეტული და უწყვეტი ამოცანების ამოხსნის გზები როგორც წესი, არის განსხვავებული. მიუხედავად განსხავებულობისა, დღეისათვის არსებობენ ისეთი ალგორითმები, რომლებიც ორივე ტიპის ამოცანების ამოხსნისას შესაძლოა იყოს გამოყენებული - ასეთია მაგალითად წრფივი პროგრამირების ამოცანის ამოხსნის მეთოდები, ასევე გენეტიკური ალგორითმები. ეს უკანასკნელი დღესდღეობით დიდი პოპულარობით სარგებლობს. წინამდებარე ნაშრომში სწორედ ამ კუთხით "გავაერთიანებთ" ამ ორი კლასის ამოცანებს. არსებობს ალგორითმები, რომლებიც წარმატებით პოულობენ ლოკალურ ოპტიმუმს, მაგრამ უფრო ხშირად საინტერესოა იმის გარკვევა, წარმოადგენს თუ არა მიღებული ლოკალური ოპტიმუმი გლობალურსაც. სწორედ გლობალური ოპტიმუმის ძიების რამდენიმე ამოცანის ამოხსნაა შემოთავაზებული წინამდებარე ნაშრომში.

გლობალური ოპტიმიზაციის ამოცანები პრაქტიკიდან მოსული ამოცანებია და მათთვის რეალურთან მიახლოებული მათემატიკური მოდელის შექმნა და ამოხსნის გზების ძიება ყოველთვის მნიშვნელოვანია, მათ სისტემატურად ეძღვნება სამეცნიერო და პრაქტიკული თვალსაზრისით აქტუალური სამეცნიერო ნაშრომები. ასეთია მაგალითად, ისეთი პრობლემები, როგორიცაა დაგეგმვა და მარშრუტიზაცია, საინჟინრო პრობლემები და სხვა. ოპტიმიზაციის ამოცანების ამოსახსნელად არსებობს ბევრი მეთოდი, ყოველი ახალი მეთოდი არის მცდელობა იმისა, რომ გააუმჯობესონ ამონახსნის მნიშვნელობა და აგრეთვე ამოხსნის სისწრაფე. ამავე დროს ზოგიერთი ამოცანის ამოხსნისას უკვე არსებული ტიპური მეთოდები შესაძლოა გაცილებით ნაკლებად ეფექტური იყოს, ვიდრე სპეციფიური მეთოდები, რომელიც შემუშავებულია იმავე ამოცანებისათვის. ახალი მეთოდების მუშაობის სიზუსტე ფასდება იმ მიღებული შედეგებით, რომლებსაც ისინი გვაძლევენ ცნობილ სატესტო ამოცანებზე აპრობაციისას. წინამდებარე ნაშრომში ორივე კლასის: უწყვეტი და დისკრეტული ამოცანებისათვის გამოყენებულია თანამედროვე სატესტო ამოცანები. ამასთან აღებულია მარტივი და ასევე რთული ტესტები: უწყვეტი ამოცანებისათვის აღებულია ფუნქციები (Jorge J. More, et al., 1981)-დან, ხოლო კომბინატორული ამოცანებისათვის აღებულია ამოცანები Or-Library (Beasley, 1990)-დან.

დღეისათვის გლობალური ოპტიმიზაციის ამოცანების ალგორითმების ჩამონათვალი საკმაოდ ფართოა:

1. ერთი ლოკალური მინიმუმიდან მეორეში გადასვლის მეთოდები. აქ შედის ალგორითმები, რომლებიც ეფუძნება ლოკალური მინიმუმის წერტილამდე დაშვებას და შემდეგ ახალ ლოკალურ მინიმუმში გადასვლების მონაცვლეობას, როგორიცაა მძიმე ბირთვის მეთოდი; მეთოდები, რომლებიც ეყრდნობიან დიფერენციალური განტოლებების ამოხსნას.
2. ალგორითმები, რომლებიც ეფუძნებიან განზომილების რედუქციას, როდესაც საწყისი ამოცანა შეცვლილია რამდენიმე მცირე განზომილების ამოცანით.
3. მეთოდები, რომლებიც ეფუძნებიან საწყისი გლობალური ოპტიმიზაციის ამოცანის შეცვლას რომელიმე სხვა ამოცანით. მაგალითად, როგორიცაა ინტერპოლაცია, ექსტრაპოლაცია, აპროქსიმაცია.
4. შემთხვევითი ძიების მეთოდები, მეთოდები, რომლებიც ემყარებიან ცოცხალი ბუნების კანონზომიერებებს: გენეტიკური ალგორითმები, ევოლუციური ალგორითმები, ჭიანჭველათა ნაკადის (კოლონიის) ალგორითმი, მონტე-კარლოს მეთოდი.
5. აგრეთვე მარკოვის ალგორითმები, შტოებისა და საზღვრების მეთოდი და სხვა.

მეთოდების ბოლო ჯგუფი დღეისათვის ძალიან პოპულარული გახდა, ვინაიდან ისინი დიდი ალბათობით თავს ართმევენ ოპტიმიზაციის მრავალ ექსტრემალურ ამოცანას.

ოპტიმიზაციის მეთოდის არჩევა ხშირად განისაზღვრება ამოსახსნელი ამოცანის თავისებურებით, მიღებული ამონახსნი კი მნიშვნელოვნად არის დამოკიდებული ალგორითმის მახასიათებლებზე: კრებადობის სიჩქარეზე, მოცემული სიზუსტის მიღწევადობაზე და ა. შ. შესაძლოა აღმოჩნდეს, რომ ამორჩეული მეთოდი უკვე აღარ არის საკმარისი ახალი ამოცანის ამოსახსნელად, მაგალითად, როდესაც ახალი ამოცანის განზომილება მნიშვნელოვნად დიდია მანამდე ამოხსნილი ამოცანების განზომილებასთან შედარებით, ამ დროს დგება მეთოდის მოდიფიცირების აუცილებლობა.

უწყვეტი არაწრფივი ოპტიმიზაციის ამოცანების ამოხსნისას მთავარი სირთულე მდგომარეობს ამ ფუნქციების არაწრფივობაში და ამოცანების განზომილებაში. ხშირად კლასიკური მეთოდებით ოპტიმალური მნიშვნელობის ძებნა უშედეგოდ მთავრდება. სწორედ ამიტომ შეიქმნა გენეტიკური ალგორითმები. გენეტიკური ალგორითმი შემთხვევითი ძიების ერთ-ერთი სახეა.

ოპტიმიზაციის კლასიკური ალგორითმების და მათი მოდიფიკაციების უმეტესობა ძიებას იწყებს საძიებელ სივრცეში აღებული ერთი რომელიმე საწყისი წერტილიდან, დეტერმინირებულად ითვლიან მიზნის ფუნქციის გრადიენტს ან უფრო მაღალი რიგის წარმოებულს (Polyak, 1987). მცირე ბიჯით ხდება გადასვლა გრადიენტის მიმართულებით. შემდეგ პროცესი მეორდება თავიდან. ამ ალგორითმებით ამონახსნი თანდათან უმჯობესდება. ასეთი მიდგომის ნაკლი მდგომარეობს იმაში, რომ შესაძლოა საბოლოოდ მოვხვდეთ ლოკალურ ექსტრემუმში.

გენეტიკური ალგორითმები შეიქმნა ცოცხალ ორგანიზმებზე დაკვირვების შედეგად, ამ ალგორითმებში ყველაფერი გავს ბიოლოგიური პოპულაციების განვითარების პროცესს (Randy L. Haupt, 2004; Рутковская , и др., 2006; Ananiashvili, 2015), ისინი ასახავენ ისეთ კანონებს, როგორიცაა: ბუნებრივი გადარჩევა, გენეტიკური ინფორმაციის მემკვიდრეობით მიღება, ორგანიზმის უნარი შეეგუოს საციცოცხლო გარემო პირობებს. ევოლუციის და

სელექციის პროცესების კოპირების მცდელობის იდეა წარმოიშვა გასული საუკუნის სამოც-სამოცდაათიან წლებში და ეკუთვნის ჰოლანდს (Holland, 1975). შესაბამისად გენეტიკურ ალგორითმებში გამოყენებული ტერმინოლოგიაც შეესაბამება ბიოლოგიაში მიღებულ ტერმინებს. დღეისათვის გენეტიკური ალგორითმების გამოყენების არეალი სულ უფრო მეტად ფართოვდება. გენეტიკური ალგორითმი ძიებას ერთდროულად რამდენიმე მიმართულებით აწარმოებს (Randy L. Haupt, 2004; Рутковская , и др., 2006). ერთი პოპულაციიდან მეორეზე გადასვლა კი საშუალებას იძლევა თავიდან ავიცილოთ ლოკალურ ექსტრემუმში მოხვედრა.

გენეტიკურ ალგორითმებს კლასიკურ ოპტიმიზაციის მეთოდებთან შედარებით ორი ძირითადი უპირატესობა აქვთ:

1. გენეტიკურ ალგორითმებს არ გააჩნია მნიშვნელოვანი მოთხოვნები მიზნის ფუნქციის მიმართ. კვლევის დროს არ არის მოდელის გამარტივების საჭიროება იმისათვის, რომ ხელოვნურად შეიქმნეს ხელმისაწვდომი მათემატიკური მეთოდების გამოყენების საშუალება. ამასთან შეიძლება გვქონდეს სრულიად სხვადასხვა სახის მიზნის ფუნქციები და შეზღუდვები: წრფივი და არაწრფივი, დისკრეტულ, უწყვეტ და შერეულ უნივერსალურ სიმრავლეზე განსაზღვრული.
2. კლასიკური მეთოდების გამოყენებისას გლობალური ოპტიმუმი შეიძლება მოიძებნოს მხოლოდ იმ შემთხვევაში, როდესაც გვაქვს ამოზნექილობის პირობა. გენეტიკური ალგორითმების ევოლუციურ ოპერაციებს ამ პირობის დროსაც და მისი შეუსრულებლობის შემთხვევაშიც ეფექტურად შეუძლიათ მოძებნონ გლობალური ოპტიმუმი.

დისერტაციის მოკლე შინაარსი თავების მიხედვით

დისერტაციის **პირველ თავში:** "მძიმე ბირთვის მეთოდის ერთი მოდიფიკაციის შესახებ" თავდაპირველად განიხილება სხვადასხვა ცნობილი ალგორითმი, რომელთა ანალიზის საფუძველზე აიგება მძიმე ბირთვის მოდიფიცირებული მოდელი და მტკიცდება ახალი მოდელის კრებადობა.

პირველი თავის **პირველ პარაგრაფში** მინიმიზაციის (1) ამოცანის მარტივი კვადრატული ფორმის მინიმიზების მაგალითზე მოყვანილია "ზიგზაგებიანი" ალგორითმების გეომეტრიული გააზრება.

პირველ პუნქტში საუბარია კვადრატული ფორმის მინიმიზაციაზე, როდესაც შესაბამისი მატრიცი არის სიმეტრიული და დადებითად განსაზღვრული.

მეორე პუნქტში მოცემულ საილუსტრაციო მაგალითზე სამგანზომილებიან სივრცეში აგებულია დონის წირები და შესაბამისი ინვარიანტული ქვესივრცეები.

მესამე პუნქტში მარტივი კვადრატული ფორმის მაგალითისათვის აგებულია სხვადასხვა მეთოდის შესაბამისი ტრაექტორიები ინვარიანტულ ქვესივრცეებთან ერთად. ამ ექსკურსის შემდეგ კი დასმულია კითხვა-რის ხარჯზე არის მძიმე ბირთვი ბევრად უფრო სწრაფი, ვიდრე უსწრაფესი დაშვების, ან თუნდაც გრადიენტული დაშვების მეთოდი? და იქვე გაკეთებულია დასკვნა: თუ მძიმე ბირთვის მეთოდში ხახუნს ავიღებთ დიდს, მაშინ ის მეტად ემსგავსება გრადიენტული დაშვების მეთოდს, რაც მომგებიანია კრებადობის

თვალსაზრისით, სისწრაფის თვალსაზრისით კი უმჯობესია ის გავდეს შეუღლებულ გრადიენტთა მეთოდს.

პირველი თავის **მეორე პარაგრაფში** განიხილება მძიმე ბირთვის უწყვეტი მოდელი, რომელიც აღიწერება დაგვიანებული ნეიტრალური ტიპის სისტემით. განიხილება აგრეთვე მისი განზოგადება კარგი ლოკალური მინიმალის ეფექტური განსაზღვრის მიზნით.

მეორე პუნქტი ეთმობა მართვადი მძიმე ბირთვის ალგორითმის კრებადობის დამტკიცებას. მართვად მძიმე ბირთვს, ჩვეულებრივთან შედარებით, გააჩნია მთელი რიგი უპირატესობები, რომელთა გამოვლენას ეთმობა ეს და შემდეგი პარაგრაფები. მძიმე ბირთვის მეთოდის კრებადობის დამტკიცება გაცილებით რთულია, მაგალითად, სტანდარტული გრადიენტული მეთოდის (ფიქსირებული ბიჯით) კრებადობის დამტკიცებასთან შედარებით. ჩვენს მიერ მოდიფიცირებული მეთოდის კრებადობის დამტკიცება სირთულის დონით პრაქტიკულად არ განსხვავდება სტანდარტული გრადიენტული მეთოდის კრებადობის დამტკიცებისგან (იხ. თეორემა 1.1).

მესამე პუნქტი ეძღვნება ახალი მიდგომის გეომეტრიულ გააზრებას, ხოლო ბოლო პუნქტი აღწერს ახალ ალგორითმს და წარმოგვიდგენს მისი პროგრამული რეალიზაციის ტესტირების შედეგებს რამდენიმე კარგად ცნობილ ამოცანაში.

წარმოდგენილი ნაშრომის მიზანს წარმოადგენს ახალი ალგორითმისთვის დარგის სპეციალისტების ყურადღების მიპყრობა და არა მისი ამომწურავი შესწავლა, ამიტომ, ჩვენ ვამტკიცებთ მხოლოდ მოდიფიცირებული მეთოდის კრებადობას და ვკმაყოფილდებით ტესტების შედეგების ჩვენებით, რაც საკმაოდ მრავლისმეტყველია.

დისერტაციის **მეორე თავში:** "ოპტიმიზაციის ამოცანის ამოხსნა გენეტიკური ალგორითმის გამოყენებით" შემოთავაზებულია ოპტიმიზაციის $f(x) \to \min, x \in R_n$ ამოცანის ამოხსნის გენეტიკური ალგორითმი, რომელიც დაფუძნებულია კლასიკური გენეტიკური ალგორითმის ძირითად პრინციპებზე. აღწერილია აგრეთვე სარგებლიანობის ფუნქციის განსაზღვრის, შეჯვარებისა და მუტაციის რეალიზების განსხვავებული მიდგომები.

მეორე თავის **მეორე პარაგრაფში** დასმულია პრობლემა და საუბარია მის გადაჭრის გზებზე.

პირველი პარაგრაფის **პირველ პუნქტში** მოყვანილია საწყისი პოპულაციის არჩევის ალგორითმი 2.1.1. არჩეულია კოდირების ორობითი სქემა და მოცემულია საწყისი პოპულაციის არჩევისათვის საჭირო ბიტების განსაზღვრის უტოლობა (2.1) და შემდეგ ამ უტოლობით განსაზღვრული რაოდენობის ბიტებში ვწერთ შემთხვევით რიცხვებს-ინდივიდებს. ვინაიდან ჩვენი ინდივიდები ინახება Unsigned Long ტიპის მასივში (რომელის ელემენტებიც 32 ბიტს იკავებენ მეხსიერებაში), ამ მასივის თითოეულ ელემენტში ინდივიდისთვის საჭირო შევსებული დაბალი თანრიგების შემდეგ დარჩენილი მაღალი თანრიგები შეივსება ნულებით.

მეორე პუნქტში განხილულია სელექცია. განსაზღვრულია სარგებლიანობის ფუნქცია და ამ ფუნქციით შეფასებულია ინდივიდები. შემდეგ კი მშობელი ინდივიდების შესარჩევად გამოყენებულია ე. წ. "რულეტკის" (Гладков , и др., 2004) მეთოდი.

მესამე პუნქტში განხილულია შეჯვარების სქემა. გამოყენებულია ერთწერტილიანი შეჯვარება. თუ ორობითი კოდების მიმართ პირდაპირ გამოვიყენებთ შეჯვარებას, მაშინ

შესაძლებელია რომ შთამომავალი ინდივიდების მნიშვნელობები ნახტომისებურად შეიცვალოს, ამის თავიდან აცილების მიზნით მშობლებისათვის ჯერ ვიყენებთ გრეის კოდირებას და შემდეგ ვახდენთ შეჯვარებას.

მეოთხე პუნქტში მოყვანილია ალგორითმი 2.1.3 მუტაციის ოპერატორის განხორციელებისათვის.

მეხუთე პუნქტში მოცემულია ექსპერიმენტების შედეგები ცხრილი 2.1.1 და 2.1.2 -ის სახით ცნობილი სატესტო ამოცანებისათვის. ცხრილებიდან ჩანს შემოთავაზებული ალგორითმის ეფექტურობა.

მეორე თავის **მეორე პარაგრაფში** მოყვანილია (2.1) ამოცანის ამოხსნის მეორე მოდიფიცირებული გენეტიკური ალგორითმი.

აქ საწყისი პოპულაციის არჩევა და შეჯვარება იმავე ალგორითმებით ხდება, როგორც პირველ პარაგრაფში მოყვანილ ალგორითმში,

პირველ პუნქტში განხილულია სელექცია. გამოყენებულია ინბრიდინგის მეთოდი.

მეორე პუნქტში მოყვანილია ალგორითმი 2.2.1 მუტაციის ოპერატორის განხორციელებისათვის.

მესამე პუნქტში მოცემულია ექსპერიმენტების შედეგები ცხრილი 2.2.1 და 2.2.2 -ის სახით. სადაც მოცემულია შემოთავაზებული ალგორითმის აპრობაცია ცნობილ სატესტო ამოცანებისათვის. აქვე მოყვანილია ერთი საილუსტრაციო მაგალითი (Bird ფუნქციისათვის).

დისერტაციის **მესამე თავში:** "უმცირესი დაყოფისა და დაფარვის ამოცანების ზუსტი ამოხსნის შესახებ", დასმულია ამოცანა და განხილულია პრობლემის გადაწყვეტის გზა ზუსტი ალგორითმის გამოყენებით.

პირველ პარაგრაფში დასმულია პრობლემა, მოყვანილია ამოცანის ფორმალიზებული მოდელი;

მეორე პარაგრაფში საუბარია უმცირესი დაყოფის ამოცანის ამოხსნის ზუსტ ალგორითმზე, შემდეგ შემოთავაზებულია დაყოფის მატრიცის კომპაქტურად ჩაწერის იდეა და მისი განხორციელებისათვის საჭირო ალგორითმი. იდეა კი შემდეგში მდგომარეობს: დაყოფის მატრიცის სვეტები ჩაიწერება კომპაქტურად, თუ მატრიცის სტრიქონების რაოდენობაა N მაშინ თითოეული სტრიქონი "შეიფუთება" [N/32]+1 (აქ [N/32] აღნიშნავს N-ის 32-ზე გაყოფის მთელ ნაწილს) რაოდენობის Unsigned Long ტიპის სტრიქონში. რაც მოგვცემს მეხსიერების ეკონომიას და თვლის დროც შემცირდება საგრძნობლად - მინიმუმ 32-ჯერ. შევნიშნოთ რომ, დიდი განზომილების ამოცანებში შესაძლებელია Unsigned Long ტიპის ნაცვალად გამოყენებული იქნეს Unsigned Long Long ტიპი, მაშინ [N/32]+1 -ის ნაცვლად დაფარვის მატრიცაში გვექნება [N/64]+1 სტრიქონი. მეორე პარაგრაფში მოყვანილია "შეფუთვის" და შემდეგ გაშლის ("განფუთვის") ალგორითმები. აქვე მოყვანილია პროგრამული კომპლექსის აპრობაციის შედეგები უმცირესი დაყოფის სატესტო ამოცანებისათვის (იხ. ცხრილი 3.1). ამოცანები კი აღებულია ცნობილი ბიბლიოთეკიდან: Or-Library-ის სატესტო ფაილებიდან.

მესამე პარაგრაფში დაყოფის ალგორითმის იდეა, რომელიც მოყვანილი იყო პირველ პარაგრაფში, გამოყენებულია დაფარვის ამოცანისათვის. ვინაიდან აქ აღარ გვაქვს სვეტების თანაუკვეთობის მოთხოვნა საჭიროა ალგორითმის მოდიფიკაცია. კერძოდ, თუ დაყოფის შემთხვევაში ყოველი $S_j = \{r_{j_1}, r_{j_2}, \dots, r_{j_k}\}$ სიმრავლის შესაბამისი სვეტი, შედიოდა მინიმალური ინდექსის მქონე p-ურ ბლოკში: $p = min(r_{j_1}, r_{j_2}, \dots, r_{j_k})$, ახლა საჭირო გახდა მისი შეტანა ყველა $r_{j_1}, r_{j_2}, \dots, r_{j_k}$ ნომრიან ბლოკებში. ცხადია ამის გამო ამოცანის განზომილება (დაფარვის მატრიცის სვეტების რაოდენობა) საგრძნობლად გაიზარდა, რამაც თავის მხრივ გამოიწვია თვლის დროის გაზრდა. ალგორითმის დანარჩენი ნაწილი იგივე დარჩა, რაც დაყოფაში გვქონდა. აქვე მოყვანილია ექსპერიმენტების შედეგები დაფარვის ამოცანებისათვის (იხ. ცხრილი 3.2). სატესტო ამოცანები კი აქაც აღებულია Or-Library-იდან.

დისერტაციის **მეოთხე თავში:** "დაფარვის ამოცანის ამოხსნა გენეტიკური ალგორითმის გამოყენებით" დასმულია პრობლემა და შემოთავაზებულია პრობლემის გადაწყვეტის გზა.

პირველ პარაგრაფში დასმულია პრობლემა და შემოთავაზებულია გენეტიკური ალგორითმის ზოგადი სქემა, საუბარია ასევე კოდირების სქემის არჩევაზე, სელექციაზე-შესაჯვარებელი ინდივიდების არჩევის წესზე, შეჯვარების სქემაზე და შემოთავაზებულია მუტაციის ცვლადი მნიშვნელობის გამოყენება, რომელიც დამოკიდებული იქნება როგორც თვითონ ინდივიდზე, ასევე ამ ინდივიდის გენოტიპების თავისებურებაზე. ამასთან რაც უფრო ახლოს ვართ ლოკალურ მინიმუმთან ფიქსირებული მუტაციის შემთხვევაში მით უფრო მეტად მიიღება მონათესავე ინდივიდები, ასე შერჩეული მუტაციის ოპერატორი კი უზრუნველყოფს პოპულაციაში არსებული ინდივიდებისაგან განსხვავებული ინდივიდების მიღებას. შესაბამისად იზრდება ოპტიმალური მნიშვნელობის მიღების ალბათობა.

მეოთხე პარაგრაფის პირველ პუნქტში მოყვანილია საწყისი პოპულაციის ფორმირების ალგორითმი (იხ. ალგორითმი 4.1), რომელიც მიიღება დაფარვის ამოცანის შესაბამისი მატრიციდან აღებული მომხმარებლის მიერ განსაზღვრული რაოდენობის სვეტებისაგან, რაოდენობას ჩვენი შეხედულებისამებრ ვარჩევთ ამოცანის განზომილებიდან გამომდინარე. უნდა გავითვალისწინოთ რომ, დიდი განზომილების აღება გაართულებს ამოცანას, პატარა განზომილების არჩევამ შესაძლოა მიახლოებითი მნიშვნელობის სიზუსტის დაკარგვა გამოიწვიოს. მოყვანილია ასევე ალგორითმი 4.2, რომლითაც შეგვიძლია ალგორითმი 4.1-ით მიღებულ ინდივიდებში "ჭარბი" სვეტების წაშლა. ჭარბი სვეტები კი ის სვეტებია, რომელთა არ არსებობის შემთხვევაში ეს ინდივიდი კვლავ წარმოადგენს დამფარავ სიმრავლეს. ჭარბი სვეტების წაშლა მეტად მნიშვნელოვანია, რადგან მათი არსებობა გაზრდის გადარჩევების რაოდენობას და ამოცანის თვლის დროს.

მეორე პუნქტში საუბარია შეჯვარების ოპერატორზე, რომელიც ასეა განსაზღვრული: მშობლების არჩევა ხდება შემდეგნაირად: კენტ იტერაციაზე პოპულაციიდან აირჩევა ის ორი ინდივიდი, რომლებსაც აქვთ სარგებლიანობის ფუნქციის მინიმალური მნიშვნელობები. ლუწ იტერაციაზე კი აირჩევა ერთი ყველაზე უკეთესი - სარგებლიანობის ფუნქციის მინიმალური მნიშვნელობის მქონე ქრომოსომა, მეორე კი ყველაზე უარესი - სარგებლიანობის ფუნქციის მაქსიმალური მნიშვნელობის მქონე. გამოიყენება ერთწერტილიანი შეჯვარება. შეჯვარების ადგილი კი განისაზღვრება შემთხვევითი რიცხვით ინდივიდის

ზომის ფარგლებში. ვინაიდან შეჯვარების შემდეგ შესაძლებელია რომ შვილობილი ქრომოსომა აღარ აღმოჩნდეს დამფარავი, საჭიროა მასში ისეთი სიმრავლეების (სვეტების) ჩამატება, რომლებიც მას დამფარავს გახდიან. ამისათვის ვიყენებთ **ალგორითმ 4.3**-ს.

მესამე პუნქტში საუბარია მუტაციის ოპერატორზე, აქ (4.2) და (4.3) ფორმულებით განსაზღვრულია ენტროპიის ზომები და ალბათობები ყოველი სვეტისათვის. შემდეგ კი მოყვანილია გენის მუტაციის პირობები, რომელშიც გათვალისწინებულია არამარტო სვეტების წონები, არამედ დაფარვის მატრიცის შესაბამის სვეტებში 1-იანების და 0-ების რაოდენობები. ისევე როგორც შეჯვარებისას, მუტაციის შემდეგაც შესაძლოა მოხდეს რომ ის ინდივიდი, რომელმაც განიცადა მუტაცია აღარ აღმოჩნდეს დამფარავი, ამიტომ კვლავ დაგვჭირდება სვეტების ჩამატების **ალგორითმი 4.3**-ის გამოყენება. ვინაიდან, სვეტების ჩამატების შემდეგ მიღებულ ქრომოსომები შესაძლოა აღარ წარმოადგენდნენ ჩიხურ დაფარვებს, ჩამატების შემდეგ გამოვიყენებთ **ალგორითმ 4.2**-ს ახლად გაჩენილი ჭარბი სვეტების წასაშლელად.

მესამე პარაგრაფში მოყვანილია ექსპერიმენტების შედეგები, შემოთავაზებულია სატესტო ამოცანების ამონახსნების ცხრილი (იხ. ცხრილები 4.2.1, 4.2.2) ცნობილი ბიბლიოთეკიდან: Or-Library-დან აღებული 4-6 და A-D სერიის ამოცანებისათვის. მოყვანილია აგრეთვე ექსპერიმენტების შედეგები E-H სერიის ამოცანებისათვის (იხ. ცხრილი 4.3). ჩატარებული ექსპერიმენტების საფუძველზე შეგვიძლია დავასკვნათ რომ შემოთავაზებული ალგორითმი საკმარისად ეფექტურად მუშაობს.

მეოთხე პარაგრაფში დასმულია დაფარვის განზოგადებული ამოცანა, რომელიც სტანდარტული დაფარვის ამოცანისაგან განსხვავდება შეზღუდვებით. ეს ამოცანა ამოხსნილია გენეტიკური ალგორითმის გამოყენებით.

დისერტაციის **მეხუთე თავში:** მოცემულია ნაშრომში მოყვანილი ალგორითმების მუშაობის სისწრაფის შეფასება.

სიახლე და ძირითადი შედეგები

წინამდებარე ნაშრომში შეგვიძლია რამდენიმე კონკრეტული სიახლის გამოყოფა, რომელიც არ გვხვდებოდა სხვა ნაშრომებში:

- ნაშრომში განიხილება მძიმე ბირთვის მეთოდის ახალი უწყვეტი დაგვიანებული მოდელი. საკმაოდ დეტალურად, თეორიული ასპექტების, გეომეტრიული გააზრების და პროგრამული რეალიზაციის თვალსაზრისით, ხდება მძიმე ბირთვის მოდიფიცირებული ალგორითმის შესწავლა, რომელსაც ჩვენ მართვად მძიმე ბირთვს ვუწოდებთ. ტერმინი ”მართვადი” გულისხმობს, რომ ასეთი მძიმე ბირთვი უშვებს დამუხრუჭების, გაჩერების, ან (პირიქით) აჩქარების შესაძლებლობას. ახალი ალგორითმის პროგრამული რეალიზაცია გაცილებით მოქნილია, აქვს პარამეტრების შერჩევის უფრო მდიდარი საშუალებები და ტესტებზე ბევრ შემთხვევაში უკეთეს ამონახსნებს ვღებულობთ, ვიდრე მძიმე ბირთვის სტანდარტული მეთოდით.

- შემოთავაზებულია ოპტიმიზაციის (1) ამოცანის ამოხსნის ორი ახალი გენეტიკური ალგორითმი, რომელებიც დაფუძნებულია კლასიკური გენეტიკური ალგორითმის ძირითად პრინციპებზე. შემოთავაზებულ ალგორითმებში აღწერილია: სარგებლანობის (ფიტნეს) ფუნქციის განსაზღვრის, შეჯვარებისა და მუტაციის რეალიზების განსხვავებული მიდგომები. ალგორითმები სატესტო ამოცანებზე საიმედოდ მუშაობს. რამდენიმე ამოცანისათვის, რომლისთვისაც კლასიკური მეთოდები უშედეგოდ მთავრდება, განხილული ალგორითმებით ვიღებთ პასუხებს.

- უმცირესი დაფარვის და დაყოფის ამოცანების დაფარვის მატრიცის კომპაქტური-შეკუმშული სახით ჩაწერა ბიტური ოპერაციების გამოყენებით. გამოყენებულია ძებნის ხის ალგორითმი. პროგრამირების ტექნიკის არაარსებითი გართულების ხარჯზე შესაძლებელი გახდა გამოყენებული მეხსიერების, ასევე გამოთვლისათვის საჭირო დროის მიახლოებით $32 \cdot M$-ჯერ შემცირება, სადაც M დამფარავი ქვესიმრავლეების რაოდენობაა. ამ მიზნით ჯერ დაყოფის მატრიცა ჩაიწერა კომპაქტურად და შემდეგ მიღებული მატრიცის ელემენტებზე ძირითადი ოპერაციები შესრულდა ლოგიკური ოპერატორებით. ამ ამოცანების ამოსახსნელად დაიწერა პროგრამების კომპლექსი ალგორითმულ ენა C++-ზე, რეალიზებულია Dev-C++ გარემოში. კომპლექსი აპრობირებულია ლიტერატურაში კარგად ცნობილ რეალურ კომბინატორული ტიპის ამოცანებზე, რომლებიც შეგვიძლია ავიღოთ სატესტო ამოცანების ცნობილი ელექტრონული ბიბლიოთეკიდან: Or-Library-დან (Beasley, 1990). დროის მისაღებ ინტერვალებში მიღებულია კარგი შედეგები დაყოფის ამოცანებისათვის და მისაღები შედეგები დაფარვის ამოცანებისათვის.

- შემოთავაზებულია მოცემული სიმრავლის მინიმალური ჯამური წონის არაერთგვაროვანი ღირებულებების მქონე ქვესიმრავლეებით დაფარვის და განზოგადოებული დაფარვის ამოცანების ამოხსნის გენეტიკური ალგორითმები. ალგორითმები დაფუძნებულია კლასიკურ გენეტიკურ ალგორითმზე შეჯვარების და მუტაციის ახალი ოპერატორებით. მიღებულია გამოთვლითი ექსპერიმენტების კარგი შედეგები ცნობილ სატესტო ამოცანებზე.

პრაქტიკული მნიშვნელობა

ნაშრომს გააჩნია პრაქტიკული მნიშვნელობა, ვინაიდან მასში წარმოდგენილია პროგრამული სისტემები, რომლებიც ხსნიან დასმულ ამოცანებს. პროგრამების ნაწილი დაწერილია ალგორითმულ ენა C++-ზე, ნაწილი კი დაწერილია Matlab-ში.

- მძიმე ბირთვის მეთოდის ახალი უწყვეტი დაგვიანებული მოდელისათვის შემოთავაზებული ალგორითმის მოქნილი პროგრამული რეალიზაცია საშუალებას გვაძლევს ამოვხსნათ ოპტიმიზაციის $f(x) \to min,\ x \in R_n$ ამოცანა, ამასთან პარამეტრების შერჩევის უფრო მდიდარი საშუალებებით შევძლებთ მივიღოთ უკეთესი შედეგები მძიმე ბირთვის მეთოდთან შედარებით.

- ოპტიმიზაციის $f(x) \to min,\ x \in R_n$ ამოცანის ამოხსნა გენეტიკური ალგრითმების გამოყენებით შესაძლებლობას გვაძლევს ვიპოვოთ საძიებელი ფუნქციის გლობალური ოპტიმალური მნიშვნელობა, ისე რომ არ მოვითხოვოთ ამ ფუნქციისაგან დამატებითი პირობები, როგორიცაა წარმოებულის არსებობა ან გლუვობა.
- უმცირესი დაყოფისა და დაფარვის ამოცანების ამოხსნის შემოთავაზებული მეთოდი შესაძლოა გამოყენებული იქნეს გრაფთა თეორიის იმ პრაქტიკული ამოცანების ამოსახსნელად, რომლებიც უშუალოდ მიიყვანებიან უმცირესი დაყოფის ან დაფარვის ამოცანაზე, ან რომლებიც წარმოადგენენ მის ქვეამოცანას (მაგალითად, გრაფში წვეროთა დომინირებადი სიმრავლის მოსაძებნად).
- გენეტიკური ალგორითმის გამოყენებით დაფარვისა და მრავალჯერადი დაფარვის ამოცანების ამოხსნით შესაძლებელი ხდება ამ ტიპის დიდი ზომის დაფარვის მატრიცის მქონე პრაქტიკული ამოცანების კარგი მიახლოების მიღება ძალიან სწრაფად.

თავი 1. მძიმე ბირთვის მეთოდის ერთი მოდიფიკაციის შესახებ

1.1. უპირობო მინიმიზაციის ალგორითმების რამდენიმე გეომეტრიული მახასიათებელი

1.1.1. მარტივი შემთხვევა-კვადრატული ფორმის მინიმიზაცია

განვიხილოთ კვადრატული ფორმის მინიმიზაციის ამოცანა:

$$x^T Ax \to \min,\ x \in R^n, \tag{1.1}$$

შემდეგში ყველგან ვიგულისხმოთ, რომ A არის სიმეტრიული დადებითად განსაზღვრული მატრიცა. (1.1)-ის სიმარტივე საშუალებას მოგვცემს, რომ ორზე მეტი განზომილების შემთხვევაშიც შევძლოთ და გეომეტრიულად გავიაზროთ ტერმინის „ზიგზაგიანი ალგორითმები“ (zigzag algorithms) შინაარსი, აგრეთვე იმ უარყოფითი ეფექტების შემსუბუქების გზები, რომლებიც დღესდღეობით არსებობს პრაქტიკაში.

პირველ რიგში, ვაჩვენოთ რომ (1.1) სახეზე დაიყვანება წრფივ განტოლებათა სისტემის ამოხსნის შესაბამისი მინიმიზაციის ამოცანა სიმეტრიული დადებითად განსაზღვრული მატრიცისთვის.

განვიხილოთ $Ax = b$ სისტემა და მისი შესაბამისი მინიმიზაციის ამოცანა:

$$0.5 \cdot x^T Ax - b^T x \to \min \tag{1.2}$$

რადგან A არის სიმეტრიული დადებითად განსაზღვრული მატრიცა, (1.2)-ის მარცხენა მხარის გაწარმოება და მისი ნულთან გატოლება (მინიმუმის აუცილებელი პირობა) გვაძლევს $Ax = b$ -ს.

ამ ამოცანაში არსებობს ერთადერთი მინიმალი, ვთქვათ $\tilde{x}$, რომელიც აკმაყოფილებს $A\tilde{x} = b$ სისტემას. მარტივი გარდაქმნები:

$$\begin{aligned}
&0.5 \cdot x^T Ax - b^T x = 0.5 \cdot x^T Ax - x^T A\tilde{x} = \\
&0.5 \cdot x^T Ax - x^T A\tilde{x} + 0.5 \cdot \tilde{x}^T A\tilde{x} - 0.5 \cdot \tilde{x}^T A\tilde{x} = \\
&0.5 \cdot \left(\tilde{x} - x\right)^T A\left(\tilde{x} - x\right) - 0.5 \cdot \tilde{x}^T A\tilde{x} = \\
&= 0.5 \cdot y^T Ay - 0.5 \cdot \tilde{x}^T A\tilde{x}
\end{aligned}$$

გვიჩვენებს, რომ $y = \tilde{x} - x$ გარდაქმნას (1.2) სახის ამოცანა გადაჰყავს (1.1) სახის ამოცანაში. ცხადია, (1.1)-ში მუდმივი წევრის დამატება ან მუდმივ დადებით კოეფიციენტზე გამრავლება ამოცანის სირთულეს არ ცვლის.

1.1.2. კვადრატული ფორმის ინვარიანტული ქვესივრცეები

რადგან მატრიცებსა და წრფივ ასახვებს შორის ურთიერთცალსახა თანადობაა, ამიტომ ასახვის მიმართ ინვარიანტული ქვესივრცის ანალოგიით შეგვიძლია განვმარტოთ მატრიცის მიმართ ინვარიანტული ქვესივრცე. კონკრეტულად, W ქვესივრცეს ეწოდება A მატრიცის მიმართ ინვარიანტული, თუ

$\forall x \in W$ -დან გამომდინარეობს, რომ $Ax \in W$.

ცხადია, A მატრიცის საკუთრივი ვექტორების ნებისმიერ ქვესიმრავლეზე მოჭიმული წრფივი გარსი არის A მატრიცის მიმართ ინვარიანტული ქვესივრცე.

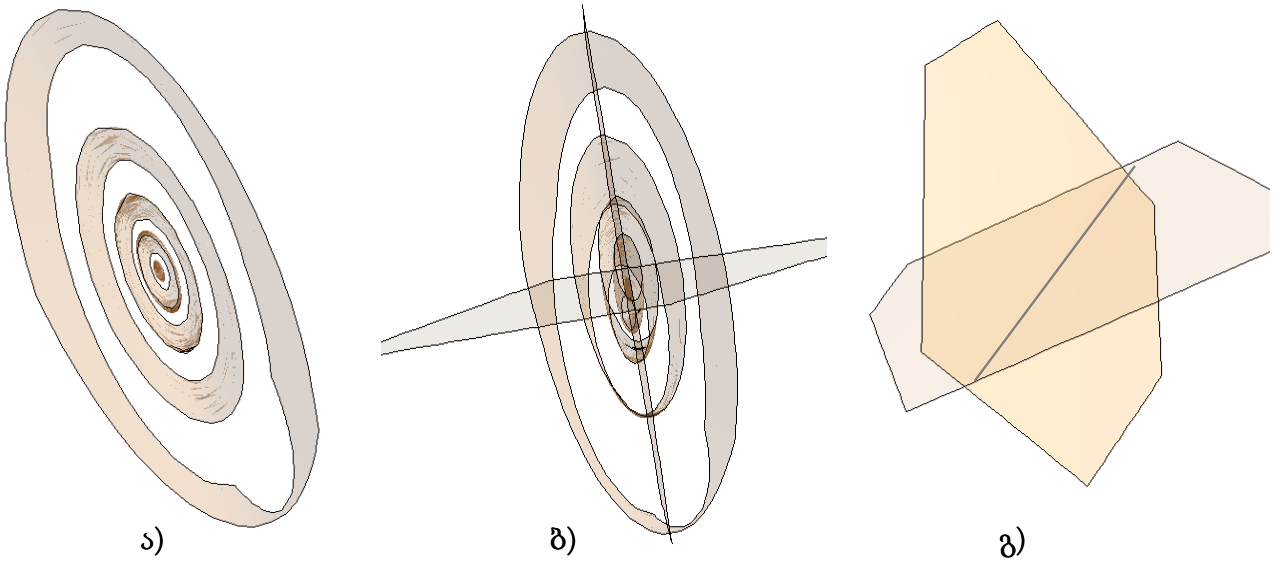
ქვემოთ, პროგრამული პაკეტის „Mathematica" საშუალებით ვაჩვენებთ, რომ ინვარიანტული ქვესივრცეების მიმართ მინიმიზაციის მეთოდის ტრაექტორიების განლაგება ცალსახად ახასიათებს მეთოდის სისწრაფეს და ეფექტურობას. თუმცა, მანამდე 3 განზომილების შემთხვევაში წარმოვიდგინოთ კვადრატული ფორმის დონის ზედაპირები და მისი შესაბამისი კვადრატული მატრიცის მიმართ ინვარიანტული ქვესივრცეები.

განვიხილოთ კვადრატული ფორმა:

$$f(x,y,z) = 0.5\big(x(66x + 79y + 72z) + y(79x + 97y + 80z) + z(72x + 80y + 132z)\big) \quad (1.3)$$

რომლის შესაბამისი მატრიცა $A = \begin{pmatrix} 66 & 79 & 72 \\ 79 & 97 & 80 \\ 72 & 80 & 132 \end{pmatrix}$ სიმეტრიულია და დადებითად განსაზღვრული (საკუთრივი რიცხვებით: $0.6138,\ 38.7795,\ 255.6067$).

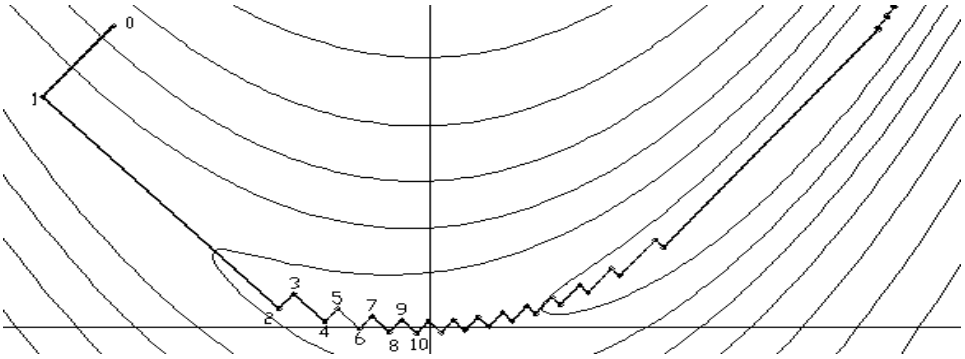
მინიმალის სიახლოვეში, (1.3) კვადრატული ფორმის (მუდმივობის) დონის ზედაპირებს აქვს ნახ. 1.1.1. ა)-ზე მოცემული სახე. თუმცა, ჩვენი გეგმებისთვის უფრო მნიშვნელოვანია ამ კვადრატული ფორმის ზოგიერთი ინვარიანტული ქვესივრცის წარმოდგენა. გარდა ტრივიალური შემთხვევებისა (ცარიელი სიმრავლე და სამგანზომილებიანი სივრცე), ორი ინვარიანტული ქვესივრცე ნაჩვენებია ნახ. 1.1.-ის ბ)-ზე. დასასრულ, იმ მოსაზრებით რომ ზედმეტად არ გადავტვირთოთ ნახაზები, შემდეგში ჩვენ კვადრატული ფორმის ნაცვლად ხშირად წარმოვადგენთ მის ერთ ან რამდენიმე ინვარიანტულ ქვესივრცეს. მაგალითად, როგორც ეს არის ნახ. 1.1.-ის გ)-ზე.



ნახ. 1.1.1. ა) (1.3) კვადრატული ფორმის დონის ზედაპირები; ბ) დონის ზედაპირები ინვარიანტულ ქვესივრცეებთან ერთად; გ) ორი ინვარიანტულ ქვესივრცე.

1.1.3. სხვადასხვა მეთოდის ტრაექტორიების ყოფაქცევა ინვარიანტული ქვესივრცეების მიმართ

სპეციალურ და პოპულარულ ლიტერატურაში ყოველთვის აღინიშნება უსწრაფესი დაშვების მეთოდის „ზიგ-ზაგებიანი“ ყოფაქცევა. მაგალითად, იხ. https://en.wikipedia.org/wiki/Gradient_descent , საიდანაც არის აღებული შემდეგი სურათი, რომელიც ასახავს როზენბროკის ფუნქციისთვის უსწრაფესი დაშვების მეთოდის ტრაექტორიის ფრაგმენტს:

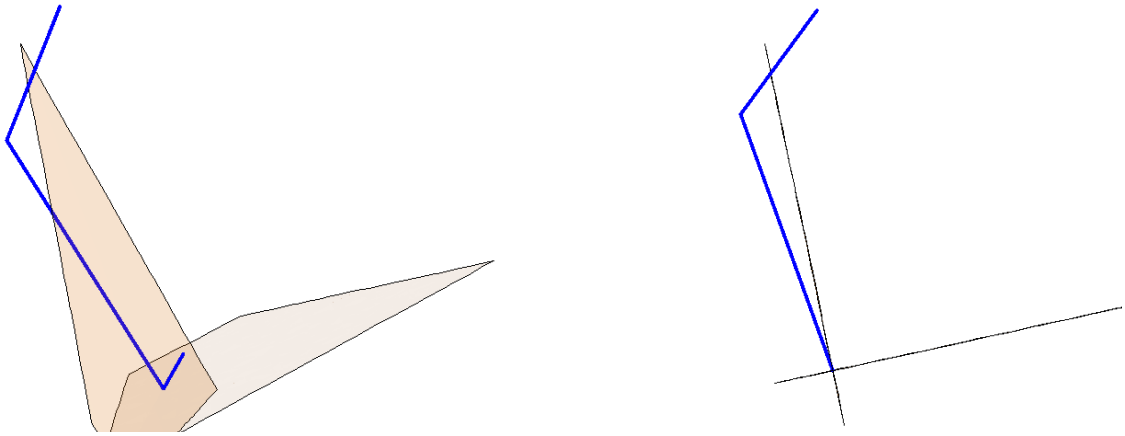


ნახ. 1.1.2. უსწრაფესი დაშვების მეთოდის ტრაექტორიის ფრაგმენტი როზენბროკის ფუნქციისათვის.

ინტუიციურად ადვილი გასაგებია რას ნიშნავს „ზიგ-ზაგები“, თუმცა მიზნის ფუნქციის არაწრფივობის გამო ძნელია ამ ტერმინისთვის მკაცრი შინაარსის მინიჭება. (3) კვადრატული ფორმის მაგალითზე, განვიხილოთ მინიმიზაციის რამდენიმე მეთოდის

ტრაექტორია. ჩვენთვის ცნობილია ამ მეთოდების გამოთვლითი ეფექტურობა, და საინტერესოა მათი სისწრაფისა და ტრაექტორიების გეომეტრიული (ვიზუალური) წარმოდგენის კორელაცია.

შეუღლებულ გრადიენტთა მეთოდი. ეს მეთოდი ზეწრფივი კრებადობით გამოირჩევა, ანუ n-განზომილებიან სივრცეში (დამრგვალების ცდომილებების გარეშე) n იტერაციაში პოულობს მინიმალს. როდესაც n = 3, შეგვიძლია გრაფიკულად წარმოვიდგინოთ მისი სისწრაფის მიზეზი. როგორც ვხედავთ, მისი ტრაექტორიები პრაქტიკულად არ აკეთებენ „ზიგზაგს“.

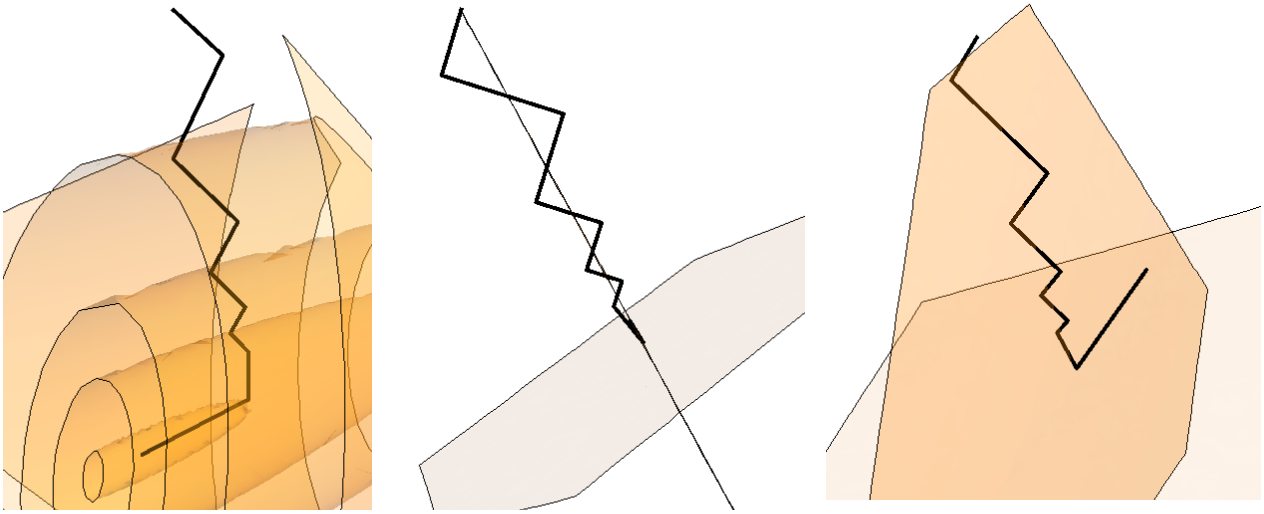


ა) ბ)

ნახ. 1.1.3. ა)-ბ) შეუღლებულ გრადიენტთა მეთოდის ტრაექტორიები სხვადასხვა ხედით ინვარიანტულ სივრცეებთან ერთად;

შემდეგში, მძიმე ბირთვის შემთხვევაში ჩვენ შევეცდებით ისეთი მოდიფიკაცია შევქმნათ, რომლის ტრაექტორიებსაც ექნებათ შეუღლებული გრადიენტების მეთოდის ტრაექტორიებთან მიმსგავსებული ყოფაქცევა, რაც ირიბი არგუმენტი იქნება მოდიფიცირებული მეთოდის კრებადობის სიჩქარის მატების სასარგებლოდ.

უსწრაფესი დაშვების მეთოდი. განვიხილოთ ნახ. 1.1.4-ზე მოყვანილი სურათები. ნახ. 1.1.4. ა)-ზე ნაჩვენებია უსწრაფესი დაშვების მეთოდის ტრაექტორია დონის ზედაპირებსა და ინვარიანტულ ქვესივრცეებთან ერთად. ნახ. 1.1.4. ბ)-ზე ნაჩვენებია ტრაექტორიის ზედა ნაწილი “გვერდხედში“ ერთ-ერთი ინვარიანტული ქვესივრცის მიმართ. ამ შემთხვევაში კარგად ჩანს, რომ ტრაექტორია ირყევა ინვარიანტული ქვესივრცის მიმართ და მონაცვლეობით გადადის მის ორივე მხარეს. ნახ. 1.1.4. გ)-ზე ნაჩვენებია სხვა დახრის კუთხით. ტრაექტორიის ქვედა ნაწილი ამჯერად ჰორიზონტალურად მოთავსებული ინვარიანტული ქვესივრცის მიმართ ირხევა, თუმცა ეს რხევა ახლა უკვე ძალიან არის შემცირებული.



ა) ბ) გ)

ნახ. 1.1.4. ა) უსწრაფესი დაშვების მეთოდის ტრაექტორია დონის ზედაპირებსა და ინვარიანტულ ქვესივრცეებთან ერთად; ბ)-გ) უსწრაფესი დაშვების მეთოდის ტრაექტორია დონის ზედაპირებთან ერთად სხვადასხვა ხედით ნაჩვენები;

უსწრაფესი დაშვება ინვარიანტულ ქვესივრცეზე შეჩერებით. განვიხილოთ ასეთი ჰიპოტეტური მეთოდის აგების შესაძლებლობა: ყოველ ბიჯზე, ვეშვებით ანტიგრადიენტის გასწვრივ მანამდე, ვიდრე არ აღმოვჩნდებით უფრო დაბალი განზომილების ინვარიანტულ ქვესივრცეში. გასაგებია, რომ ბიჯების რაოდენობა ვერ იქნება სივრცის განზომილებაზე მეტი, რადგან ყოველ ბიჯზე სულ უფრო დაბალი განზომილების ინვარიანტულ ქვესივრცეში შევდივართ, საიდანაც გამოსვლა ტრაექტორიას არ შეუძლია ინვარიანტული ქვესივრცის განმარტების ძალით.

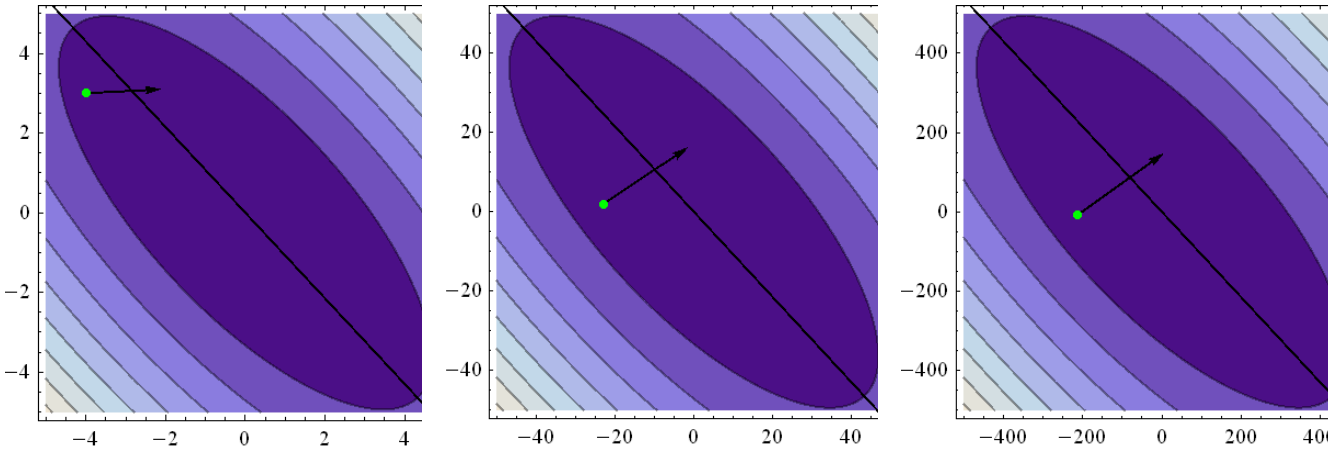
თეორიულად, მართლაც შესაძლებელია დავადგინოთ, თუ რა პირობას უნდა აკმაყოფილებდეს წერტილი ანტიგრადიენტის გასწვრივ, რომელიც უფრო დაბალი განზომილების ინვარიანტულ ქვესივრცეშია, ვიდრე ის წინა იტერაციის დროს იმყოფებოდა. რადგან მეთოდის პრაქტიკულ რეალიზაციას დიდ დაბრკოლებას უქმნის დამრგვალებასთან დაკავშირებული პრობლემები, ამიტომ საკითხი განვიხილოთ არამკაცრად, კონკრეტულ მაგალითზე.

განვიხილოთ (1.1) ამოცანა, სადაც

$$A = \begin{pmatrix} 10 & 7 \\ 7 & 9 \end{pmatrix}$$

ავიღოთ $x_0 = (-4,3)$. ნახ. 1.1.5. ა)-ზე ნაჩვენებია ანტიგრადიენტის მიმართულება კვადრატული ფორმის გრაფიკისა და მისი ერთ-ერთი ინვარიანტული ქვესივრცის ფონზე. ნახ. 1.1.5. ბ)-ზე ნაჩვენებია იგივე ფორმის გრაფიკი ინვარიანტულ ქვესივრცესთან ერთად. ამჯერად, მონიშნული წერტილი მოთავსებულია x_0-იდან მისი გრადიენტის მიმართულე-

ბით. ახალი წერტილიდან ჩვეულებრივად ითვლება კვადრატული ფორმის ანტიგრადიენტის მიმართულება და იგი წარმოდგენილია ისრით. დასასრულ, ნახ. 1.1.5. გ)-ზე მონიშნული წერტილი კვლავ აღებულია x_0-იდან მისი გრადიენტის მიმართულებით, ოღონდ უფრო შორს. როგორც ვხედავთ, ამჯერად მისი ანტიგრადიენტი თითქმის მართობულია ინვარიანტული ქვესივრცის.



ა) ბ) გ)

ნახ. 1.1.5. ა) მონიშნული წერტილიდან ნაჩვენებია ანტიგრადიენტის მიმართულება კვადრატული ფორმის გრაფიკისა და მისი ერთ-ერთი ინვარიანტული ქვესივრცის ფონზე; ბ) იგივეა რაც ა)-ზე, მხოლოდ ნაჩვენებია გრადიენტის მიმართულება; გ) იგივეა რაც ა)-ზე, მხოლოდ წერტილი აღებულია მოშორებით.

უსწრაფესი დაშვების მეთოდში, ამონახსნის x_i მიახლოებიდან მომდევნოზე გადასვლის პირობა არის $x_0 + t \cdot r$ სხივზე (r ანტიგრადიენტს აღნიშნავს ამ იტერაციისთვის) ისეთი წერტილის მოძებნა, რომელშიც ახალი ანტიგრადიენტი არის r-ის მართობული. ამისგან განსხვავებით, ინვარიანტულ ქვესივრცეზე დაშვება გარანტირებული იქნება, თუ აღნიშნულ სხივზე შევარჩევთ ისეთ წერტილს, რომლიდანაც ახალი ანტიგრადიენტი არის Ar-ის მართობული (რადგან სწორედ Ar არის იმ ზღვრული ვექტორის კოლინეარული, რომელიც მიიღება ნახაზებზე ნაჩვენები პროცესის გაგრძელებით).

როგორც აღვნიშნეთ, ეს მეთოდი ზედმეტად მგრძნობიარეა დამრგვალების ცდომილებების მიმართ და ერთი და იმავე ინვარიანტულ ქვესივრცეზე შესაძლოა რამდენჯერმე სცადოს დაშვება. თუმცა ჩვენს ექსპერიმენტებში საუკეთესო შედეგი შეუღლებული გრადიენტების მეთოდს მაინც ერთი რიგით ჩამორჩება. ცნობისთვის, შემთხვევითი სიმეტრიული დადებითად განსაზღვული მატრიცების შემთხვევაში, უსწრაფესი დაშვების მეთოდი დაახლოებით 10000-ჯერ ნელია შეუღლებული გრადიენტების მეთოდზე (ცვლადების რიცხვი როდესაც 10-ეულებით იზომება). მეთოდის კოდი მოყვანილია დანართში.

გრადიენტული დაშვების მეთოდი. მისი სიხისტე. ძალიან მოკლედ თუ აღვწერთ,

$$F(x) \to \min, \; x \in R^n, \qquad (1.4)$$

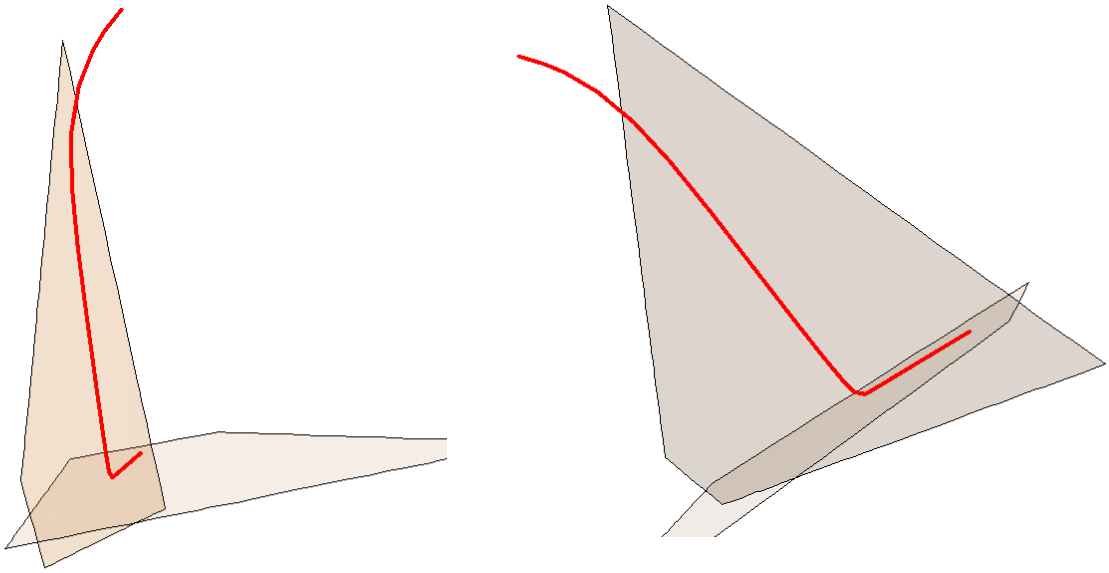
უპირობო მინიმიზაციის ამოცანისთვის გრადიენტული დაშვება ნიშნავს

$$\dot{x} = -F'(x), \qquad (1.5)$$

დიფერენციალური განტოლების ამოხსნას მოცემული საწყისი პირობით. ინვარიანტული ქვესივრცის განმარტებიდან და (1.5) განტოლების სახიდან პირდაპირ გამომდინარეობს, რომ თუ საწყისი მნიშვნელობა ეკუთვნის რომელიმე ინვარიანტულ ქვესივრცეს, იგივე ქვესივრცეში დარჩება მთელი ტრაექტორია. ამიტომ ამ მეთოდს არ გააჩნია არანაირი ზიგ-ზაგები, იგი ვერ დატოვებს ან/და გადაკვეთს ინვარიანტულ ქვესივრცეებს.

სამწუხაროდ, ეს ფაქტი არ ნიშნავს რომ მეთოდი სწრაფად იკრიბება მინიმალისკენ. მიზეზი ადვილად ასახსნელია გეომეტრიულად (და შემოწმებადი პროგრამულად).

განვიხილოთ იგივე კვადრატული ფორმის მინიმიზაციის საკითხი გრადიენტული მეთოდის გამოყენებით.

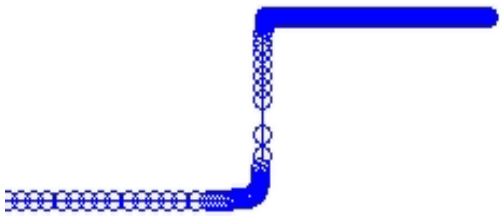


ა) ბ)

ნახ.1.1.6. ა)-ბ) გრადიენტული დაშვების მეთოდის ტრაექტორიის სახე ორი განსხვავებული ხედით.

ნახ. 1.1.6. გვიჩვენებს ტრაექტორიის სახეს ორი განსხვავებული ხედით. როგორც ვხედავთ, ტრაექტორია მიიზიდება ჯერ „უახლოესი“ ინვარიანტული ქვესივრცისკენ, შემდეგ, მის გაყოლებაზე, უფრო დაბალი განზომილების ინვარიანტული ქვესივრცისკენ და ა.შ., როგორც ზემოთ იყო ნახსენები გამომდინარე დიფერენციალური განტოლების სახიდან.

თუმცა ამ სურათების შემდეგ მეთოდის სისწრაფესთან დაკავშირებული პრობლემები გასაგები ხდება: (1.5) დიფერენციალური განტოლება, თავისი სიმარტივის მიუხედავად არის ხისტი. არამკაცრ დონეზე, ამაში გვარწმუნებს ცხადი მსგავსება ნახ.1.1.7.-სა და ნახ.1.1.6.-ის სურათებს შორის. თვითონ ნახ.1.1.6. აღებულია ხისტი სისტემებისადმი მიძღვნილი სტატიიდან:
http://www.scholarpedia.org/article/Stiff_systems



ნახ.1.1.7. ხისტი სისტემის ამოხსნის ნიმუში

(1.5) სისტემის სიხისტის დამტკიცება ჩვენ შეგვიძლია ირიბი მეთოდით, პროგრამული პაკეტი MATLAB-ის ფუნქციების გამოყენებით: მეთოდები, რომლებიც ზოგადია და იყენებენ რუნგა-კუტას მეთოდს (თუნდაც მაღალი რიგის), ბევრად ნელა მუშაობენ ხისტ სიტემებთან სამუშაოდ განკუთვნილ სპეციალურ მეთოდებთან შედარებით.

ზოგადად, MATLAB იყენებს ode23, ode45, ode113 ფუნქციებს არახისტ სისტემებთან სამუშაოდ. ამასთან ode23 ფუნქციაში რეალიზებულია რუნგე-კუტას 2/3 რიგის ალგორითმი ბოგარცკი-შამინის მოდიფიკაციით, ode45 ფუნქციაში რეალიზებულია რუნგე-კუტას 4/5 რიგის ალგორითმი დორმანდ-პრინცის მოდიფიკაციით, ხოლო ode113 ფუნქცია წარმოად-გენს ადამსის მეთოდს ბაშფოტ-მოულტონის მოდიფიკაციით. MATLAB-ის წიგნების ავტორების უმეტესობა რეკომენდაციას უწევს თავდაპირველად ode45 ფუნქციის გამოყენე-ბას, რადგან რუნგე-კუტას მეთოდებს შორის ის ყველაზე მეტი სიზუსტით ხასიათდება, გარდა ამისა ის მართლაც კარგად ხსნის "კარგ" და "პირობითად კარგ" სისტემებს.

გამოთვლებისათვის საჭირო დანახარჯებზე შეგვიძლია ვიმსჯელოთ ექსპერიმენტების ჩატარების შემდეგ, კერძოდ მოცემული სიზუსტით ამოხსნისას შეგვიძლია დავითვალოთ თითოეული ფუნქციის მუშაობის დრო და აგრეთვე რამდენჯერ მოხდა მარჯვენა მხარეზე მიმართვა და რამდენჯერ მოხდა ფუნქციის გამოთვლა.

MATLAB-ის ამ ფუნქციების გამოყენებით, (1.5) სისტემისათვის ექსპერიმენტების ჩატარება დაკავშირებულია შემდეგ სირთულესთან: ფუნქციაში უნდა მივუთითოთ დროის ინტერვალი, ხოლო მინიმიზაციის ამოცანაში წინასწარ შეუძლებელია იმის გამოცნობა, თუ რა დროის შემდეგ მიაღწევს ტრაექტორიის ბოლო საჭირო სიზუსტეს. ამიტომ, ერთი და იგივე ინტერვალის აღება გვიწევს რამდენჯერმე, ვიდრე ბოლო წერტილში გრადიენტი საკმარისად არ შემცირდება.

ინტერვალებად ავიღეთ: [0,50], [0,100]და [0,1000]. სიზუსტე ყველა შემთხვევაში იყო 0.01. ამ სიზუსტეში იგულისხმება რომ მიღებულ ტრაექტორიის საბოლოო წერტილში ფუნქციის გრადიენტის ნორმა უნდა გახდეს ნაკლები 0.01-ზე.

ცხრილი 1.1.1. გვიჩვენებს, თუ რა დროში ამოხსნა სისტემა არახისტი (კარგი) სისტემე-ბისთვის გამიზნულმა ამომხსნელებმა და რამდენჯერ მოხდა მარჯვენა მხარის გამოთვლა.

ცხრილი 1.1.1. ექსპერიმენტის შედეგები (1.5) სისტემისათვის MATLAB-ის არახისტი სისტემებისათვის განკუთვნილი ამომხსნელების გამოყენებით.

	ინტეგრების ინტერვალი t∈[0,50]		ინტეგრების ინტერვალი t∈[0,100]		ინტეგრების ინტერვალი t∈[0,1000]	
ფუნქცია	თვლის დრო	მარჯვენა მხარეზე მიმართვების რაოდენობა	თვლის დრო	მარჯვენა მხარეზე მიმართვების რაოდენობა	თვლის დრო	მარჯვენა მხარეზე მიმართვების რაოდენობა
ode45	100.872429	1543780	203.9727	3084124	3004.777937	30809780
ode113	555.30317	3322650	1207.8544	6634619	-	-
ode23	17.358368	132637	34.45375	264876	520.117379	2645221

ჩვეულებრივი დიფერენციალური განტოლების ხისტი სისტემებისათვის განკუთვნილი ამომხსნელებია: ode15s, ode23s, ode23t, ode23tb. დავახასიათოთ ისინი ძალიან მოკლედ: ode15s ფუნქცია იყენებს სასრულსხვაობიან მეთოდს შებრუნებული დიფერენცირების სქემით, ode23s - როზენბროკის მე-2 რიგის მოდიფიცირებული მეთოდია, ode23t – ტრაპეციის მეთოდია "თავისუფალი" ინტერპოლანტის გამოყენებით და რეკომენდებულია მკაცრად ხისტი სისტემების ამოხსნისას. ode23tb - რუნგე-კუტას კომბინირებული მეთოდია ტრაპეციის მეთოდთან და შებრუნებულ დიფერენცირების სქემასთან. ამ ფუნქციებისათვის იგივე პირობებში, რაც წინა ფუნქციებით ექსპერიმენტირებისას გვქონდა ჩატარებული გამოთვლების შედეგები მოცემულია ქვემოთ ცხრილში:

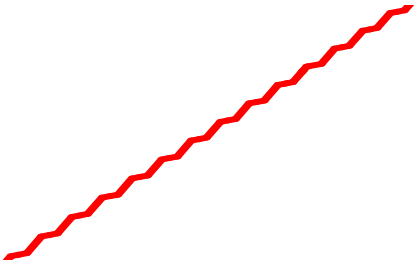
ცხრილი 1.1.2. ექსპერიმენტის შედეგები (1.5) სისტემისათვის MATLAB-ის ხისტი სისტემებისათვის განკუთვნილი ამომხსნელების გამოყენებით.

	ინტეგრების ინტერვალი t∈[0,50]		ინტეგრების ინტერვალი t∈[0,100]		ინტეგრების ინტერვალი t∈[0,1000]	
ფუნქცია	თვლის დრო	მარჯვენა მხარეზე მიმართვების რაოდენობა	თვლის დრო	მარჯვენა მხარეზე მიმართვების რაოდენობა	თვლის დრო	მარჯვენა მხარეზე მიმართვების რაოდენობა
ode15s	4.123009	13470	3.906377	14011	4.654299	15023
ode23s	1.723359	1823	1.082467	1884	1.146167	1965
ode23t	3.795137	11544	3.89989	11829	3.964995	12279
ode23tb	1.096987	2429	0.830762	2494	1.136194	2594

ცხრილი 1.1.2.-ისა და ცხრილი 1.1.2.-ის შედეგები მკვეთრად განსხვავდებიან ერთმანეთისაგან. როგორც ვხედავთ, ხისტი სისტემის ამომხსნელები რამდენიმე ათეულჯერ

სწრაფად ხსნიან (1.5) სისტემას. რამდენადაც სიხისტის ცნება მკაცრად არც არის განსაზღვრული, შეგვიძლია ამ ფაქტის საფუძველზე დავასკვნათ, რომ (1.5) არის ხისტი სისტემა.

ზიგ-ზაგიანი მეთოდები - უსწრაფესი დაშვებიდან გრადიენტულამდე. ბუნებრივად ჩნდება კითხვა: რა მოხდება, თუ მინიმიზაციის მეთოდი ყოველ იტერაციაზე ანტიგრადიენტის გასწვრივ მოძებნის ისეთ წერტილს, რომელშიც ახალი ანტიგრადიენტი ძველ მიმართულებასთან შეადგენს არა $90°$, არამედ უფრო გაშლილ, ვთქვათ $135°$ კუთხეს. ვუწოდოთ ამ მიდგომას ინვარიანტული კუთხის მქონე გრადიენტული უსწრაფესი დაშვება. თვითონ უსწრაფესი დაშვებისთვის ინვარიანტული კუთხე არის $90°$, გრადიენტული დაშვებისთვის - $180°$ ანუ გაშლილი კუთხე.



ნახ.1.1.8. ზიგზაგიანი მეთოდის ყოფაქცევა ინვარიანტული ქვესივრცის სიახლოვეს

სამწუხაროდ, ამ მიდგომით მხოლოდ პირველი რამდენიმე იტერაცია აღმოჩნდება გაცილებით ახლოს მინიმალთან. შემდეგ, ინვარიანტული ქვესივრცეების სიახლოვეს ვღებულობთ უფრო ხშირ ზიგ-ზაგებსაც კი, ხოლო საბოლოოდ კრებადობა მხოლოდ უმნიშვნელოდ იზრდება. მაგალითად, ნახ.1.1.8.-ზე მოცემულია იგივე ამოცანაში (რაც ზემოთ იყო) ტრაექტორიის ყოფაქცევა ერთ-ერთი ინვარიანტული ქვესივრცის სიახლოვეს.

მძიმე ბირთვის მეთოდი. ამ მეთოდის და მისი ერთი მოდიფიკაციის შესახებ დეტალურად ვისაუბრებთ შემდეგ პარაგრაფში. ამჯერად, შევეხოთ მხოლოდ ასეთ საკითხს - რის ხარჯზე არის მძიმე ბირთვი ბევრად უფრო სწრაფი, ვიდრე უსწრაფესი დაშვების, ან თუნდაც გრადიენტული დაშვების მეთოდი?

ისევე როგორც გრადიენტული დაშვება, მძიმე ბირთვის ალგორითმი შეესაბამება უწყვეტ დინამიკურ სისტემას, რომელიც შეიძლება აღიწეროს ან მეორე რიგის დიფერენციალური განტოლებით:

$$\ddot{x} = a\dot{x} + b\nabla f(x) = 0 \qquad (1.6)$$

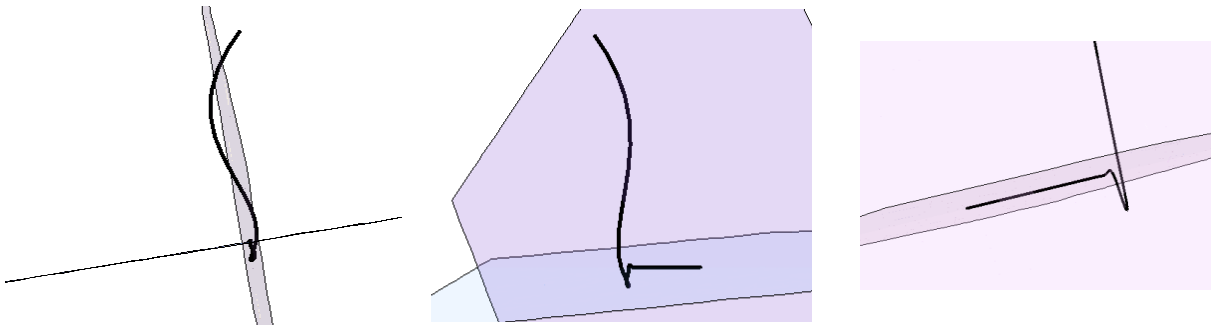
სადაც $\nabla f(x)$-აღნიშნავს $f(x)$ ფუნქციის მეორე რიგის წარმოებულების მატრიცას.

ან როგორც ნეიტრალური ტიპის დაგვიანებული სისტემით:

$$\begin{cases} \dot{x}(t) = 0, \quad t \le 0; \quad x(0) = x_0; \\ \dot{x}(t) = \alpha\dot{x}(t-\tau) - \beta f'(x(t)), \quad t > 0. \end{cases} \qquad (1.7)$$

როგორც ზემოთ ვნახეთ, გრადიენტული დაშვების უწყვეტი მოდელი არის ხისტი სისტემა და ამით აიხსნება მისი დაბალი სიჩქარე. მძიმე ბირთვის მეთოდის სწრაფი

კრებადობის მიზეზზე წარმოდგენის შესაქმნელად განვიხილოთ იგივე ამოცანაზე (რაც წინა მეთოდებისთვის იყო განხილული) მისი ტრაექტორია რამდენიმე განსხვავებული ხედით:



ა) გ) გ)

ნახ.1.1.9. ა)-ბ)-გ) მძიმე ბირთვის ტრაექტორის სხვადასხვა ხედი.

როგორც ვხედავთ, მძიმე ბირთვი, თუ შეიძლება ასე ითქვას, სიხისტეს ამცირებს იმის ხარჯზე, რომ მკვეთრად მოხვევის მაგივრად აგრძელებს მოძრაობას იგივე მიმართულებით, შემდეგ აკეთებს „ნახევარ-მარყუჟებს“, ან რკალს, და ასე უბრუნდება საჭირო მიმართულებას.

მძიმე ბირთვის მოდელში რაც უფრო მეტ წონას მივცემთ ხახუნს და ამით შევამცირებთ ინერციის გავლენას, ტრაექტორია მით უფრო დაემსგავსება გრადიენტული დაშვების მეთოდის ტრაექტორიას. თუმცა, სისწრაფის თვლასაზრისით უფრო მომგებიანი იქნება, თუ მოვიფიქრებთ მის ისეთ მოდიფიკაციას, რომ ტრაექტორია მაქსიმალურად დაემსგავსოს შეუღლებული გრადიენტების მეთოდის ტრაექტორიას. შემდეგ პარაგრაფში სწორედ ასეთი მცდელობა გვექნება და როგორც ექსპერიმენტების შედეგები გვიჩვენებს, საკმაოდ წარმატებულიც არაწრფივ შემთხვევაში.

1.2. მძიმე ბირთვის მოდიფიცირებული მეთოდი

1.2.1. მძიმე ბირთვის მეთოდი ზოგადად

მძიმე ბირთვის მეთოდის სტანდარტული ალგორითმი განისაზღვრება რეკურენტული თანაფარდობით:

$$\begin{cases} x_1 = x_0 - \beta_0 f'(x_0), \\ x_{i+1} = x_i + \alpha_i (x_i - x_{i-1}) - \beta_i f'(x_i), \quad i > 0; \end{cases} \tag{1.8}$$

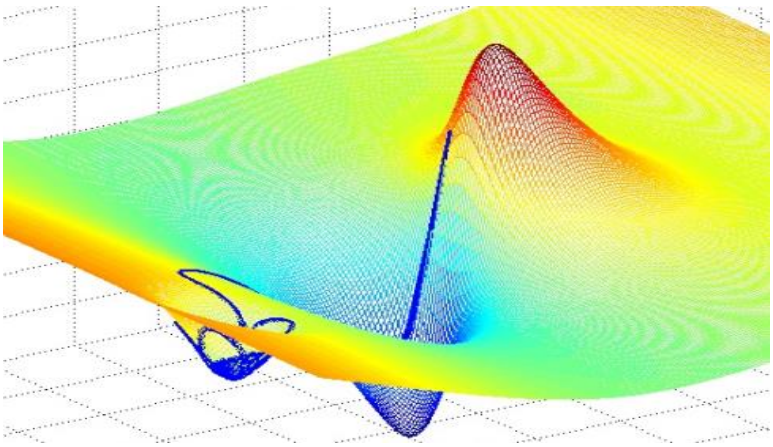
სადაც $f: R_n \longrightarrow R$ არის მიზნის ფუნქცია, x_0 არის ამონახსნის საწყისი მიახლოება, $0 \le \alpha_i \le 1$ და $\beta_i > 0$, $i > 0$. უკიდურეს შემთხვევებში, როდესაც $\alpha_i = 0$ და $\alpha_i = 1$, ჩვენ

ვღებულობთ სტანდარტულ ფიქსირებულ-ბიჯიან გრადიენტულ მეთოდს, და მძიმე ბირთვის უსასრულო მოძრაობას ხახუნის გარეშე. გავრცელებული მოსაზრებით, მძიმე ბირთვის მეთოდი, კარგად შერჩეული $0 \le \alpha_i \le 1$ და $\beta_i > 0$ კოეფიციენტების შემთხვევაში, ერთ-ერთი საუკეთესოა ოპტიმიზაციის რიცხვით მეთოდებს შორის. ეს მიიღწევა იმის ხარჯზე, რომ მინიმუმის სიახლოვეს კრებადობის სიჩქარე არ იკლებს. თუმცა, კოეფიციენტების შერჩევა რთულია და არსებითია კონკრეტული ამოცანებისთვის ჩატარებული ექსპერიმენტების შედეგად დაგროვილი გამოცდილების გათვალისწინება.

(1.8) სქემის უწყვეტ მოდელს, წარმოადგენს მეორე რიგის დიფერენციალური სისტემა (იხ. (Soo Y. Chang, et al., 1989; Goudou, et al., 2009; Cabot A., 2009)). მაგრამ ეს მოდელი გარკვეულ შემთხვევებში (იხ. ქვემოთ) ნაკლებად ინფორმატიულია, ამიტომ ჩვენ ვარჩევთ განსხვავებული მოდელით სარგებლობას.

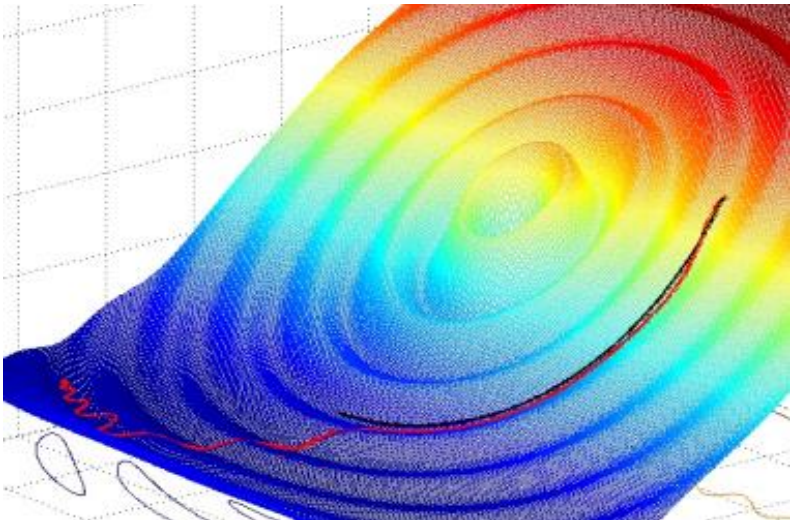
თუ ჩავთვლით, რომ (1.7)-ში ბიჯი საკმაოდ მცირეა, მაშინ (1.8) სქემის უწყვეტ მოდელად შეგვიძლია მივიღოთ ნეიტრალური ტიპის დაგვიანებული (1.7) სისტემა.

სიმარტივისთვის, აქ და ქვემოთ ვგულისხმობთ რომ α და β მუდმივებია. τ-არის დაგვიანება.



ნახ. 1.2.1. მძიმე ბირთვი გასცდა გლობალურ მინიმალს

ასეთი ალტერნატივა შესაძლოა საკამათოა, მაგრამ ინერციის ფაქტორის რეალიზება დაგვიანების საშუალებით ხშირ შემთხვევებში ძალიან მოსახერხებელია, რადგან გვიჩვენებს მძიმე ბირთვის მეთოდის ერთი ცნობილი დეფექტის გამოსწორების შესაძლებლობას. კერძოდ, მძიმე ბირთვის მეთოდის გამოყენება გლობალური მინიმალის განსაზღვრისთვის ზოგჯერ არაადეკვატურ შედეგს იძლევა. ერთი ასეთი შემთხვევა ნაჩვენებია ნახ. 1.2.1.-ზე, სადაც ტრაექტორია სრულდება არა გლობალურ, არამედ "ცუდ" ლოკალურ მინიმალში. მძიმე ბირთვის ტრაექტორია გაივლის ერთ-ერთი მინიმალის სიახლოვეს და შემდეგ მიისწრაფვის შედარებთ "უარესი" ლოკალური მინიმალისკენ.



ნახ.1.2.2. ___ სტანდარტული გრადიენტული მეთოდისა და —მძიმე ბირთვის ერთ-ერთი ილუსტრაცია

სამწუხაროდ, ასეთი სიტუაცია ტიპიურია ტალღური ზედაპირის მქონე ფუნქციათა საკმაოდ ვრცელი კლასისთვის. ნახ.1.2.2.-ზე ჩვენ ვხედავთ, რომ სტანდარტული გრადიენტული მეთოდი ფიქსირებული ბიჯით იკრიბება უკეთესი ლოკალური მინიმალისკენ, ვიდრე მძიმე ბირთვი.

(1.7) მოდელი ადვილად შეგვიძლია გავხადოთ უფრო "ჭკვიანი":

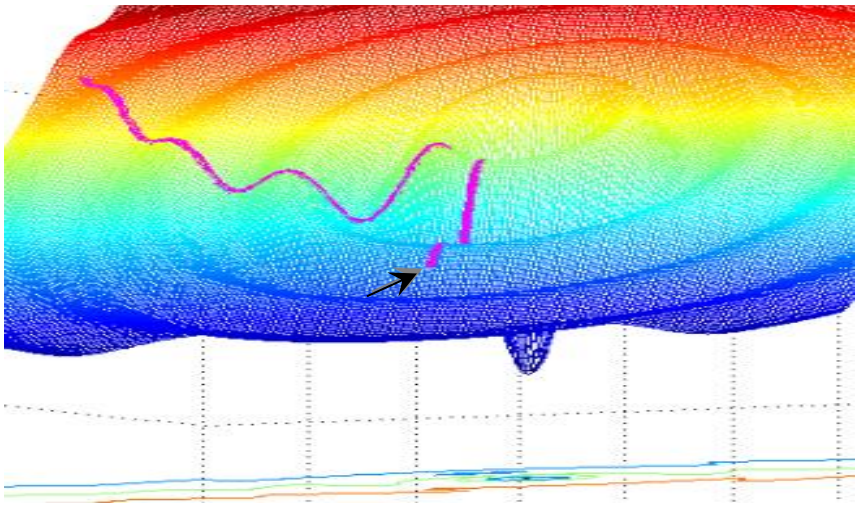
$$\begin{cases} \dot{x}(t) = 0, \quad t \le 0; \quad x(0) = x_0, \\ \dot{x}(t) = \alpha \dot{x}(t-\tau) - \beta f'(x(t)), \quad 0 < t \le \tau, \\ \dot{x}(t) = \alpha \dot{x}(t-\tau) - \beta \cdot \operatorname{sign}\left(\left(\dot{x}(t-\tau) \right) \cdot \left(f'(x(t)) \right)^T \right) f'(x(t)), \quad t > \tau. \end{cases} \tag{1.9}$$

(1.9)-ს მესამე სტრიქონში, მეორე შესაკრებში, თანამამრავლი წარმოადგენს ტრაექტორიის დაგვიანებული წარმოებულისა და ტრაექტორიის მიმდინარე წერტილში ველის სკალარული ნამრავლის ნიშანს. შედეგად, ტრაექტორიის გასწვრივ პირველივე ლოკალური მინიმალის გავლის შემდეგ (1.9) სისტემა გადაერთვება ლოკალური მაქსიმალის ძებნის რეჟიმზე, ლოკალური მაქსიმალის გავლის შემდეგ (1.9) სისტემა გადაერთვება ლოკალური მინიმალის ძებნის რეჟიმზე, და ა. შ. მონაცვლეობით.

განსაკუთრებით საინტერესოა ის ფაქტი, რომ ძებნა უნდა განხორციელდეს როგორც პირველივე ლოკალური მინიმალის, ასევე პირველივე ლოკალური მაქსიმალის მიმართულებითაც, ანუ შემდეგი სისტემის საშუალებით:

$$\begin{cases} \dot{x}(t) = 0, \quad t \le 0; \quad x(0) = x_0, \\ \dot{x}(t) = \alpha \dot{x}(t-\tau) + \beta f'\big(x(t)\big), \quad 0 < t \le \tau, \\ \dot{x}(t) = \alpha \dot{x}(t-\tau) + \beta \cdot \operatorname{sign}\Big(\big(\dot{x}(t-\tau)\big) \cdot \big(f'\big(x(t)\big)\big)^T\Big) f'\big(x(t)\big), \quad t > \tau. \end{cases} \tag{1.10}$$

რადგან წინასწარ გაურკვეველია, თუ რომელ მხარეს იქნება კარგი ლოკალური მინიმალი. მაგალითად, ასე ვიქცევით ნიმუშად განხილული ტალღური ფუნქციის მაგალითზე, როდესაც (1.10) სისტემა იძლევა ტრაექტორიას (იხ. ნახ. 1.2.3, ტრაექტორიის დასაწყისი ნახაზის ქვედა ნაწილში მონიშნულია ისრით), რომელიც მაქსიმალურად ახლოს გაივლის გლობალურ მინიმალთან. განსხვავებით ზემოთ მოყვანილი შემთხვევებისგან, ამ საწყისი წერტილიდან გლობალური მინიმალის სიახლოვეს შეუძლებელია მოვხვდეთ (ყოველ შემთხვევაში ჩვენთვის ცნობილი) სხვა რიცხვითი მეთოდებით.



ნახ. 1.2.3. (1.10) სისტემით მიღებული შედეგის ერთ-ერთი ილუსტრაცია

სტანდარტულ პირობებში, (1.7), (1.9)/(1.10) სისტემების ამონახსნის არსებობის საკითხის გარკვევა არ წარმოადგენს სირთულეს. თუმცა, წარმოდგენილი ნაშრომის მიზანს არ წარმოადგენს მძიმე ბირთვის მეთოდის განვითარება გლობალური და კარგი ლოკალური მინიმალის ძებნის მიმართულებით, რადგან ეს ძალიან ვრცელი და რთული საკითხი დამოუკიდებელი კვლევის თემაა. მაგრამ მსჯელობის სისრულისთვის უნდა შევნიშნოთ, რომ მძიმე ბირთვის სტანდარტული ალგორითმისგან განსხვავებით, ახლა ბუნებრივი დამუხრუჭება აღარ მოხდება. ამიტომ, თუ ძებნის გაგრძელებას ვაპირებთ დროის საკმაოდ ვრცელ შუალედზე, შესაძლოა გონივრული იყოს ინერციის მდგენელის დათრგუნვა სიჩქარის პროპორციულად, ანუ:

$$\alpha(t) = \exp\Big(-M \cdot \big\| f'\big(x(t)\big) \big\|^2\Big)$$

სადაც M არის დადებითი მუდმივი. ასეთი ცვლილება საშუალებას მოგვცემს, რომ ნახ. 1.2.3-ის მსგავსი ტრაექტორია მივიღოთ საწყისი წერტილების გაცილებით ვრცელი სიმრავლისთვის, ვიდრე ეს გვექნება ინერციის მდგენელის დათრგუნვის გარეშე.

1.2.2. მართვადი მძიმე ბირთვი (მოდიფიცირებული მეთოდი)

განვიხილოთ მძიმე ბირთვის მეთოდის ერთი მოდიფიკაცია, რომელიც გაცილებით სწრაფია და მისი კოეფიციენტების შერჩევა შესაძლოა გაცილებით ვრცელი დიაპაზონიდან.

ვთქვათ, $f: R_n \longrightarrow R$ არის მიზნის ფუნქცია, x_0 არის ამონახსნის საწყისი მიახლოება, $\alpha \geq 0$ და $\beta > 0$ (რომელიც საკმაოდ მცირეა) მუდმივი სიდიდეებია და აღვწეროთ ალგორითმი:

ავაგოთ $\{y_i\}_{i=0}^{\infty}$ მიმდევრობა შემდეგნაირად:

- ავიღოთ $y_0 = x_0$, (1.8) ფორმულით განვსაზღვროთ შემდეგი მიახლოებები, ვიდრე მიზნის ფუნქციის მნიშვნელობები კლებულობს; როგორც კი რომელიღაც x_i-ისთვის მივიღებთ $f(x_i) \leq f(x_{i+1})$, ავიღებთ $y_1 = x_i$, $x_0 = y_1$. შევნიშნოთ, რომ $f(y_1) \leq f(y_0)$.
- ვთქვათ, უკვე აგებულია y_n ისე რომ $f(y_n) < f(y_{n-1})$ და $x_0 = y_n$. კვლავ (1.8) ფორმულით განვსაზღვროთ შემდეგი მიახლოებები, ვიდრე მიზნის ფუნქციის მნიშვნელობები კლებულობს; როგორც კი რომელიღაც x_i-ისთვის მივიღებთ $f(x_i) < f(x_{i+1})$, ავიღებთ $y_{n+1} = x_i$, $x_0 = y_{n+1}$. კვლავ გვაქვს $f(y_{n+1}) \leq f(y_n)$.
- $\{y_i\}$ მიმდევრობის წევრების განსაზღვრას ვაგრძელებთ მანამდე, ვიდრე არ დაკმაყოფილდება გაჩერების პირობა (ჩვეულებრივ, ეს უკავშირდება y_k წერტილში მიზნის ფუნქციის გრადიენტის სიმცირეს).

დავამტკიცოთ მეთოდის კრებადობა სტანდარტულ პირობებში (იხ. მაგალითად, *(Brown, 1951)*). ამისათვის შევნიშნოთ, რომ ჩვენს მიერ აგებულ $\{y_i\}_{i=0}^{\infty}$ მიმდევრობას აქვს შემდეგი თვისებები: ყოველი $k \in \{0,1,2,\ldots\}$ ინდექსისთვის, უკვე მოძებნილი y_k მიახლოება განსაზღვრავს საშუალედო მნიშვნელობას $y_k - \beta f'(y_k)$, რომლის საშუალებითაც ვპოულობთ y_{k+1}-ს (თუ როგორ ამას კრებადობის დამტკიცებისთვის მნიშვნელობა არა აქვს), ისეთს რომ კმაყოფილდება უტოლობა $f(y_{k+1}) \leq f(y_k - \beta f'(y_k))$.

თეორემა 1.1. ვთქვათ, $f(x)$ წარმოებადია R^n–ზე და $f(x)$-ის გრადიენტი აკმაყოფილებს ლიფშიცის პირობას:

$$\|f'(x) - f'(y)\| \leq L\|x - y\|; \qquad (1.11)$$

$f(x)$ ქვემოდან შემოსაზღვრულია

$$f(x) \geq f^* > -\infty; \qquad (1.12)$$

β აკმაყოფილებს პირობას:

$$0 < \beta < 2 / L; \qquad (1.13)$$

$\{y_k\}_{k=0}^{\infty}$ მიმდევრობაში y_0 ნებისმიერად აღებული ვექტორია, ხოლო ყოველი $k \in \{0,1,2,\ldots\}$ ინდექსისთვის y_k და y_{k+1} მიახლოებები ერთმანეთს მხოლოდ $f(y_{k+1}) \leq f(y_k - \beta f'(y_k))$ უტოლობის საშუალებით უკავშირდება.

მაშინ:

- $f(\cdot)$ ფუნქცია მონოტონურად კლებადია $\{y_k\}_{k=0}^{\infty}$ მიმდევრობაზე: ყოველი k-სთვის სრულდება $f(y_{k+1}) \leq f(y_k)$;
- გრადიენტი მიისწრაფვის ნულისაკენ: $\lim_{k\to\infty} f'(y_k) = 0$;

დამტკიცება. დამტკიცება იმეორებს უმარტივესი გრადიენტული მეთოდის კრებადობის სტანდარტული დამტკიცების ძირითად მომენტებს. გამოვიყენოთ ფორმულა:

$$f(x+y) = f(x) + \int_0^1 f'(x+\tau y)\cdot y^T d\tau$$

რომელიც გვძლევს:

$$f(y_k - \beta f'(y_k)) =$$

$$= f(y_k) + \int_0^1 f'\left(y_k + \tau\left(-\beta f'(y_k)\right)\right)\cdot\left(-\beta f'(y_k)\right)^T d\tau =$$

(დავამატოთ და გამოვაკლოთ $f'(y_k)\cdot\left(-\beta f'(y_k)\right)^T$)

$$= f(y_k) + f'(y_k)\cdot\left(-\beta f'(y_k)\right)^T + \int_0^1 \left[f'\left(y_k + \tau\left(-\beta f'(y_k)\right)\right) - f'(y_k)\right]\cdot\left(-\beta f'(y_k)\right)^T d\tau \leq$$

$$\leq f(y_k) - \beta\left\|f'(y_k)\right\|^2 + \left|\int_0^1 \left[f'\left(y_k + \tau\left(-\beta f'(y_k)\right)\right) - f'(y_k)\right]\cdot\left(-\beta f'(y_k)\right)^T d\tau\right| \leq$$

$$\leq f(y_k) - \beta\left\|f'(y_k)\right\|^2 + \int_0^1 \left\|f'\left(y_k + \tau\left(-\beta f'(y_k)\right)\right) - f'(y_k)\right\|\cdot\left\|-\beta f'(y_k)\right\| d\tau \leq$$

$$\le f(y_k) - \beta \left\| f'(y_k) \right\|^2 + \int_0^1 L \cdot \left\| \tau\left(-\beta f'(y_k)\right) \right\| \cdot \left\| -\beta f'(y_k) \right\| d\tau =$$

$$= f(y_k) - \beta \left\| f'(y_k) \right\|^2 + \frac{L\beta^2}{2} \left\| f'(y_k) \right\|^2 =$$

$$= f(y_k) - \beta \left(1 - \frac{L\beta}{2} \right) \left\| f'(y_k) \right\|^2 .$$

მივიღეთ:

$$f\left(y_k - \beta f'(y_k)\right) \le f(y_k) - \beta \left(1 - \frac{L\beta}{2} \right) \left\| f'(y_k) \right\|^2 .$$

რადგან $f(y_{k+1}) \le f\left(y_k - \beta f'(y_k)\right)$, ამიტომ

$$f(y_{k+1}) \le f(y_k) - \beta \left(1 - \frac{L\beta}{2} \right) \left\| f'(y_k) \right\|^2 ,$$

ანუ

$$f(y_{k+1}) \le f(y_k) - \alpha \left\| f'(y_k) \right\|^2 , \qquad (1.14)$$

სადაც $\alpha = \beta \left(1 - \frac{L\beta}{2} \right)$. ეს ფორმულა ამტკიცებს თეორემის პირველ ნაწილს.

თუ (1.14)-ის მარჯვენა მხარეში მიმდევრობით რამდენჯერმე გამოვიყენებთ იგივე ფორმულას, მივიღებთ:

$$f(y_{k+1}) \le f(y_0) - \alpha \left(\left\| f'(y_0) \right\|^2 + \cdots + \left\| f'(y_k) \right\|^2 \right) = f(y_0) - \alpha \sum_{i=0}^{k} \left\| f'(y_i) \right\|^2 .$$

(1.13)-ის გათვალისწინებით, $\alpha > 0$, ამიტომ ყოველი k -სთვის სრულდება

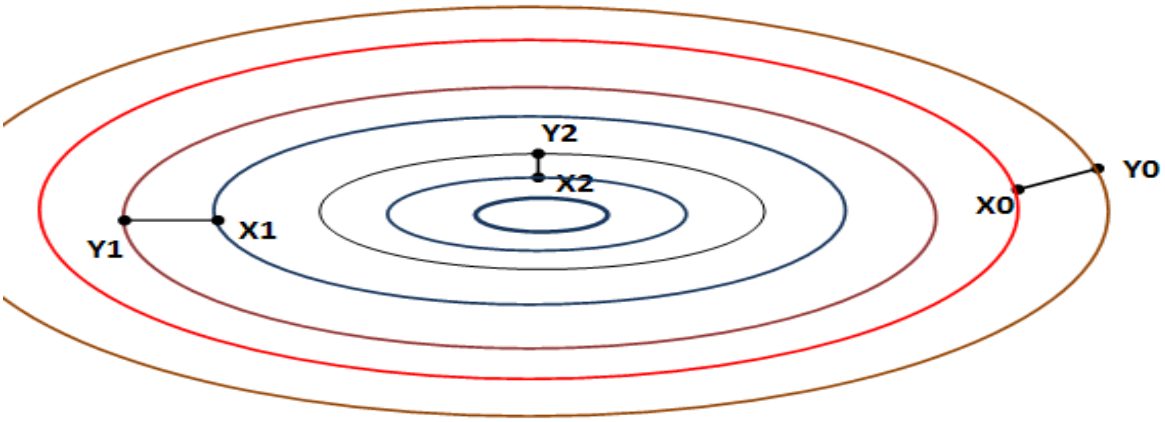
$$\sum_{i=0}^{k} \left\| f'(y_i) \right\|^2 \le \alpha^{-1} \left(f(y_0) - f(y_{k+1}) \right) \le \alpha^{-1} \left(f(y_0) - f^* \right).$$

ეს უტოლობა სრულდება ყოველი i -სთვის, და მარჯვენა მხარე დამოკიდებული არაა i -ზე, ამიტომ სრულდება აგრეთვე

$$\sum_{i=0}^{\infty} \left\| f'(y_i) \right\|^2 \le \alpha^{-1} \left(f(y_0) - f^* \right),$$

რაც ამტკიცებს, რომ $\left\| f'(y_i) \right\| \to 0$ როდესაც $i \to \infty$. თეორემა დამტკიცებულია.

თუ ყოველი $k \in \{0,1,2,\ldots\}$ ინდექსისთვის აღვნიშნავთ $x_k = y_k - \beta f'(y_k)$, მაშინ $\{y_k\}_{k=0}^{\infty}$ მიმდევრობის რამდენიმე პირველი წევრი შესაძლოა ისე იყოს განლაგებული, როგორც ნახაზ 1.2.4.-ზეა ნაჩვენები.



ნახ. 1.2.4. $x_k = y_k - \beta f'(y_k)$ -ით მიღებული მიმდევრობის საწყისი წევრები

თეორემა 1.2. თუ დამტკიცებული თეორემა 1.1-ის პირობების გარდა, მიზნის ფუნქცია არის მკაცრად ამოზნექილი, მაშინ თეორემა 1.1.-ით აგებული $\{y_i\}_{i=1}^{\infty}$ მიმდევრობა კრებადია მიზნის $f(\cdot)$ ფუნქციის გლობალური მინიმუმისაკენ.

დამტკიცება: მიზნის $f(\cdot)$ ფუნქცია არის მკაცრად ამოზნექილი, ამიტომ მას გააჩნია ერთადერთი მინიმალი $\hat{y}$, აგრეთვე კარგადაა ცნობილი, რომ

$$f'(x) = 0$$

არის მინიმუმის აუცილებელი და საკმარისი პირობა ამოზნექილი ფუნქციისათვის. კვლავ $f(\cdot)$ ფუნქციის მკაცრად ამოზნექილობიდან გამომდინარეობს, რომ სიმრავლე

$$S = \{x \in R_n \mid f(x) \le f(y_0)\}$$

შემოსაზღვრულია და ამოზნექილი. აქედან გამომდინარე $\{y_i\}_{i=1}^{\infty}$-ს აქვს ზღვარი. ვთქვათ, $\tilde{y}$ ნებისმიერი ზღვრული წერტლია $\{y_i\}_{i=1}^{\infty}$-ის და $y_{i_j} \to \tilde{y}$, როდესაც $j \to \infty$. $f(\cdot)$ უწყვეტად დიფერენცირებადი ფუნქციაა და $\|\cdot\|$ აგრეთვე უწყვეტი ფუნქციაა, ამიტომ

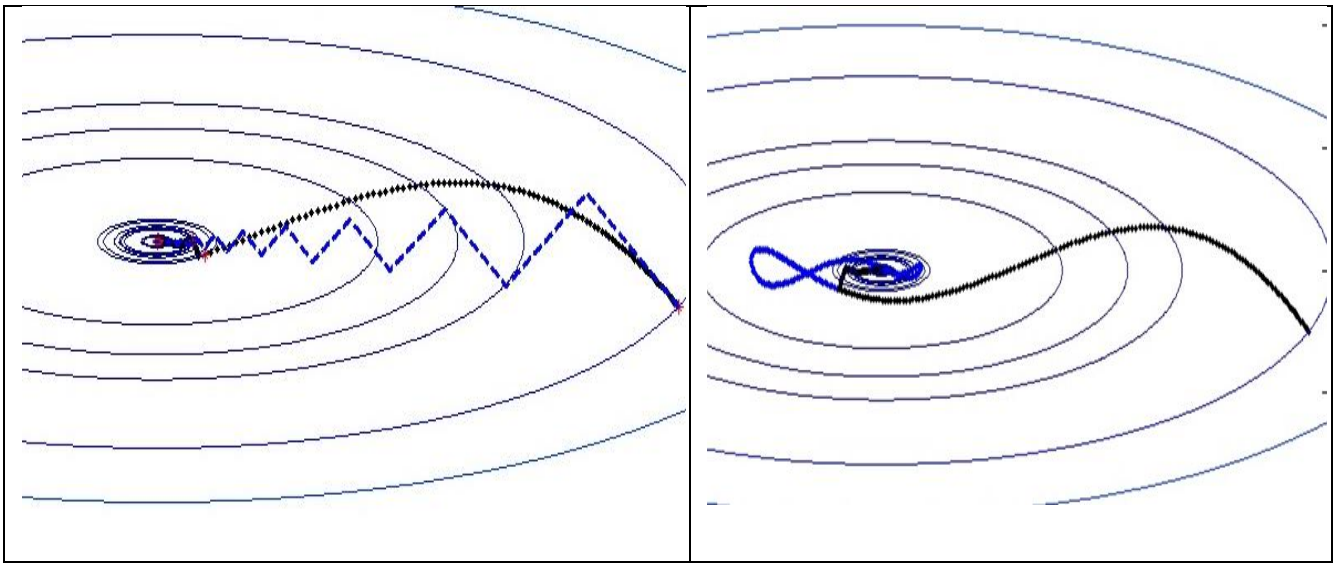
$$0 = \lim_{j\to\infty} \left\| f'\left(y_{i_j}\right) \right\| = \|f'(\hat{y})\|$$

$0 = f'(\tilde{y}) = f'(\hat{y})$ აქედან გამომდინარეობს $\tilde{y} = \hat{y} \cdot \tilde{y}$ -ის ნებისმიერობიდან გამომდინარე $\{y_i\}_{i=1}^{\infty}$-ის ნებისმიერი ქვემიმდევრობა იკრიბება $\tilde{y}$-ისკენ. ეს კი ნიშნავს რომ $\{y_i\}_{i=1}^{\infty}$ მიმდევრობაც იკრიბება $\tilde{y}$-ისკენ. თეორემა დამტკიცებულია.

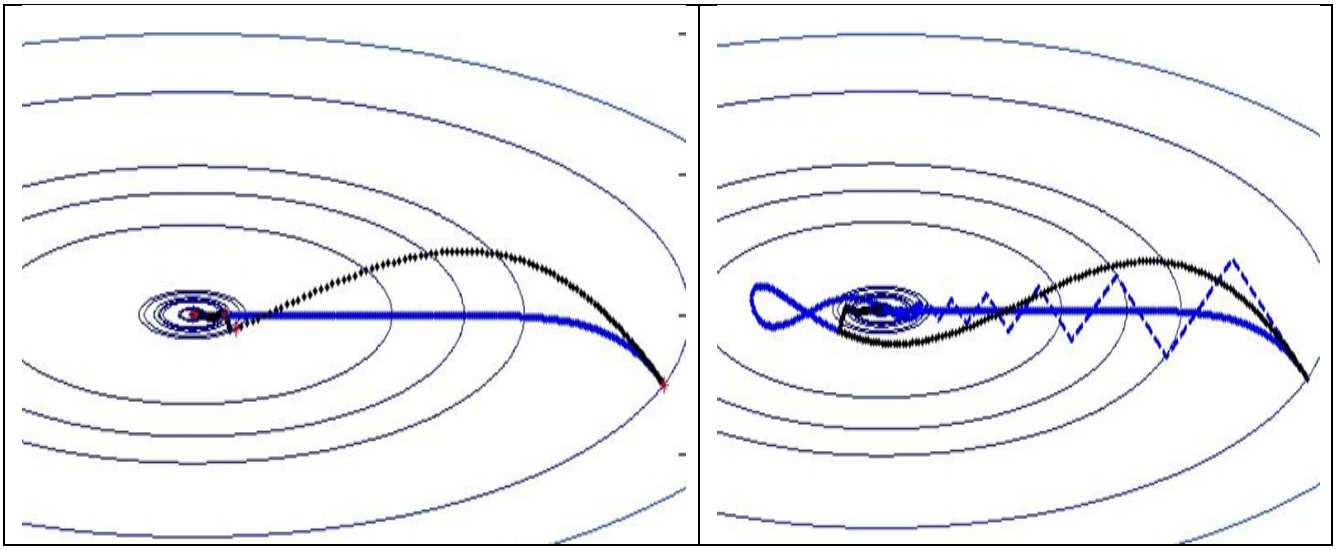
1.2.3. მეთოდის გეომეტრიული გააზრება კვადრატული მიზნის ფუნქციისთვის

ნახაზ 1.2.5.-ზე მოცემულია მინიმიზაციის სამი გავრცელებული მეთოდის ტრაექტორიასთან დაწყვილებული მართვადი მძიმე ბირთვის ტრაექტორია. მიზნის ფუნქციად აღებულია $x^2 + 10y^2$ ფუნქცია. საწყისი პირობები ყველგან ერთი და იგივეა.

სტანდარტული და მართვადი მძიმე ბირთვების კოეფიციენტები ტოლია. მართვადი ბირთვის ”დამუხრუჭების” წერტილები წითელი წერტილებითაა მონიშნული. ნახ. 1.2.5. ა) უჯრაში ერთდროულად ნაჩვენებია უსწრაფესი დაშვების და მართვადი ბირთვის ტრაექტორიები (სხვა მეთოდებთან დაკავშირებული ტერმინოლოგიის შესახებ იხ. ბოლო პუნქტში). ბ) ნახაზზე ჩანს სად ჩერდება მართვადი ბირთვი, განსხვავებით სტანდარტული მძიმე ბირთვისგან, და მერე როგორ ხდება მისი მოძრაობა. გ) ნახაზზე მეთოდს ვადარებთ სტანდარტულ გარადიენტულ მეთოდს ფიქსირებული ბიჯით. დ) ნახაზზე ნაჩვენებია ოთხივე მეთოდის ტრაექტორია ერთდროულად:



ა) ბ)



გ) დ)

ნახ. 1.2.5. ა) უსწრაფესი დაშვების და მართვადი ბირთვის ტრაექტორიები; ბ) მართვადი და მძიმე ბირთვის ტრაექტორიები; გ) მართვადი ბირთვი და სტანდარტული გარადიენტული მეთოდი; დ) ოთხივე მეთოდის ტრაექტორია .

მომდევნო ნახაზზე 1.2.6. ნაჩვენებია ტრაექტორიები როზენბროკის ფუნქციის მინიმიზაციისთვის, რომლებიც შესაბამისად მიიღება შემდეგი მეთოდების გამოყენებით:

ა) უსწრაფესი დაშვების;

ბ) მძიმე ბირთვის;

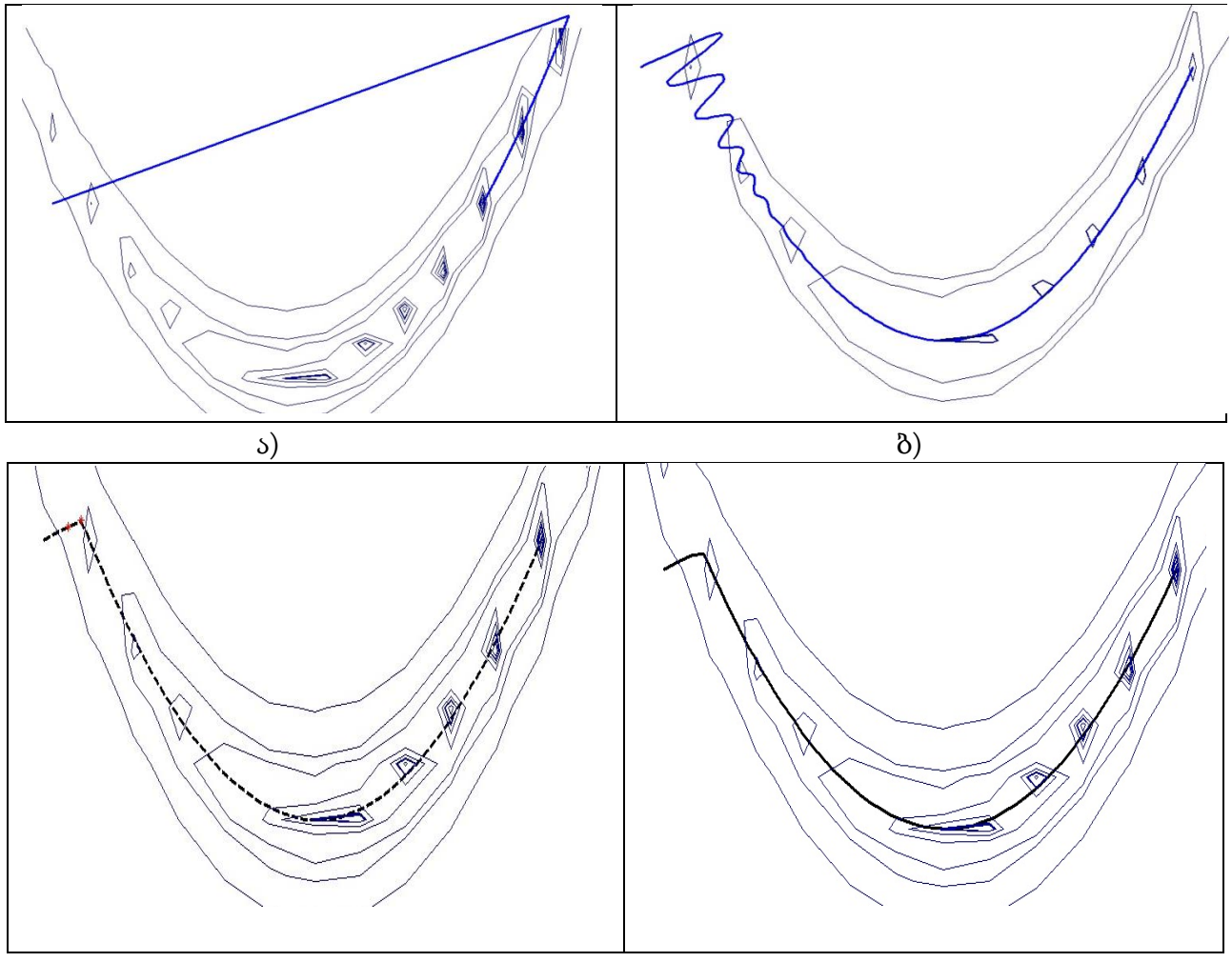
გ) მართვადი ბირთვის;

დ) სტანდარტული გრადიენტული მეთოდი ფიქსირებული ბიჯით;

(როზენბროკის ფუნქციის შესახებ შედარებით დეტალური ინფორმაცია იხ. ქვემოთ).

როგორც ვხედავთ, იგივე ეფექტი, რაც კვადრატული ფუნქციის შემთხვევაში მძიმე და მართვადი ბირთვების შედარებისას იჩენს თავს ამონახსნის სიახლოვეს, როზენბროკის ფუნქციის შემთხვევაში თავს იჩენს ტრაექტორიის დასაწყისშივე.

ასევე საინტერესოა, რომ მოდიფიცირებული მძიმე ბირთვის მეთოდის ტრაექტორია საკმაოდ ახლოსაა სტანდარტულ გრადიენტულ მეთოდთან, თუმცა მოდიფიცირებული მეთოდი გაცილებით სწრაფია.



ა) ბ)

გ) დ)

ნახ. 1.2.6. როზენბროკის ფუნქციის მინიმიზაცია სხვადასხვა მეთოდით: ა) უსწრაფესი დაშვების; ბ) მძიმე ბირთვის; გ) მართვადი ბირთვის; დ) სტანდარტული გრადიენტული მეთოდი ფიქსირებული ბიჯით;

უსწრაფესი დაშვების მეთოდის ტრაექტორიის მკვეთრად განსხვავებული სახე აიხსნება იმ გარემოებით, რომ ეს მეთოდი, ისევე როგორც ფლეტჩერ-რივსის მეთოდი, არსებითადაა დამოკიდებული გამოყენებული ერთგანზომილებიანი მინიმიზაციის ამოცანის ალგორითმზე. როდესაც ერთგანზომილებიანი ალგორითმი ცდილობს სწრაფად (და უხეშად) დაადგინოს ძებნის ინტერვალის ბოლოები, არაამოზნექილი ფუნქციის შემთხვევაში შესაძლოა ერთგანზომილებიანი მინიმალი არ იყოს მოცემული მიმართულებით გლობალური, ან კარგი ლოკალური მინიმალი. განსხვავებული რეალიზაციის პირობებში, შესაძლებელია განსხვავებული ტრაექტორიის მიღება.

1.2.4. მართვადი ბირთვის ალგორითმის ტესტირების შედეგები

სამეცნიერო ლიტერატურაში იშვიათად გვხვდება ნაშრომები, რომლებიც სხვადასხვა ალგორითმის შესაბამისი პროგრამული კოდის ურთიერთშედარებას და ურთიერთანალიზს ეძღვნება. ერთი მიზეზი შესაძლოა ისაა, რომ ასეთი ტიპის ნაშრომების დასკვნები ზოგჯერ სუბიექტურია და ამიტომ ეწინააღმდეგება ერთმანეთს. ჩვენ შევეცადეთ, რომ ტესტირება ჩაგვეტარებინა იგივე ტექნოლოგიური საშუალებების გამოყენებით რაც გამოიყენება (Press, et al., 2007)-ში, რომელიც გარკვეული აზრით დეფაქტო სტანდარტს წარმოადგენს. მეთოდოლოგიური მოსაზრებებით, (Jorge J. More, et al., 1981; Ravindran, et al., 2006)-იდან ჩვენ შევარჩიეთ რამდენიმე ძალიან კარგად ცნობილი მინიმიზაციის ამოცანა, რომლებისთვისაც (მართვადი მძიმე ბირთვის გარდა) სხვა ალგორითმების ტესტები ცნობილი იყო. თუმცა, ჩვენ საჭიროდ ჩავთვალეთ რამდენიმე ალგორითმის საკუთარი იმპლემენტაცია გაგვესინჯა და შეგვედარებინა მართვადი მძიმე ბირთვის მეთოდისთვის.

მართვადი ბირთვის ალგორითმის კოდი უაღრესად მარტივი დასაწერია. შესაძლოა სწორედ ამის გამო, იგი უაღრესად საიმედოდ მუშაობს, რაშიც დავრწმუნდებით ტესტირების შედეგების განხილვისას.

მართვადი ბირთვის ალგორითმის აღწერა და იმპლემენტაცია, სიმარტივისთვის მოთავსებულია ერთ ფაილში: “ControllableHeavyBall”. შევნიშნოთ, რომ STL ბიბლიოთეკის საშუალებების გამოყენების წყალობით, არ არის ინდექსირებული ცვლადების გამოყენების აუცილებლობა, ტრაექტორიაში წერტილების საჭირო რაოდენობის დამახსოვრება მარტივად რეგულირდება. რადგან შეზღუდული მოცულობის გამო მოგვყავს მხოლოდ მართვადი ბირთვის კოდი, ამიტომ მას არ ვათავსებთ ცალკე `namespace`-ში. (Polyak, 1987)-ისგან განსხვავებით, `"ControllableHeavyBall.h"` ფაილი მიზნის ფუნქციის შესახებ საჭირო ინფორმაციას იღებს `"data.h"` ფაილიდან, რომელიც საჭიროების შემთხვევაში შეიძლება შეიცვალოს, რაც კოდს მოქნილობას მატებს.

ყურადღება მივაქციოთ, რომ სტანდარტული მძიმე ბირთვის შემთხვევაში

```
double alpha =  1.05;
```

სტრიქონი უეჭველად გამოიწვევდა მეთოდის დაციკვლას. ჩვენს მეთოდში ეს ერთერთი მძლავრი რესურსია კრებადობის დასაჩქარებლად. სხვა განსხვავებული და საინტერესო რესურსი, ჩვენი აზრით, არის წევრი `inertia`, რომლის განულება ხდება ყოველთვის, როდესაც ტრაექტორიის გასწვრივ მიზნის ფუნქციის გაზრდის საშიშროება ჩნდება. ეს ნიშნავს რომ ჩვენ ვამუხრუჭებთ ბირთვს ამ მომენტში და მერე მას უწევს სიჩქარის

აკრეფა. საინტერესო პერსპექტივას ქმნის ინერციის შენარჩუნება გარკვეულ დონეზე, ოღონდ მისი მობრუნება საჭირო მიმართულებით, რაც შექმნიდა მოძრაობაში ბირთვის მობრუნების ეფექტს. მოცემულ ნაშრომში ჩვენ, სიმარტივისთვის, ამ ეფეტზე არ ვამახვილებთ ყურადღებას.

კოდის სიმარტივის გამო, მეტი კომენტირების აუცილებლობა არაა:

```
//file "ControllableHeavyBall.h"
#include "data.h"  //მიმართვა მონაცემებზე
#include <iostream>
#include <vector>
const double EPS = 1E-6;
double beta = 1E-4;
double alpha =  1.05;
vector<double> inertia(DIMENSIONALITY);
vector<vector<double> > trajectory;
template<typename T>
void getNextIteration( vector<T> & x0)
{
    vector<T> x1(DIMENSIONALITY,0.0);
    while(true)
    {
        objFunctionPrime(x0);
        for(int i=0; i < DIMENSIONALITY; ++i)
            x1[i] = x0[i] + alpha *inertia[i] - beta*yPrime[i];
        if(objFunction(x0)<= objFunction(x1))
            return;
        for(int i=0; i < DIMENSIONALITY; ++i)
            inertia[i] = x1[i]-x0[i];
        x0 = x1;
        if(dist(x0, trajectory[trajectory.size()-1])>0.2)
               trajectory.push_back(x0);
    }
}
template<typename T>
void getMinimal( vector<T> & x0)
{
    while(true)
    {
        getNextIteration(x0);
        solutionIterations.push_back(x0);
        for(int i=0; i < DIMENSIONALITY; ++i)
            inertia[i] = 0.0;
        if(norm(yPrime)<EPS )
            break;
    }
}
```

იგულისხმება, რომ "ControllableHeavyBall.h" ფაილი "data.h" ფაილიდან მიიღებს შემდეგ ინფორმაციას (პაუელის ფუნქციის შემთხვევაში):

```
const int DIMENSIONALITY = 4;           //ამოცანის განზომილება
double array[]={3.0, -1.0, 0.0, 1.0};  //{0.0, 0.0};
vector<double> x0(array,array+4);       //საწყისი წერტილი
double y(0.0);                          //წერტილი ფუნქციის
მნიშვნელობების შესანახად
vector<double> yPrime(DIMENSIONALITY); //ვექტორი ფუნქციის
                                        //წარმოებულის შესანახად
template<typename T>
T objFunction( vector<T> &x)
{   y =(x[0]+10*x[1])*(x[0]+10*x[1]) + 5*(x[2]- x[3])*(x[2]-
    x[3])+pow((x[1]-2*x[2]), 4) + 10*pow((x[0] - x[3]), 4);
    return y;
}
template<typename T>
void objFunctionPrime(const vector<T> & x)
{   yPrime[0]=2*(x[0] + 10*x[1]) + 40*pow((x[0] - x[3]),3);
    yPrime[1]=20*(x[0] + 10*x[1]) + 4*pow((x[1] - 2*x[2]), 3);
    yPrime[2]=10*(x[2]-x[3]) - 8*pow((x[1]-2*x[2]),3);
    yPrime[3]=-10*(x[2]-x[3]) - 40*pow((x[0]-x[3]), 3);
}
```

რადგან ჩვენს ძირითად მიზანს წარმოადგენს მკვლევარების ყურადღების მიპყრობა ამ მიდგომისადმი და არა საკითხის ამომწურავი გამოკვლევა ნაშრომის ფარგლებში, ამიტომ ძირითადი ყურადღება გავამახვილეთ ალგორითმის შესრულების დროზე. ალგორითმები მუშაობას ამთავრებენ, როდესაც წარმოებული მოცემულ სიზუსტეს აკმაყოფილებს. სიზუსტის შერჩევის კრიტერიუმად კონკრეტულ ტესტებზე ჩვენ ჩავთვალეთ ის მინიმალური მნიშვნელობა, რომლისთვისაც ფლეტჩერ-რივსის მეთოდი გამართულად მუშაობდა. ტესტირება ჩატარდა სტანდარტული სიმძლავრის პერსონალურ კომპიუტერზე მონაცემებით: Intel(R) Pentium(R) Dual CPU E2200 2.20 GHz, 2.00 GB of RAM. ტესტირებისთვის გამოყენებული მეთოდები (ზოგიერთი პარამეტრის მითითებით) ჩამოთვლილია სვეტებში, ხოლო ფუნქციების სახელები - სტრიქონებში.

ტესტირების შედეგები მოყვანილია **1.2.1.** ცხრილში. თითოეულ უჯრაში მოყვანილია სტრიქონის შესაბამისი მეთოდის სვეტის შესაბამის ფუნქციაზე ტესტირების შედეგები. უჯრის ყოველ სტრიქონში მოყვანილია წყვილი (მაგალითად, 0.001 /1E-12), რომლის პირველი წევრი არის დრო წამებში (გასაშუალებული რამდენიმე ცდის შედეგებისთვის), ხოლო მეორე - სიზუსტე. უპირველეს ყოვლისა, ჩვენ განვიხილავთ სიზუსტეს, რომელიც ყველა მეთოდისთვის საერთოა (ძირითადად, ესაა 1E-6, თუმცა ფლეტჩერის მეთოდისთვის გარკვეული გამონაკლისი კეთდება). შესაძლებლობის შემთხვევაში, მოგვყავს გაორმაგებული სიზუსტის შესაბამისი დრო, ან მივუთითებთ იმ სიზუსტეს, რომლისთვისაც ჩვენს მიერ გამოყენებული იმპლემენტაცია მოცემულ ფუნქციაზე წყვეტს სტაბილური შედეგის

ჩვენებას (--- /1E-12 სახით, მაგალითად). მძიმე და მართვადი მძიმე ბირთვის შემთხვევაში, ბიჯის სიგრძედ ყველგან აღებულია

```
double beta = 1E-4;
```

და ჩვენ არ ვცდილობთ ამ პარამეტრის შერჩევის ხარჯზე ამ მეთოდების კრებადობის დაჩქარებას, რაც გარკვეულ უპირატესობას ანიჭებს დანარჩენ სამ მეთოდს. თუმცა, უსწრაფესი დაშვების (Steepest descent) და ფლეტჩერ-რივსის (Fletcher-Reeves) მეთოდებში ერთგანზომილებიანი მინიმიზაციისთვის ვიყენებთ ოქროს კვეთის მეთოდს, რაც აგრეთვე არაა საუკეთესო არჩევანი.

ცხრილი 1.2.1. შედეგები სატესტო ამოცანებისათვის

	იაზონის (Eason) ფუნქცია	პაუელის (Powell) ფუნქცია	როზენბროკის (Rosenbrock) ფუნქცია	ვუდის (Wood) ფუნქცია
უსწრაფესი დაშვების მეთოდი	0.001 /1E-5 0.001 /1E-7 0.001 /1E-12	1.123 /1E-5 10.4342 /1E-7	1.096 /1E-5 1.1156 /1E-7	---
სტანდარტული გრადიენტული მეთოდი	1.321 /1E-5 1.3996 /1E-7 3.085 / 1E-11 --- / 1E-12	26.489 /1E-5 26.525 /1E-5 122.453 / 1E-6 ---- / 1E-7	0.586 /1E-5 0.849 /1E-7 1.503 /1E-12	2.28 /1E-5 2.53 /1E-7 3.02 /1E-11 --- /1E-12
მართვადი ბირთვი alpha = 1.05; beta = 1E-4	0.002 /1E-5 0.005 /1E-6 --- /1E-7	0.005/1E-5 0.006 /1E-6 0.011 /1E-7 0.025 /1E-12	0.001 /1E-5 0.002 /1E-6 0.002 /1E-7 0.004 /1E-12	0.006 /1E-5 0.009 /1E-6 0.007 /1E-7 0.01 /1E-12
მართვადი ბირთვი alpha = 1.25; beta = 1E-4	0.001 /1E-6 --- /1E-7	0.017 /1E-6 0.038 /1E-7 0.208 /1E-12	0.001 /1E-6 0.003 /1E-7 0.003 /1E-12	0.023 /1E-6 0.023 /1E-7 0.038 /1E-11 --- /1E-12
მართვადი ბირთვი alpha = 2. 05; beta = 1E-4	0.001 /1E-5 0.001 /1E-6 --- /1E-7	0.857/1E-5 3.994 /1E-6 18.488 /1E-7 --- /1E-8	0.007 /1E-5 0.011 /1E-6 0.011 /1E-7 0.013 /1E-12	0.029 /1E-5 0.031 /1E-6 0.035 /1E-7 0.046 /1E-11 --- /1E-12
მძიმე ბირთვი alpha = 0.96 beta = 1E-4	0.378 /1E-5 0.395 /1E-7 0.429 /1E-12	0.495 /1E-5 10.556 /1E-7 --- /1E-8	0.023 /1E-5 0.03 /1E-7 0.056 /1E-12	----
ფლეტჩერ-რივსის მეთოდი	0.001 /1E-5 0.001 /1E-6 --- /1E-7	2.711 /1E-5 --- /1E-6	0.096 /1E-5 --- /1E-6	0.421 /1E-5 --- /1E-6

შევნიშნოთ აგრეთვე, რომ სამეცნიერო ლიტერატურაში ადგილი აქვს ტერმინების აღრევას. რასაც ჩვენ ვუწოდებთ სტანდარტულ გრადიენტულ მეთოდს ფიქსირებული ბიჯით, იგივე მეთოდს, მაგალითად Wikipedia-ს მიხედვით, ზოგჯერ უწოდებენ უსწრაფესი დაშვების მეთოდს (Steepest descent). ჩვენ ტერმინს "უსწრაფესი დაშვება" (Steepest descent) ვიყენებთ (Polyak, 1987)-ის მიხედვით.

თავი 2. ოპტიმიზაციის ამოცანის ამოხსნა გენეტიკური ალგორითმის გამოყენებით

2.1. გენეტიკური ალგორითმის ზოგადი სქემა

გენეტიკური ალგორითმებით ოპტიმიზაციის ამოცანების ამოხსნისას ამოცანის პარამეტრები წარმოიდგინება კოდირებული მნიშვნელობებით-გენებით, გენების ერთობლიობა ქმნის ქრომოსომებს. ქრომოსომებისაგან შედგება პოპულაცია. ყოველ ქრომოსომას შეესაბამება სარგებლიანობის ფუნქცია, რომელიც მოცემული ინდივიდის ხარისხის ზომას განსაზღვრავს მიმდინარე ამონახსნში. უფრო ხშირად სარგებლიანობის ფუნქციად იღებენ საწყისი ამოცანის მიზნის ფუნქციას.

გა-ს (**გა** - გენეტიკური ალგორითმი) ყოველ იტერაციაზე გენეტიკური ოპერატორების გამოყენებით ხდება საწყისი პოპულაციის ევოლუცია, ანუ იცვლილება ქრომოსომაში შემავალი ინფორმაცია. დღეისათვის შემუშავებულია **გა**-ს უამრავი მოდელი. ქვემოთ შემოთავაზებული ალგორითმი ეყრდნობა გენეტიკური ალგორითმების ზოგად პრინციპებს.

ვიდრე ჩვენი მოდიფიცირებული ალგორითმის აღწერას შევუდგებოდეთ ზოგადად განვიხილოთ რაში მდგომარეობს გენეტიკური ალგორითმების არსი და მათი მუშაობის პრინციპები. გენეტიკური ალგორითმები უახლესი ალგორითმებია და ისინი სხვადასხვა ფორმით გამოიყენებიან მრავალი სამეცნიერო და ტექნიკური პრობლემის გადასაწყვეტად. გენეტიკური ალგორითმები გამოიყენება გამოთვლითი სტრუქტურების, მაგალითად, ნეირონული ქსელების პროექტირებისას. ისინი გამოიყენებიან სხვადასხვა საგნობრივ სფეროებში: ბიოლოგიისა და სოციოლოგიის სისტემებში. შესაძლებელია გენეტიკური ალგორითმების გამოყენება ოპტიმიზაციის ამოცანების ამოხსნისას. რეალური ამოცანების ამოხსნისას შესაძლოა მოითხოვებოდეს პარამეტრების იმ მნიშვნელობების განსაზღვრა, რომლებიც მოგვცემენ ზუსტ ამონახსნს, სხვა შემთხვევაში ზუსტი ოპტიმუმის პოვნა არაა საჭირო, არამედ ამონახსნად შეიძლება ჩაითვალოს ნებისმიერი ამონახსნი, რომელიც უკეთესია მოცემულ ამონახსნთან შედარებით. ასეთ შემთხვევაში გენეტიკური ალგორითმი მისაღები მეთოდია საჭირო ამონახსნის საპოვნელად. გენეტიკური ალგორითმების ძალა მდგომარეობს მათ მიერ მრავალ პარამეტრთან ერთდროულად მანიპულირებაში. დღესდღეობით გენეტიკური ალგორითმები დიდ კონკურენციას უწევენ ოპტიმიზაციის ტრადიციულ ძიების მეთოდებს და საკმაოდ წარმატებითაც.

გენეტიკური ალგორითმები პირველად შეიმუშავა ჯონ ჰოლანდმა (Holland, 1975), როგორც ბუნებრივი ევოლუციის გზის კომპიუტერული რეალიზაცია. ბუნებრივი ევოლუცია განიხილავს დაწყებული სიცოცხლის უმარტივესი ფორმებიდან როგორიცაა, ერთუჯრედიანები, ასევე რთული ორგანიზმების (მაგალითად, მცენარეები და ცხოველები) განვითარებას. ჰოლანდის გენეტიკური ალგორითმები, ეს ბუნებრივი ევოლუციის და მემკვდრეობითობის მარტივი მოდელებია. მცენარეების და ცხოველების ზრდის პროცესი თესლიდან ან კვერცხუჯრედიდან დაწყებული, კონტროლირდება იმ გენებით, რომლებიც მათ მემკვიდრეობით მიიღეს მშობლებისაგან. გენები ინახება დნმ (დეზოქსირიბონუკლეინის მჟავა)-ის ერთ ან რამდენიმე სპირალში. უსქესო გამრავლების შემთხვევაში

შვილების დნმ წარმოადგენეს მშობლის დნმ-ის ასლს, თუმცა შესაძლებელია მცირე შემთხვევითი ცვლილებაც, რომელსაც მუტაციას უწოდებენ. სქესობრივი გამრავლების გზით შვილი დნმ-ს მემკვიდრეობით იღებს ორივე მშობლიდან. ხშირად დაახლოებით ორივე მშობლიდან ნახევარი დნმ-ის კოპირება ხდება შვილში, სადაც ისინი შეერთდებიან.ზოგადად შვილების დნმ განსხვავდება ორივე მშობლის დნმ-ისაგან. ბუნებრივი ევოლუცია მიმდინარეობს იქ, სადაც ძლიერი ინდივიდები გადარჩებიან და დნმ-ს შემდეგ თაობებს გადასცემენ მემკვიდრეობით. ანუ პოპულაციები ვითარდებიან მასში შემავალი ინდვიდების გადარჩევის გზით. გენეტიკური ალგორითმები შეიცავენ სასინჯ პოპულაციებს. ყოველი ინდივიდი მოდელირებულია რაიმე სტრიქონში, რომელიც ამ ინდივიდის დნმ-ს შეესაბამება. პოპულაციები "ევოლუციონირდებიან"- მრავალჯერადი სელექციის (გადარჩევის) გზით მიიღება ახალი "ამონახსნები" და მათი ჩანაცვლებით ძველი "ამონახსნების" ადგილზე.

გენეტიკური ალგორითმის იდეა ოპტიმიზაციის ამოცანებისათვის კი, შემდეგში მდგომარეობს: უნდა წარმოვიდგინოთ ხელოვნური სამყარო, რომელშიც ბევრი ინდივიდია, ამასთან ყოველი მათგანი არის მოცემული ამოცანის რაიმე ამონახსნი. ჩავთვალოთ, რომ ინდივიდი უფრო მეტად ვარგისია, რაც უფრო უკეთეს ამონახსნს გვაძლევს. მაშინ ოპტიმიზაციის ამოცანის ამოხსნა მიიყვანება რაც შეიძლება საუკეთესო სარგებლიანობის მქონე ინდივიდის მოძებნაზე. ცხადია, ყველა ინდივიდის ნახვა შეუძლებელია მათი მრავალრიცხოვნობის გამო, მაგრამ შესაძლებელია თაობათა სიმრავლეების განხილვა. თუ ჩვენ ამ პროცესში გავითვალისწინებთ გენეტიკურ მემკვიდრეობას, მაშინ მივიღებთ გარემოს, რომელიც დაექვემდებარება ევოლუციურ პროცესს. ამ ხელოვნური ევოლუციური პროცესის მიზანი კი იქნება საუკეთესო ამონახსნის შექმნა-განსაზღვრა. ცხადია ევოლუცია უსასრულო პროცესია, რომლის მიმდინარეობისას ინდივიდების სარგებლიანობა თანდათან იზრდება. ჩვენ იძულებულები ვართ ეს პროცესი რომელიმე ეტაპზე შევწყვიტოთ და მიღებული თაობებიდან ამოვარჩიოთ საუკეთესო სარგებლიანობის მქონე ინდივიდი. ამით აბსოლუტურად ზუსტ ამონახსნს ვერ მივიღებთ, მაგრამ ეს ამონახსნი შესაძლოა ძალიან ახლოს იყოს ოპტიმალურთან.

მაშასადამე, გენეტიკური ალგორითმები ახდენენ გენეტიკური მემკვიდრეობითობის და ბუნებრივი გადარჩევის პროცესის იმიტირებას. ამ პროცესში ინდივიდები ერთის მხრივ ინარჩუნებენ მშობლებისაგან მიღებულ ძირითად თვისებებს, მეორეს მხრივ განიცდიან მუტაციას და იძენენ ახალ თვისებებს. ეს ახალი თვისებებიც შენარჩუნდება და გადაეცემა მემკვიდრეობით. მოკლედ კი გენეტიკური ალგორითმების მუშაობის პრინციპები შეიძლება ასე აღვწეროთ: იმისათვის რომ ვისაუბროთ გენეტიკურ მემკვიდრეობითობაზე პირველ რიგში ჩვენი ინდივიდები უნდა ვაქციოთ ქრომოსომებად. გენეტიკურ ალგორითმში ქრომოსომა - ეს არის რიცხვითი მასივი, რომელიც შეესაბამება რომელიმე შერჩეულ პარამეტრს.

მარტივი გენეტიკური ალგორითმი შემთხვევითი წესით აგენერირებს საწყის პოპულაციას. გენეტიკური ალგორითმის მუშაობის პროცესი არის იტერაციული და ის მთავრდება მომხმარებლის მიერ არჩეული გარკვეული რაოდენობის თაობების შეცვლის, ან სხვა

კრიტერიუმის შესრულების შემდეგ. გენეტიკური ალგორითმის ყოველ თაობაში ხდება პოპულაციის სარგებლიანობის შემოწმება, შეჯვარება და მუტაცია.

იმისათვის, რომ ეფექტური გენეტიკური ალგორითმი მიეღოთ უამრავი კვლევა ჩატარდა და დღესაც გრძელდება მუშაობა ამ მიმართულებით.

გენეტიკური ალგორითმები გამარტივებული სახით ინარჩუნებს ბიოლოგიურ ტერმინოლოგიას. გამოსაკვლევი მიზნის ფუნქცია $f(x)$ ეკვივალენტურია ცოცხალი ორგანიზმის სარგებლიანობის ცნებისა. მიზნის ფუნქციის არგუმენტს $x = (x_1,\ x_2, \dots, x_n) \in R^n$ უწოდებენ ფენოტიპს, x ვექტორის ცალკეულ x_i კომპონენტებს უწოდებენ თვისებებს. გენეტიკურ ალგორითმებში x_i-ებს წარმოადგენენ რაიმე s_i ფორმით, რომელიც მომხმარებლის შეხედულებისამებრ უფრო მოსახერხებელია ალგორითმის რეალიზაციისათვის. s_i-ებს გენებს უწოდებენ, ამგვარად მთლიანად x ვექტორი ჩაიწერება ახალი $s = (s_1,\ s_2, \dots, s_n)\ \epsilon\ S$ სტრუქტურული ფორმით და მას ქრომოსომას, გენოტიპს ან ინდივიდს უწოდებენ. როგორც წესი უნდა არსებობდეს ურთიერთცალსახა გარდაქმნის წესი-მექანიზმი ფენოტიპისა გენოტიპში და პირიქით. S სივრცისთვის შემოაქვთ მიზნის ფუნქციის მსგავსი $\varphi(s)$ ფუნქცია $\varphi: S \to R$, ისეთი რომ $f(x) \to min, x \in R_n$ ამოცანის ამოხსნა მიიყვანება $\varphi(s) \to min, s \in S$ ამოცანის ამოხსნაზე. $\varphi(s)$ ფუნქციას ფიტნეს ფუნქციას უწოდებენ. $s^k = \left(s_1^k, s_2^k, \dots,\ s_n^k\right),\ k = 1,2, \dots, m$ ნაკრებებს, უწოდებენ პოპულაციებს. სადაც s^k - k ნომრიანი პოპულაციაა, ხოლო m - პოპულაციის ზომა, s_i^k- i-ური გენია k-ურ პოპულაციაში.

კლასიკური გენეტიკური ალგორითმი არაფორმალურად შეიძლება აღვწეროთ შემდეგნაირად:

ალგორითმი 2.1. ზოგადი გენეტიკური ალგორითმი

ნაბიჯი 1. საწყისი პოპულაციის არჩევა.

ნაბიჯი 2. სარგებლიანობის ("ფიტნეს") ფუნქციის განსაზღვრა და შესაბამისად მიღებულ პოპულაციაში ინდივიდების სარგებლიანობის შეფასება.

ნაბიჯი 3. სელექცია-სარგებლიანობის მიხედვით მშობლების არჩევა.

ნაბიჯი 4. რეპროდუქცია და პოპულაციის შეცვლა : შერჩეულ ინდივიდებზე გენეტიკური ოპერატორების: შეჯვარების და მუტაციის გამოყენება და შთამომავალი ინდივიდების მიღება. მიღებული შთამომავალი ინდივიდების სარგებლიანობის შეფასება და სარგებლიანობის მიხედვით საწყის პოპულაციაში ერთ-ერთი მშობლის ჩანაცვლება საუკეთესო შთამომავლით.

ნაბიჯი 5. დასასრული, თუ შესრულებულია ალგორითმის დასრულების პირობა, წინააღმდეგ შემთხვევაში გადადი ნაბიჯ 3-ზე.

მე-5 ნაბიჯში "ალგორითმის დასრულების პირობა"-ს ქვემოთ დავაზუსტებთ, ალგორითმის ბიჯების დახასიათებისას.

დავახასიათოთ კლასიკური გენეტიკური ალგორითმის ბიჯები:

1. **საწყისი პოპულაციის არჩევა-ინიციალიზაცია** (Initialization)-საწყისი პოპულაცია აირჩევა შემთხვევითად საძიებო არის მნიშვნელობათა სიმრავლიდან. პოპულაციის ზომა-ქრომოსომების რაოდენობა დამოკიდებულია კონკრეტულ ამოცანაზე.

2. **სარგებლიანობის** (ფიტნეს, Fitness function) ფუნქციის— განსაზღვრა. ამ ფუნქციით განისაზღვრება თუ რომელი ამონახსნი რამდენად კარგია, ანუ განსაზღვრავს გენის სიძლიერეს, რამდენადაც ძლიერია გენი მით მეტი შანსი აქვს რომ გადარჩეს. სწორედ ასეთი გენებს ექნებათ დიდი შანსი რომ სელექციის დროს დარჩნენ არჩეულ ინდივიდებს შორის.

3. **სელექცია** (Selection)- სელექცია ეს არის ოპერაცია რომელიც გულისხმობს სარგებლიანობის ფუნქციის მნიშვნელობების შესაბამისად გადაირჩეს ინდივიდები შემდგომი შეჯვარებისათვის. აქ გათვალისწინებული უნდა იქნეს ის სიტუაცია რომ, თუ ავირჩევთ ყველაზე საუკეთესო ქრომოსომებს, გამოირიცხება მათი მრავალფეროვნება და კრებადობის პროცესი დაჩქარდება, სუსტი გენების არჩევა კი ევოლუციის პროცესს შეანელებს, ანუ ამ აზრის გათვალისწინებით უნდა მოხდეს შერჩევის დაბალანსება. არსებობს სელექციის რამდენიმე მეთოდი, მათგან ყველაზე გავრცელებულია:

- პანმიქსია(panmixia) (შემთხვევითი თანაბარალბათური არჩევა)- ამ დროს ინდივიდების არჩევა ხდება მთლიანი პოპულაციიდან და ნებისმიერი მათგანი შესაძლოა მოხვდეს შესაჯვარებელ წყვილში. ეს მეთოდი მიუხედავად სიმარტივისა უნივერსალურია სხვადასხვა კლასის ამოცანებისათვის.

- რანგების მეთოდი (Rank Selection)-პროპორციული არჩევის მეთოდი - ამ მეთოდის არსი მდგომარეობს იმაში, რომ "მშობლები" შესაძლოა გახდნენ ის ინდივიდები, რომელთა სარგებლიანობის ფუნქცია არ არის ნაკლები მთლიანი პოპულაციის ინდივიდების სარგებლიანობის ფუნქციის საშუალო მნიშვნელობაზე.

- რულეტკის მეთოდი (roulette-wheel selection) - გულისხმობს "რულეტკის" დატრიალებით მშობლების არჩევას. თუ ვირჩევთ ორ მშობელს, რულეტკა დატრიალდება ორჯერ, რაც შეეხება თვითონ სექტორებს მათი რაოდენობა უდრის ინდივიდების რაოდენობას პოპულაციაში, სექტორის სიდიდე კი დამოკიდებულია თვითონ ინდივიდების სარგებლიანობის ფუნქციის ფარდობით მთლიანი პოპულაციის ინდივიდების სარგებლიანობის ფუნქციის მნიშვნელობების ჯამთან.

- ტურნირის მეთოდი (tournament selection) - აქ ჯერ აირჩევა რამდენიმე ინდივიდი შემთხვევითი წესით და მერე მათ შორის საუკეთესო;

- ინბრიდინგი (Inbreeding selection)-მშობლების წყვილიდან პირველი ინდივიდი აირჩევა შემთხვევით, ხოლო მეორე პირველთან დიდი ალბათობით მასთან "მიახლოებული მონათესავე". მონათესავეში იგულისხმება: ჰემინგის მანძილი ამ ორ ინდივიდს შორის უნდა იყოს მინიმალური;

- აუთბრიდინგი (Outbreeding selection) - ინბრიდინგის საპირისპიროა და შესაჯვარებელ წყვილებად ირჩევს მაქსიმალურად "დაშორებულ" ინდივიდებს.

და სხვა.

4. **რეპროდუქცია და პოპულაციის შეცვლა** (Reproduction). რეპროდუქცია გულისხმობს შემდეგი თაობის პოპულაციის გენერაციას წინა თაობის ინდივიდებისაგან. შემდეგი თაობის გენერაცია ხდება გენეტიკური ოპერატორების დახმარებით. ეს ნიშნავს, რომ შემდეგი თაობის ერთ-ერთი ინდივიდი მიიღება წინა თაობის ერთ ან რამდენიმე ინდივიდს შორის კონკრეტული გენეტიკური ოპერაციის ჩატარებით. არსებობს რამდენიმე გენეტიკური ოპერატორი, ყველაზე ხშირად გამოიყენება შეჯვარება და მუტაცია.

შეჯვარება (Crossover) - ეს ოპერაცია გულისხმობს რამდენიმე, საზოგადოდ კი ორი ქრომოსომისაგან, რომელთაც ჰქვიათ მშობლები, ერთი ან რამდენიმე ახალი ქრომოსომის გაჩენას, რომელთაც ჰქვიათ შვილები (შთამომავლები). მარტივ შემთხვევაში შეჯვარება **გა**-ში რეალიზდება როგორც ბიოლოგიაში: ქრომოსომები იყოფიან შემთხვევით აღებულ წერტილში და ცვლიან ნაწილებს ერთმანეთს შორის.
შეჯვარების სახეები:

- ერთწერტილიანი შეჯვარება: თავიდან შემთხვევით, ან რაიმე წესით ფორმირდება გაყოფის წერტილი ორობითი სტრიქონის ორ მეზობელ ბიტს შორის, მშობლები იყოფიან ამ წერტილში ორ სეგმენტად და შემდეგ შესაბამისი სექტორები "ეწებებიან" სხვა მშობლის ნაწილს, ასე მიიღება ორი შთამომავალი.

- მრავალწერტილიანი შეჯვარება: აირჩევა ორი ან მეტი გახლეჩვის წერტილი და მშობლები ცვლიან სეგმენტებს ამ წერტილებს შორის.

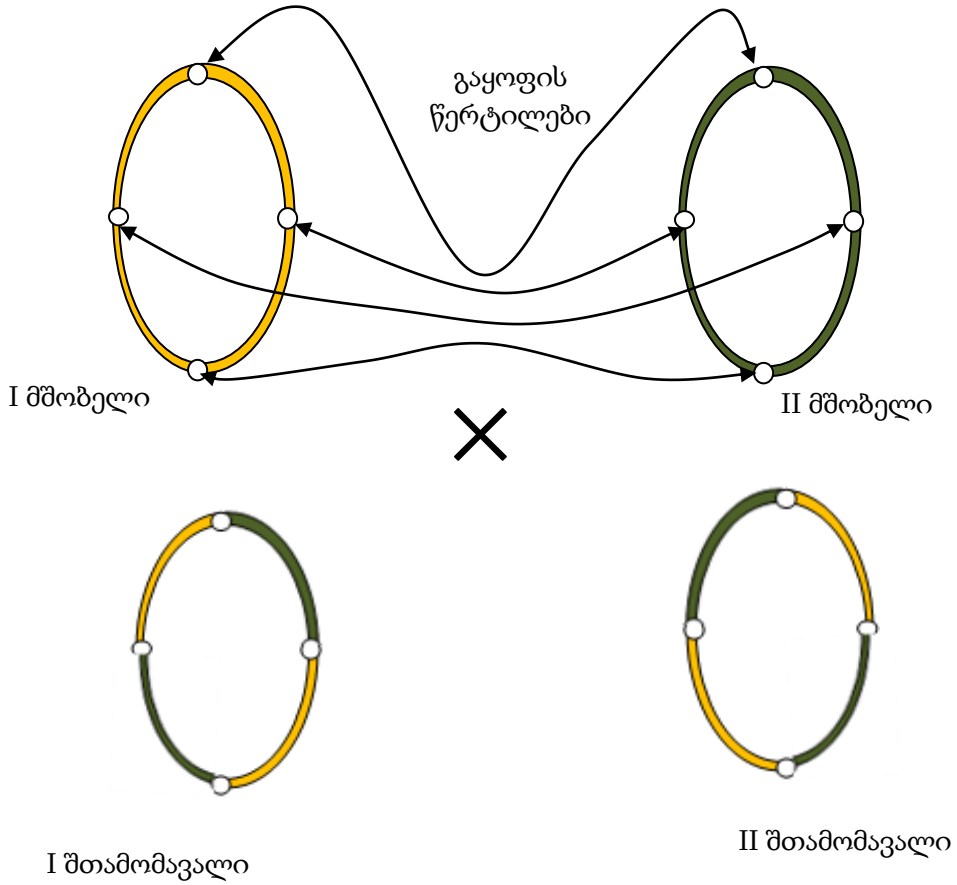
- თანაბარი შეჯვარება: პირველი შთამომავალი მემკვიდრეობით იღებს თითოეულ ბიტს რაიმე p_j ალბათობით პირველი მშობლიდან, ხოლო მეორე მშობლიდან კი $1 - p_j$ ალბათობით.

თვალსაჩინოებისათვის მოვიყვანოთ მრავალწერტილიანი კლასიკური შეჯვარების სქემა, რომელშიც ქრომოსომა წარმოდგენილია რგოლის სახით. ქრომოსომას ყოფენ სეგ-მენტებად, თუ ეს სქემა 4-წერტილიანია, მაშინ ის დაიყოფა 4 სეგმენტად (ქვემოთ იხ. ნახ. 2.1.1).

მუტაცია (Mutation)-მუტაცია გულისხმობს ქრომოსომების გარდაქმნას, შემთხვევით აღებული ერთი ან რამდენიმე ბიტის მნიშვნელობის ინვერტირების გზით. ანუ თუ ბიტის მნიშვნელობა იყო 1, ის გახდება 0-ის ტოლი და თუ იყო 0 გახდება 1. მუტაციის დანიშნულებაა პოპულაციაში ინდივიდების მრავალფეროვნების შენარჩუნება, რომ არ მოხდეს დაჩქარებული კრებადობა არასასურველი მინიმუმისკენ.
მუტაციის სახეებია:

- დამატება - გულისხმობს ყველა კოდირებულ გენში ყველა 1-იანის და 0-ების ინვერ-სიებს;
- ინვერსია - გულისხმობს ყველა გენის შემთხვევით არჩეულ ნაწილში ინვერსიას;

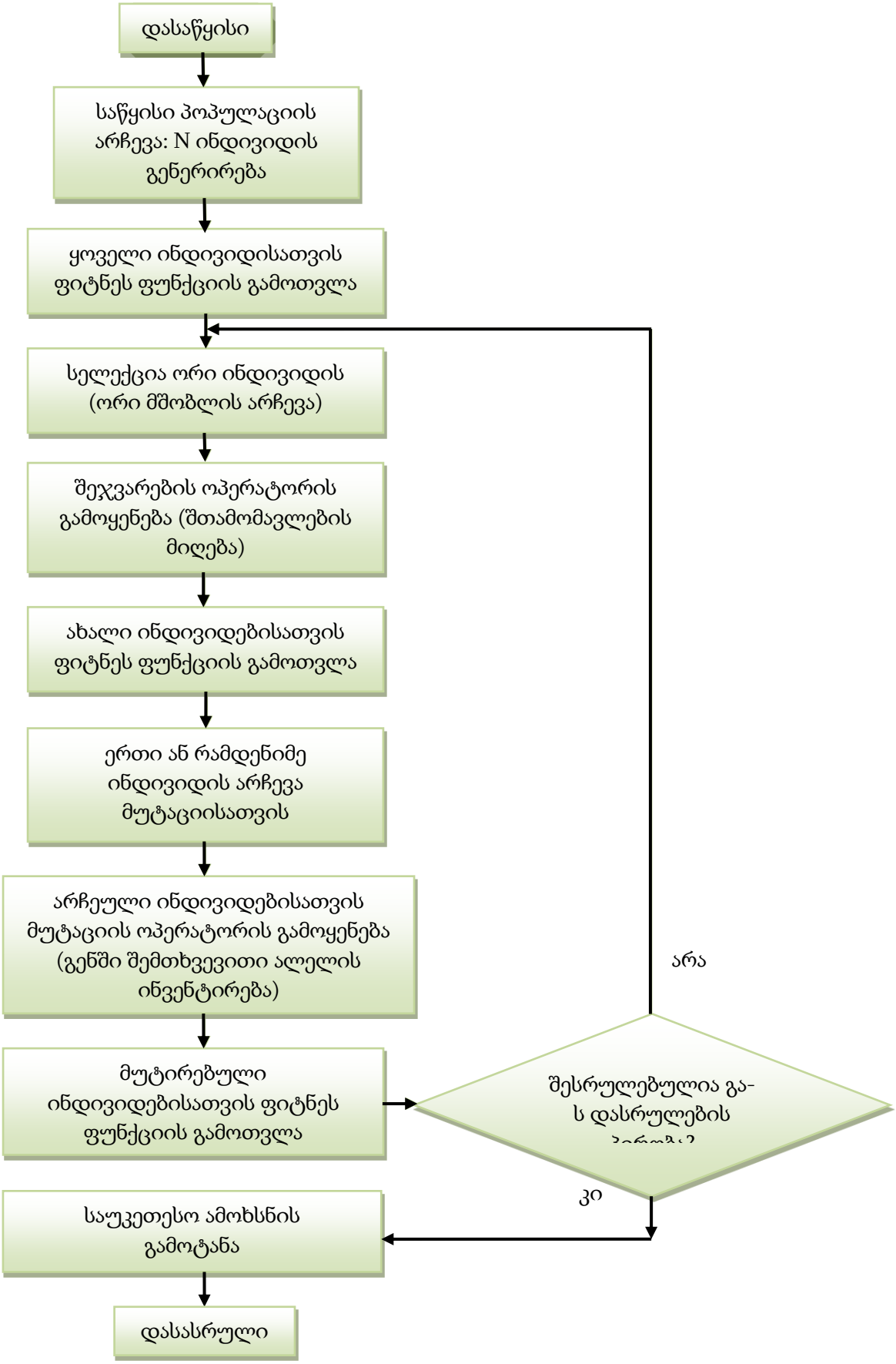
- ტრანსლოკაცია - გულისხმობს ყველა გენის შემთხვევით არჩეულ რამდენიმე ნაწილში ინვერსიას; და სხვა.



ნახ. 2.1.1. 4 წერტილიანი შეჯვარების მუშაობის სქემა

5. **ალგორითმის დასრულება** (Termination). ალგორითმი გრძელდება მანამ, სანამ "დასრულების პირობა"— არ შესრულდება, ეს პირობები შესაძლოა იყოს შემდეგი:
   - ნაპოვნი პასუხი აკმაყოფილებს წინასწარ არჩეულ კრიტერიუმს, "ახლოსაა" ოპტიმალურთან;
   - შესრულდა წინასწარ განსაზღვრული რეპროდუქციების რაოდენობა (კოდში ეს შეესაბამება იტერაციების რაოდენობას);
   - ახალ იტერაციებზე აღარ ხდება სარგებლიანობის ფუნქციის გაუმჯობესება;

ამგვარად, გენეტიკურ ალგორითმებში კარგი მახასიათებლები გადაეცემა თაობიდან თაობას. ნახ. 2.1.2.-ზე კი მოცემულია გენეტიკური ალგორითმის მუშაობის ბლოკ-სქემა.



ნახ. 2.1.2. გენეტიკური ალგორითმის მუშაობის სქემა

2.2. პირველი მოდიფიცირებული გენეტიკური ალგორითმი

ამ თავში ვიხილავთ მინიმიზაციის ამოცანის: $f(x) \to min, x \in R_n$ (2.1) ამოხსნას გენეტიკური ალგორითმის გამოყენებით (Панченко, 2007; Гладков , и др., 2004). გენეტიკური ალგორითმები კლასიკური ოპტიმიზაციის მეთოდებისაგან ძირითადად განსხვავდება შემდეგით: 1. გენეტიკური ალგორითმები მუშაობენ არა თვითონ პარამეტრებთან, არამედ მათ კოდირებულ წარმოდგენებთან; 2. გამოთვლების დასაწყებად **გა** არ თხოულობს საწყის მნიშვნელობებს, ვინაიდან ამოხსნის საწყის ეტაპზე გენერირდება პოპულაცია, რომელიც შეიცავს შემთხვევითი მიახლოებების ნაკრებს, რომელიც აკმაყოფილებს ამოცანის შეზღუდვებს; 3. **გა** მუშაობს არა ერთ ცალკეულ ამონახსნთან, არამედ ამონახსნთა სიმრავლესთან; 4. **გა** თავის გამოთვლებში იყენებს მხოლოდ მიზნის ფუნქციის მნიშვნელობას და არ მოითხოვს წარმოებულის გამოთვლას; 6. **გა**-ს მეტი მოქნილობა აქვს სხვადასხვა სახის ამოცანების ამოხსნისას.

ამ პარაგრაფში და **§2.3.**-ში შემოთავაზებული იქნება ორი ახალი მოდიფიცირებული გენეტიკური ალგორითმი (Ananiashvili, 2015). ორივე ალგორითმში (2.1) ამოცანის ამოსახსნელად ვიყენებთ კლასიკური გენეტიკური ალგორითმის ზოგად სქემას.

მოვიყვანოთ ჩვენს მიერ შემოთავაზებული მოდიფიცირებული ალგორითმების აღწერა ნაბიჯ-ნაბიჯ. ამ ალგორითმებში კლასიკური ალგორითმისაგან განსხვავებით, რომელშიც გამოიყენება ორობითი კოდირების სქემა, ჩვენ გამოვიყენეთ გრეის კოდირება, როგორც შეჯვარებისას, ასევე მუტაციის განხორციელებისას. ამავე დროს სელექციისათვის გამოვიყენეთ რულეტკის მეთოდი, რომლის განხორციელებისათვის გამოვიყენეთ კლასიკურისაგან განსხვავებული ფორმულები. მუტაცია განვახორციელეთ რამდენადმე განსხვავებული წესით, რომელიც აღწერილია ქვემოთ მოყვანილ ალგორითმებში (იხ. ალგორითმი **2.2.3.** და ალგორითმი **2.3.1.**).

2.2.1. საწყისი პოპულაციის არჩევა

მოვიყვანოთ საწყისი პოპულაციის არჩევის ალგორითმი ბიჯების სახით ჩაწერილი:

ალგორითმი 2.2.1. საწყისი პოპულაციის ფორმირება

ბიჯი 1. განვსაზღვროთ საწყისი პოპულაციის ზომა: P;

ბიჯი 2. ავირჩიოთ კოდირების სქემა: ორობითი კოდირება;

ბიჯი3. მონაცემის მოცემული $k0$ თანრიგის სიზუსტით კოდირებისათვის საჭირო რაოდენობის - k თანრიგების განსაზღვრა;

ბიჯი 4. მოცემულ დიაპაზონში $x = (x_1, x_2, ..., x_p)$ ათობითი რიცხვების არჩევა;

ბიჯი 5. ათობითი x მასივის კოდირება ბინარულ s მასივში;

ბიჯი 6. ბინარული s მასივის გადაყვანა გრეის კოდში;

მოვიყვანოთ ამ ბიჯების რეალიზაციის სიტყვიერი აღწერა.

ვთქვათ, საწყისი პოპულაციის ზომა არის P (P-ს განსაზღვრავს მომხმარებელი ამოცანის მიხედვით, ხშირად იღებენ: 10, 20, 25, 100-ს). ავირჩიოთ ბინარული კოდირების სქემა. მას შემდეგ, რაც კოდირების სქემას ავირჩევთ, ვიწყებთ საწყისი პოპულაციისათვის ელემენტების შერჩევას, მოცემულ ინტერვალზე. ვთქვათ, გვინდა რომ რიცხვების წარმოდგენის სიზუსტე უდრიდეს $k0$ ათობით თანრიგს მძიმის შემდეგ. განსახილველი ინტერვალი იყოს $[a, b]$. $lbit$-ით აღვნიშნოთ ბიტების მაქსიმალური რაოდენობა, რომელსაც გამოვიყენებთ ამ რიცხვების ჩასაწერად ორობით კოდში, შეგვიძლია გამოვიყენოთ 32 თანრიგიანი ან 16 თანრიგიანი კოდირება, ნაშრომში გამოყენებულია $lbit = 32$ თანრიგიანი კოდირება. კოდირებისათვის რეალურად საჭირო ბიტების რაოდენობა კი შეგვიძლია განვსაზღვროთ შემდეგი ფსევდოკოდის გამოყენებით:

$\delta = (b - a) \cdot 10^{k0}$, $k = k0$

$\boldsymbol{for}(\boldsymbol{int}\ i = k;\ i < lbit;\ i++)$

$\boldsymbol{if}((2^{k-1} < \delta)\&(\delta \le 2^k - 1))\ \boldsymbol{break};$

$\boldsymbol{else}\ k++;$

მაშასადამე, ჩვენი რიცხვები დაიკავებენ k ბიტს-პირველ თანრიგებს $lbit$ თანრიგიდან. ეს k ბიტი კი შეგვიძლია შევავსოთ ნებისმიერად 0-ების და 1-იანების მიდევრობით, ხოლო დანარჩენი უფროსი ბიტები უნდა შევავსოთ 0-ებით. მაგალითად, თუ $k = 18$ სქემატურად ეს შეგვიძლია ასე გამოვსახოთ:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	1	0	0	1

ამ რიცხვების მიღების ფსევდოკოდი კი შეიძლება ასეთი იყოს:

$p0 = 2^k / RAND\_MAX$

$\boldsymbol{unsigned\ long}\ popul[P];$

$\boldsymbol{for}(\boldsymbol{int}\ i = 0;\ i < S;\ i++)$

{

$\boldsymbol{if}(p0 == 0)\ popul[i] = rand()\%2^k;$

$\boldsymbol{else}\ popul[i] = rand()\%p0;$

}

$popul[i]$ მასივში მივიღებთ კოდირებულ P რაოდენობის ქრომოსომას.

პირიქითა გარდაქმნა ბიტების სტრიქონის ნამდვილ რიცხვებად გარდაქმნა, ანუ დეკოდირება კი შესაძლებელია შემდეგი ფსევდოკოდით:

$\boldsymbol{for}(\boldsymbol{int}\ i = 0;\ i < \mathrm{P};\ i++)$

{ $\boldsymbol{float}\ popul10[i] = a + popul[i] * ((b - a)/(2^k - 1));$ }

$popul10[i]$ მასივში მივიღებთ, ქრომოსომების ათობით მნიშვნელობებს.

ქვემოთ განვიხილავთ საილუსტრაციო მაგალითს. მანამდე კი ცოტა რამ კოდირების არჩევის შესახებ.

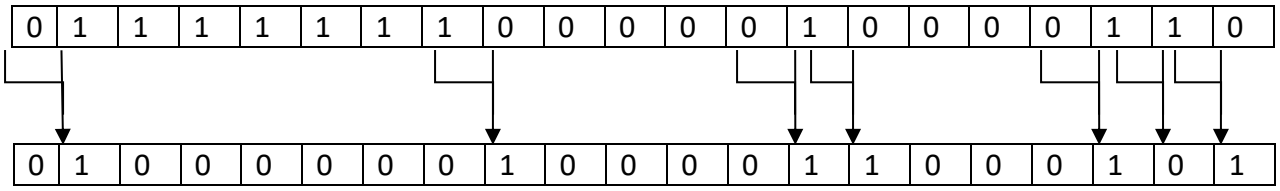
როცა რიცხვი პოზიციურ ორობით კოდშია ჩაწერილი, ასეთი სახით მისი დეკოდირების შემდეგ გარკვეულ პრობლემები ჩნდება. კერძოდ, ორი მეზობელი ათობითი რიცხვის კოდი ერთმანეთისაგან განსხვავდება დიდი რაოდენობის თანრიგებით. მაგალითად, რიცხვები 7 და 8 განსხვავდება 4 თანრიგით: 0111 და 1000. ამის გამო ხშირად ხდება ის რომ, გენეტიკური ალგორითმის მუშაობის პროცესი მთავრდება დიდი ცდომილებით ოპტიმალური ამონახსნიდან, ვინაიდან ოპტიმალურზე გადასვლას ამ სიტუაციაში დასჭირდება კიდევ ბევრი ცვლილება ორობით სტრიქონში, ანუ კიდევ ბევრი ბიჯის შესრულება. ამ სიტუაციის გამოსწორება შესაძლებელია რომელიმე სხვა ბინარული კოდირების გამოყენებით. ამიტომ უმჯობესია თუ გამოვიყენებთ გრეის კოდს. თვალსაჩინოებისათვის მოვიყვანოთ ცხრილი პოზიციური ბინარული კოდების, შესაბამისი გრეის კოდების და მათი ათობითი მნიშვნელობებისათვის:

ათობითი კოდისა და გრეის კოდის შესაბამისობა (4 ბიტიანი)					
ორობითი კოდირება		გრეის კოდის მიხედვით კოდირება	ორობითი კოდირება		გრეის კოდის მიხედვით კოდირება
ათობითი კოდი	ორობითი მნიშვნელობა	ორობითი მნიშვნელობა	ათობითი კოდი	ორობითი მნიშვნელობა	ორობითი მნიშვნელობა
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

ცხრილი 2.2.1. ათობითი რიცხვები და შესაბამისი გრეის კოდები(4 ბიტიანი)

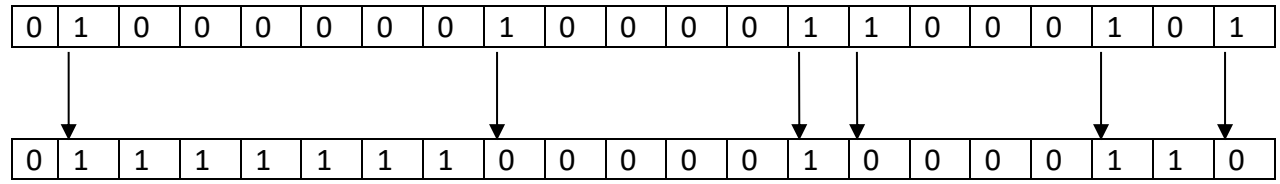
ცხრილიდან კარგად ჩანს რომ გრეის კოდში მეზობელ რიცხვებს შორის თითო ბიტშია განსხვავება. შესაბამისად გრეის კოდს უწოდებენ ერთ ბიჯიანს, ხოლო პირდაპირ ორობით კოდს მრავალბიჯიანს, რადგან ამ უკანასკნელის შემთხვევაში ერთი რიცხვიდან მის მეზობელ ათობით რიცხვზე გადასვლისას შიძლება შეიცვალოს რამდენიმე თანრიგი. ამ მოსაზრებების გათვალისწინებით ალგორითმის რეალიზაციის დროს ვიყენებ გრეის კოდში ჩაწერილი რიცხვებს.

იმისათვის რომ ორობითი კოდიდან რიცხვი გადავიყვანოთ გრეის კოდში ყოველი თანრიგი უნდა გავუტოლოთ ამ რიცხვის პირდაპირი ორობითი კოდის ამავე და მომდევნო თანრიგების ჯამს ორის მოდულით. ეს სქემატურად შემდეგნაირად შეგვიძლია გამოვსახოთ: xor ოპერაციით



ნახ. 2.2.1. პირდაპირი ორობითი კოდიდან გრეის კოდში გადაყვანის სქემა

გრეის კოდიდან პოზიციურ ორობით კოდში გადასაყვანად: ყოველი პოზიციური ორობითი თანრიგი უდრის შესაბამისი გრეის კოდიდან აღებული ამავე თანრიგისა და ყველა მაღალი თანრიგის მნიშვნელობების ჯამს ორის მოდულით. ეს სქემატურად შემდეგნაირად შეგვიძლია გამოვსახოთ:



ნახ. 2.2.2. გრეის კოდიდან პირდაპირ ორობით კოდში გადაყვანის სქემა

განვიხილოთ კონკრეტული მაგალითი: ვთქვათ, $x_j \in [a_j,\ b_j]$, და საჭირო სიზუსტე არის მძიმის შემდეგ 7 ნიშანი. ჯერ უნდა განვსაზღვროთ ამოცანაში შემავალი ცვლადების ჩასაწერად საჭირო ბიტების რაოდენობა. ბინარული სტრიქონის სიგრძე დამოკიდებულია იმაზე, თუ რა სიზუსტით გვინდა კოდირება. კოდირებისათვის საჭირო ბიტების რაოდენობას k_j -ს განვსაზღვრავთ უტოლობიდან:

$$2^{k_j-1} < (b_j - a_j) \cdot 10^7 \leq 2^{k_j} - 1 \quad (2.1)$$

ვთქვათ, (4.1) უტოლობიდან მივიღეთ რომ $k_j = 21$. ესე იგი, თითოეული ქრომოსომისათვის დაგვჭირდება 21 ორობითი თანრიგი, მასში 0-ები და 1-იანები შეგვიძლია ჩავწეროთ ნებისმიერად. მაგ.:

$$q_1 = (010000110001000111010)$$

ავირჩევთ საწყისი პოპულაციის ზომას, მაგ.: $P = 25$ და ამ წესით შევქმნით საწყის პოპულაციას. პოპულაციის თითოეულ ელემენტს ეწოდება ქრომოსომა. ამგვარად მივიღებთ შემთხვევით არჩეულ P ქრომოსომას: $q_1, q_2, \dots, q_P$.

უკუგადარქმნა ბიტური სტრიქონიდან x_j-ს ნამდვილი მნიშვნელობის მისაღებად შეგვიძლია შემდეგი ფორმულით:

$$x_j = a_j + D \cdot \frac{b_j - a_j}{2^{k_j} - 1} \qquad (2.2)$$

აქ D- ბინარულ q_j სტრიქონში კოდირებული ათობითი მნიშვნელობაა.

ჩვენს მაგალითში q_1 -ორობითი რიცხვის შესაბამისი ათობით მნიშვნელობა: $D = 549434$. x_1-ის მნიშვნელობას კი გამოვითვლით (2.2) ფორმულიდან:

$$x_j = 1.1 + 549434 \cdot \frac{2.9 - 1.1}{2^{21} - 1} = 1.571583$$

ასე გამოვითვლით ყოველი q_j , $j = 1,2, \dots P$-ს რეალურ-ათობით მნიშვნელობებს: $x_1, x_2, \dots, x_P$.

2.2.2. სელექცია

რადგან საწყისი პოპულაცია შევარჩიეთ, ახლა განვსაზღვროთ ფიტნეს (სარგებლიანობის) ფუნქცია და შემდეგ შეფასდება მიღებულ პოპულაციაში ინდივიდების სარგებლიანობა. შემდეგ ჩვენ გავსაზღვრავთ კრიტერიუმს, რომლის მიხედვითაც ავირჩევთ შესაჯვარებელ წყვილებს. ცხადია, სარგებლიანობის ფუნქციის მნიშვნელობა დამოკიდებულია მიზნის ფუნქციის იმ მნიშვნელობებზე, რომლებსაც ის გვაძლევს მიმდინარე პოპულაციის ინდივიდებისათვის. გარდა ამისა ყოველი ქრომოსომისათვის უნდა გამოვითვალოთ მათი შესაბამისი ალბათობები.

გამოვითვალოთ თითოეული ქრომოსომისათვის შესაბამისი მიზნის ფუნქციის მნიშვნელობები:

$$f(x_j), \qquad j = 1,2, \dots, P$$

აღვნიშნოთ:

$$fmin = min\{f(x_1), f(x_2), \dots, f(x_P)\} \qquad (2.3)$$

გამოვითვალოთ პოპულაციის შესაბამისობის ზოგადი ფუნქცია:

$$F = \sum_{i=1}^{P} (f(x_i) - fmin) \qquad (2.4)$$

ყოველი ქრომოსომისათვის გამოვითვალოთ ალბათობა:

$$p_j = \frac{f(x_j) - fmin}{F} , \ j = 1,2, \dots, P \qquad (2.5)$$

ასევე ყოველი $f(x_j)$, $j = 1,2, \dots, P$ -სათვის გამოვითვალოთ ალბათობები:

$$\tau_j = \sum_{i=1}^{j} p_i \ , \ j = 1,2, \dots, P \qquad (2.6)$$

არსებული პოპულაციიდან მომდევნო პოპულაციის მისაღებად ჯერ რაიმე წესით უნდა ავირჩიოთ ინდივიდები წინა პოპულაციიდან და შემდეგ მოვახდინოთ მათი შეჯვარება. შემდეგ უკვე არჩეული-მშობელი ინდივიდები ჩანაცვლდებიან შვილობილებით - შეჯვარების შედეგად მიღებული ინდივიდებით. არჩევანის გასაკეთებლად გამოვიყენოთ ე.წ. "რულეტკის" მეთოდი. რულეტკის მეთოდის გამოყენებით პოპულაციაში ინდივიდებს სხვა რიგით გადავალაგებთ. რიგითობის შერჩევის პროცესი კი დაიწყება რულეტკის "დატრიალებით", სულ დავატრიალებთ P-ჯერ. თითოეული დატრიალება გულისხმობს შემთხვევითი ნორმალიზებული რიცხვების აღებას $[0,1]$ ინტერვალიდან, ამასთან ყოველ ჯერზე აირჩევა ერთი ქრომოსომა შემდეგი ალგორითმით:

1. $[0,1]$ ინტერვალიდან ვიღებთ შემთხვევით რიცხვს, ვთქვათ ეს არის r.
2. თუ $r \leq \tau_1$, მაშინ ვირჩევთ პირველ ქრომოსომას: x_1-ს, წინააღმდეგ შემთხვევაში ავირჩევთ იმ x_i-ქრომოსომას, რომლისთვისაც: $\tau_{i-1} < r \leq \tau_i \ , \ 2 \leq i \leq P$.

ამ წესით ნაპოვნი ქრომოსომა დაიკავებს პირველ ადგილს პოპულაციაში. შემდეგ იგივე წესით ვიპოვით მეორე ქრომოსომას და ა. შ. P-ურის ჩათვლით.

ამის შემდეგ, რაც პოპულაციაში ინდივიდები გადავალაგეთ "რულეტკის" გამოყენებით, საჭიროა მიმდინარე პოპულაციიდან შესაჯვარებელი ორი მშობელი ინდივიდის ამორჩევა და შემდგომ "შეჯვარება". შეჯვარების ოპერატორს ქვემოთ განვიხილავთ. ამოვარჩიოთ ეს ინდივიდები შემდეგი წესით:

1. $[0,1]$ ინტერვალიდან ავიღოთ P რაოდენობის შემთხვევითი რიცხვი: $\varphi_1, \varphi_2, \dots, \varphi_P$.
2. ავირჩიოთ ორი მინიმალური ამ რიცხვებს შორის: $\varphi_{J_1}, \varphi_{J_2}$.

ალგორითმის შემდეგი ნაბიჯი იქნება სწორედ ამ $\varphi_{J_1}, \varphi_{J_2}$-ის შესაბამისი j_1 და j_2 ნომრიანი ინდივიდების შეჯვარება.

2.2.3. შეჯვარება

შემოთავაზებულ ალგორითმში გამოყენებულია შეჯვარების ერთწერტილიანი სქემა. როგორც უკვე აღვნიშნეთ ერთწერტილიანი შეჯვარებისას შემთხვევითი წესით აირჩევა გაყოფის წერტილი. ორობითში კოდირებული რიცხვებისათვის ეს წერტილი არის ადგილი - პოზიცია მეზობელ ბიტებს შორის. ორივე მშობელი (გენი) - სტრუქტურა იყოფა ორ სეგმენტად (ბიტების მიმდევრობად), შემდეგ განსხვავებული მშობლების შესაბამისი სეგმენტები (რიცხვების უმცროსი და უფროსი ბიტების მიმდევრობები ამ წერტილიდან-პოზიციიდან დაწყებული) ეწებებიან ერთმანეთს. მოვიყვანოთ შეჯვარების ალგორითმი ბიჯების მიხედვით:

ალგორითმი 2.2.2. შეჯვარება

ბიჯი 1. მოვახდინოთ j_1 და j_2 ნომრიანი ქრომოსომების დეკოდირება, ისინი საწყისი პოპულაციის არჩევის დროს გადავიყვანეთ გრეის კოდში. ახლა მათი მნიშვნელობები ბიტურად ჩავწეროთ vf და vm მასივებში.

ბიჯი 2. ავიღოთ შემთხვევითი რიცხვი: $\vartheta \in [1, lbit]$ ინტერვალიდან, სადაც lbit ქრომოსომების კოდირებისათვის საჭირო ბიტების რაოდენობაა.

ბიჯი 3. vf და vm მასივების 0-დან lbit-მდე ინდექსიან ელემენტებს შევუცვალოთ ადგილები;

ბიჯი 4. მოვახდინოთ "შეჯვარებული" vf და vm მასივების კოდირება კვლავ j_1 და j_2 ნომრიანი ქრომოსომების ადგილებში თავდაპირველად პოზიციურ ორობით კოდში. შემდეგ კი ორობითიდან კვლავ გადაგვყავს გრეის კოდში.

შეჯვარების დასრულების შემდეგ მიიღება ახალი პოპულაცია, რომელიც წინა პოპულაციისაგან იმით განსხვავდება, რომ მასში მშობელი ქრომოსომები ჩანაცვლდებიან შვილი ქრომოსომებით, ამით ახალი პოპულაციის ფორმირება არ სრულდება. ქრომოსომებიდან ერთმა ან რამდენიმემ უნდა განიცადოს მუტაცია. მუტაციის ოპერატორს ქვემოთ განვიხილავთ.

მოვიყვანოთ ალგორითმი 2.2.2-ის ბიჯების რეალიზაციის სიტყვიერი აღწერა.

გამოყენებულია ერთწერტილიანი შეჯვარების მეთოდი. $[0, 1]$ ინტერვალიდან ავიღოთ შემთხვევითი რიცხვები: $\varphi_1, \varphi_2, \dots, \varphi_P$. მათ შორის ამოვარჩიოთ ორი მინიმალური მნიშვნელობის მქონე. ვთქვათ, ასეთებია: φ_4 და φ_6. ანუ შესაჯვარებელი ქრომოსომებია მე-4 და მე-6 ქრომოსომა. მათ უწოდებენ მშობელებს. ვინაიდან, შეჯვარების დროს ჩვენ დაგვჭირდება ბიტური ოპერაციების შესრულება, რომელთა რეალიზება მოხერხებულია C++ ენის ბიტური ოპერატორებით, სიტყვიერი აღწერის ნაცვლად უმჯობესია მათი პროგრამულ ფრაგმენტების სახით შემოთავაზება. ბიტური ოპერაციები ცნობილია და მათი აღქმა არ წარმოადგენს სირთულეს. იმისათვის რომ ინდივიდის(ქრომოსომის) ცვლილება არ იყოს ძალიან დიდი, ორობითში კოდირებულ x_4 და x_6 ქრომოსომების მიმართ გამოვიყენოთ გრეის კოდირება. კონკრეტულად კი, ჩვენი ქრომოსომები კოდირებულია unsigned long ტიპის მასივში, ამ მასივის ნებისმიერი ელემენტის, გრეის კოდში გადაყვანა შესაძლოა უზრუნველვყოთ შემდეგი ბრძანებით:

```
gf=xch[numb]^(xch[numb]>>1);
```

სადაც gf ცვლადია unsigned long ტიპის, ხოლო xch-ქრომოსომების ასევე unsigned long ტიპის მასივია. ხოლო numb კოდირებისათვის არჩეული ქრომოსომის ნომერია, ჩვენს მაგალითში ის შეიძლება იყოს 4 ან 6. შემდეგ მოვახდინოთ gf-ის დეკოდირება: მისი ბიტების მნიშვნელობები ჩავწეროთ რაიმე vf მასივში. ეს შეგვიძლია განვახორციელოთ პროგრამის შემდეგი ფრაგმენტით:

```
for (int i=0; i<bit; i++)
    vf[i]=0;
  unsigned long c0, c; int c1;
    for (int j=0; j<bit; j++){
        c0=1;
        c=c0<<(bit-j-1);
        c1=gf & c;
        if (c1!=0) vf[j]=1;       }
```

სადაც bit ქრომოსომის კოდირებისათვის საჭირო ბიტების რაოდენობაა, ჩვენს შემთხვევაში bit=21. (2.1)-დან გამოთვლის შესაბამისად თუ ჩვენ კოდირებისათვის გვჭირდება მხოლოდ 21 თანრიგი, იმისათვის რომ, x_j-ები დარჩნენ $[a_j, b_j]$ დიაპაზონში, თითოეული ქრომოსომის ბინარულ წარმოდგენაში 21-დან დაწყებული 32-მდე მაღალ თანრიგებში უნდა გვეწეროს 0-ები. ამ წესით ჩაწერილ x_4 და x_6 ქრომოსომებს ჯერ გადავიყვანთ გრეის კოდში, შემდეგ კი მოვახდენთ დეკოდირებას. ვთქვათ, x_4-ის და x_6 -ის შესაბამისი მასივებია vf დავm :

vf: 0000 0000 0001 0011 0111 1011 0100 1000

vm: 0000 0000 0000 1100 0110 0101 1101 0000

ავიღოთ შემთხვევითი რიცხვი $\vartheta \in [1,\ 21]$, სწორედ ეს ϑ იქნება ის პოზიცია, რომლის მიმართაც მოხდება მე-4 და მე-6 ქრომოსომის შეჯვარება. ვთქვათ $\vartheta = 11$, მაშინ vf დავm ასე შეიცვლება:

vf: 0000 0000 0001 0011 0111 1101 1101 0000

vm: 0000 0000 0000 1100 0110 0011 0100 1000

პირიქით დეკოდირებისათვის გამოვიყენოთ პროგრამის შემდეგი ფრაგმენტი:

```
gf=0;
for(int p=1; p<=bit; p++){
        if (vf[p-1]==1) c=1;
        else c=0;
        c=c<<(bit-p); gf=gf|c;       }
```

გრეის კოდში გადასაყვანად, კი:

```
for (xch[numbf] = 0; gf; gf >>= 1)
    xch[numbf] ^= gf;
```

2.2.4. მუტაცია

შეჯვარების შემდეგ ხორციელდება მუტაციის ბიჯი. მოვიყვანოთ მუტაციის ალგორითმი ორი ცვლადის ფუნქციისათვის. მუტაციის დროს იცვლება ერთი ან რამდენიმე გენი.

ალგორითმი 2.2.3. მუტაცია

ვთქვათ, პოპულაციის ზომაა - P, k_i-პოპულაციაში x_i-ქრომოსომის კოდირებისათვის საჭირო ბიტების რაოდენობა,

ბიჯი 1. გამოვთვალოთ ჯამი: $K = \sum_{i=1}^{S} k_i$; ვიპოვოთ $K \cdot S$-ის 1%, ანუ: $\tau = \lfloor 0.01 \cdot K \cdot P \rfloor$ (სადაც $\lfloor 0.01 \cdot K \cdot P \rfloor$ აღნიშნავს $0.01 \cdot K \cdot P$ –ს მთელ ნაწილს);

ბიჯი 2. $[1,\ K \cdot P]$ ინტერვალიდან ავიღოთ შემთხვევითი ნორმალიზებული რიცხვები: $r_1,\ r_2, \dots, r_\tau$.

ბიჯი 3. ავიღოთ: $i = 1$;

ბიჯი 4. ვიპოვოთ ისეთი j, რომლისთვისაც:

$$\frac{rand()}{100} \cdot k_j = max\left(\frac{rand()}{100} \cdot k_i\right), \qquad i = 1,2, \dots, P$$

სწორედ ეს j-ური ქრომოსომა განიცდის მუტაციას.

ბიჯი 5. გამოვთვალოთ: $bit = abs\left(k_j - \lfloor r_i/(S \cdot k_i) \rfloor\right)$, -ეს არის იმ ბიტის ნომერი, რომელმაც უნდა განიცადოს მუტაცია;

ბიჯი 6. x_i^j გენოტიპში bit ნომრიან ბიტში ჩავწეროთ 0, თუ მანამდე იქ ეწერა 1 და ჩავწეროთ 1, თუ ეწერა 0. (იხ. ალგორითმი);

ბიჯი 7. ავიღოთ: $i = i + 1$; თუ $i \leq \tau$, გადავიდეთ ბიჯი 4-ზე, წინააღმდეგ შემთხვევაში დასასრული.

ალგორითმის საილუსტრაციოდ განვიხილოთ ჩვენი მაგალითი:

მუტირებული გენების რაოდენობა დავითვალოთ შემდეგნაირად: პოპულაციაში ბიტების რაოდენობა გავამრავლოთ პოპულაციის ზომაზე: 21**P*=21*25=525, და მუტაციას დავუქვემდებაროთ ამ რიცხვის ერთი პროცენტი(რომელიც დამრგვალებით უდრის 5-ს), ანუ 5 გენი. ავიღოთ 5 შემთხვევითი რიცხვი $[1, 525]$ ინტერვალიდან. ვთქვათ, ეს რიცხვებია: r_1, r_2, r_3, r_4, r_5. სწორედ ამ ნომრიანი გენოტიპები, განიცდიან მუტაციას. მაგ.: თუ $r_1 = 65$, მაშინ მეოთხე (65/21=3) ქრომოსომაში მეორე (65-3*21=2) ბიტში თუ წერია 0, უნდა შევცვალოთ 1-იანით და პირიქით, თუ 1-იანია 0-ით. მუტაციამდე აქ კვლავ ვიყენებთ ქრომოსომის გრეის კოდში ჩაწერას, ისევე როგორც ეს გავაკეთეთ შეჯვარების დროს.

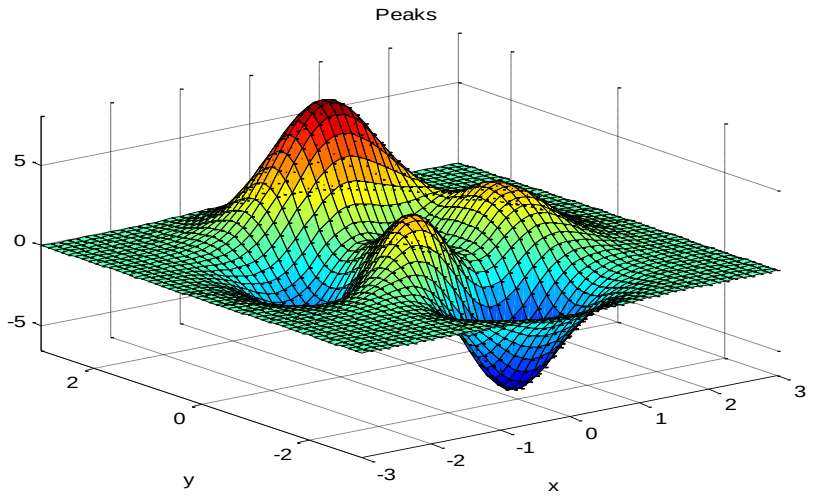
მუტაციით მთავრდება ჩვენი ალგორითმის პირველი იტერაცია. ამ ბიჯებს (სელექცია, შეჯვარება, მუტაცია) ვიმეორებთ იტერაციების ამოწურვამდე. ყოველი იტერაციის შემდეგ ვირჩევთ მიზნის ფუნქციის საუკეთესო მნიშვნელობას, ვადარებთ წინა და ახალი იტერაციის

ბოლოს მიღებულ საუკეთესო მნიშვნელობებს და ვიმახსოვრებთ უკეთესს. იტერაციების ამოწურვისას სწორედ ეს საუკეთესო იქნება ჩვენი ამოცანის ამონახსნი.

განვიხილოთ შემდეგი ფუნქციის მაგალითი, (Peaks function):

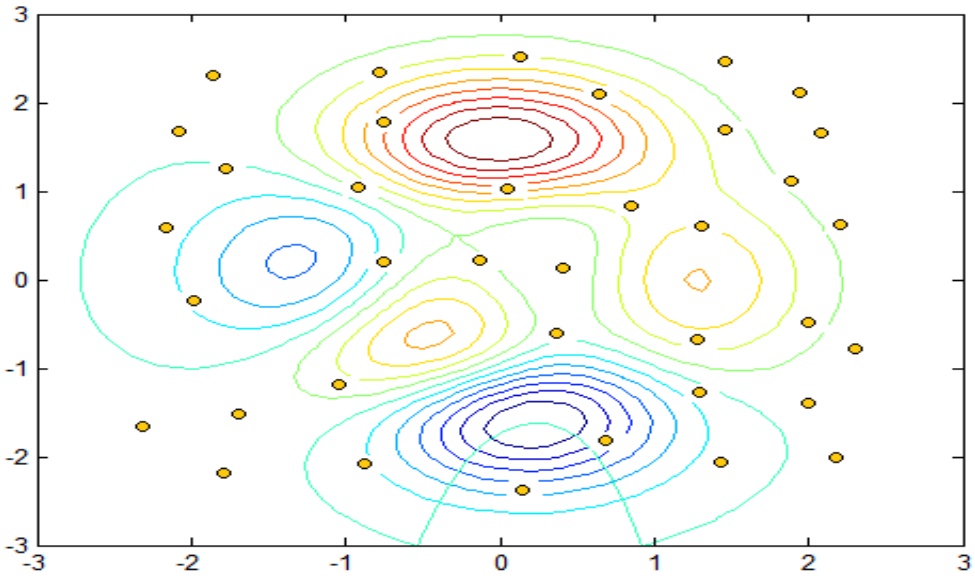
$$z = 3(1-x)^2\, e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{-x^2-y^2} - \frac{1}{3}\, e^{-(x+1)^2-y^2}$$

პიკების ფუნქციის გრაფიკი:



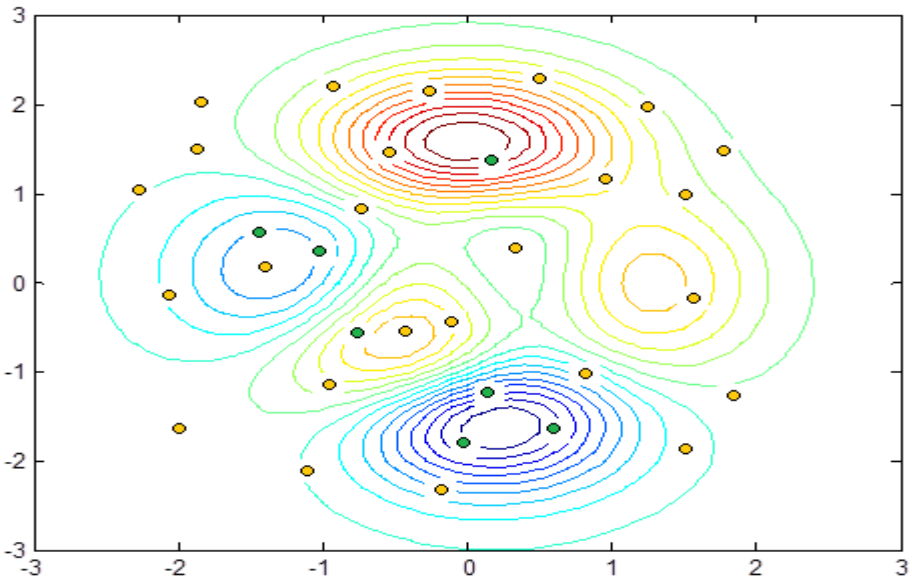
ნახ. 2.2.3. Peaks ფუნქციის გრაფიკი

ჩვენი ალგორითმის რეალიზაციის პირველი იტერაციაზე მიღებული შედეგის ერთ-ერთი ვარიანტი შეიძლება ასეთი იყოს:



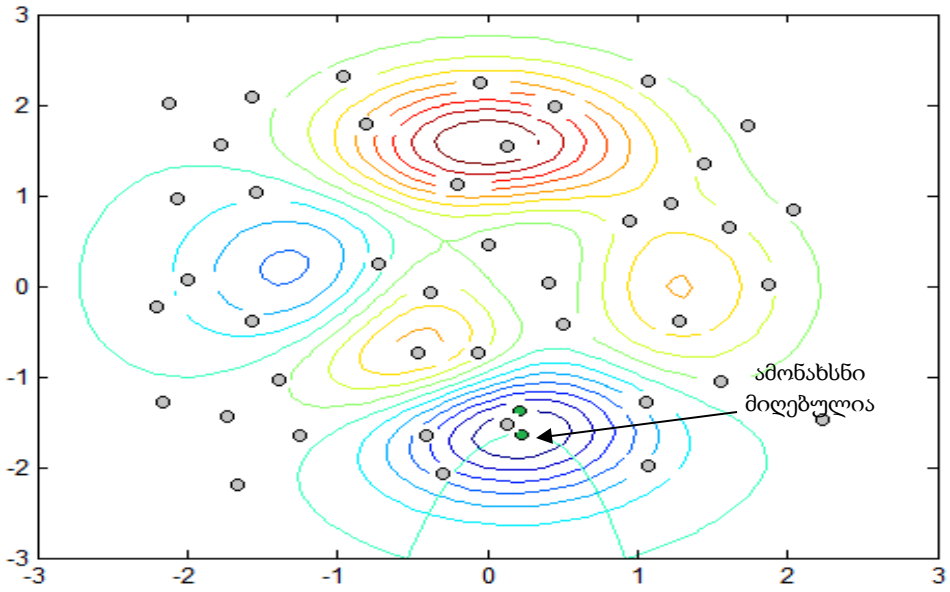
ნახ. 2.2.4. პირველი იტერაციის შედეგი

მომდევნო იტერაციებზე უკვე გვექნება პოპულაციის რამდენიმე მინიმალურთან მიახლოებული მნიშვნელობა, ამასთან ზოგი ლოკალურ მინიმუმთან იქნება ახლოს ზოგი კი გლობალურთან:



ნახ. 2.2.5. 43-ე იტერაციის შედეგი

კოდის მუშაობის დასრულებისათვის-ბოლო იტერაციაზე კი გვექნება ასეთი სურათი:



ნახ. 2.2.6. 278-ე იტერაციის შედეგი

2.2.5. ტესტირების შედეგები

ალორითმი სტაბილურად გვაძლევს კარგ მიახლოებებს სატესტო ამოცანებზე. სისწრაფე კი სრულიად შეესაბამება გენეტიკური ალგორითმებისთვის მიღებულ (არამკაცრ) სტანდარტებს.

ცხრილი 2.2.2. მოდიფიცირებული გა-ის გამოთვლის შედეგები სატესტო (ორი ცვლადის) ფუნქციებისათვის

ფუნქცია	ფუნქციის მნიშვნელობა	x_1	x_2	თვლის დრო(წმ)
იაზონის (Eason) ფუნქცია	0,98655	3.07515	3.07164	0,055
რასტრიგინის (Rsatrigin) ფუნქცია	0.11925	0.01203	0.02139	3.829
ექვსკუზიანი შავი აქლემის (Six-hump camel black) ფუნქცია	-1.0306	-0.09898	0.72261	3.781
სამკუზიანი შავი აქლემის (Three-hump camel black) ფუნქცია	-5.0793e-004	0.01131	-0.02251	272.08

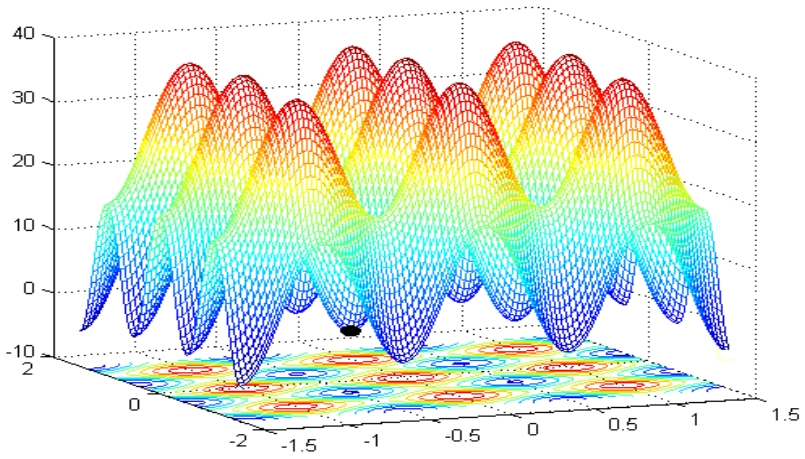
როზენბროკის (Rozenbrok) ფუნქცია	-0.00111234	1.02942	1.05813	12.45
შეფერის (Shaffer) ფუნქცია	0.011215	0.01252	0.10531	58.29
Bird's ფუნქცია	106.6743 106.2419	-1.5732 4.7540	-3.1542 3.1851	15.765 32.79
პიკების (Peaks) ფუნქცია	-6.0261	0.1411	-1.82212	8.586

ცხრილი 2.2.3. მოდიფიცირებული **გა**-ს გამოთვლის შედეგები სატესტო (ოთხი ცვლადის) ფუნქციებისათვის

ფუნქცია	ფუნქციის მნიშვნელობა	x_1	x_2	x_3	x_4	თვლის დრო(წმ)
ვუდის (Wood) ფუნქცია	-0.0133553	1.00307	1.003102	1.003874	1.01732	78.12
პაუელის (Powell) ფუნქცია	0.027593	-0.170792	0.031448	0.021555	0.280749	133.01

ტესტირება ჩატარდა სტანდარტული სიმძლავრის პერსონალურ კომპიუტერზე მონაცემებით: Intel(R) Pentium(R) Dual CPU E2200 2.20 GHz, 2.00 GB of RAM.

ნახ. 2.2.8-ზე მუქი შავი წერტილით ნაჩვენებია ჩვენი ალგორითმით მიღებული გლობალური მინიმუმის წერტილი, რასტრიგინის ფუნქციის მაგალითზე.



ნახ. 2.2.7. რასტრიგინის ფუნქციის გრაფიკი

2.3. მეორე მოდიფიცირებული გენეტიკური ალგორითმი

მოვიყვანოთ კიდევ ერთი მოდიფიცირებული გენეტიკური ალგორითმი, სქემატურად **გა**-ს ალგორითმი მოცემულია ნახ. 2.2.-ზე. ამ ალგორითმში მონაცემების ჩასაწერად კვლავ გამოყენებული იქნება ორობითი კოდირების სქემა. რომელიც ზემოთ §2.2.-შია აღწერილი.

საწყისი პოპულაციის ფორმირებაც ისეთივეა როგორც ზემოთ აღვწერეთ 2.2.1-ში (იხ. ალგორითმი 2.2.1)

მოვიყვანოთ დანარჩენი ბიჯების აღწერა მეორე ალგორითმისათვის.

2.3.1. სელექცია ინბრიდინგის მეთოდის გამოყენებით

სელექცია გულისხმობს იმ კრიტერიუმების განსაზღვრას, რომლის მიხედვითაც ავირჩევთ შესაჯვარებელ წყვილებს. ნაშრომში შემოთავაზებულია ინბრიდინგის მეთოდი, ანუ ერთ-ერთ მშობელს ვირჩევთ შემთვევითი წესით, ხოლო მეორეს ავირჩევთ მაქსიმალურად "მიახლოებულს" უკვე არჩეულ ინდივიდთან. სიახლოვეს კი განვსაზღვრავთ ჰემინგის მანძილით. ჰემინგის მანძილი - ეს არის მოცემული ორი s^r და s^p ინდივიდის შესაბამის ორობით სტრიქონში განსხავევბული ბიტების რაოდენობა:

$$d_h(s^r, s^p) = \sum_{i=1}^{n} |s^r - s^p|$$

ჩავთვალოთ რომ ორი s^r და s^p ინდივიდი ახლოსაა თუ

$$d_h(s^r, s^p) = \min\{d_h(s^r, s^p), \forall r = 1,2, \dots S, p = 1,2, \dots, S, r \neq p\}$$

მას შემდეგ, რაც მიმდინარე პოპულაციიდან ინბრინდინგის წესით არჩეულია შესაჯვარებელი ორი მშობელი - ინდივიდი გამოვიყენე ერთწერტილიანი შეჯვარება. იხ. შეჯვარების ალგორითმი 2.2.2.

შეჯვარების დასრულების შემდეგ მიიღება ახალი პოპულაცია, რომელიც წინა პოპულაციისაგან იმით განსხვავდება რომ მასში მშობელი ქრომოსომები ჩანაცვლდებიან შვილი ქრომოსომებით, ამით ახალი პოპულაციის ფორმირება არ სრულდება. ქრომოსომებიდან ერთმა ან რამდენიმემ უნდა განიცადოს მუტაცია. აღვწეროთ მუტაციის განსახორციელებელი ალგორითმი.

2.3.2. მუტაცია გრეის კოდირების გამოყენებით

ვთქვათ, პოპულაციის ზომაა - S, k_1 და k_2 -პოპულაციაში (x, y)-ქრომოსომის კოდირებისათვის საჭირო ბიტების რაოდენობებია, i-ური ქრომოსომა კი კოდირებულია xch და ych მასივებში, მოვიყვანოთ მუტაციის ალგორითმი ორი ცვლადის ფუნქციისათვის ბიჯების სახით:

(შენიშვნა: აქაც და ქვემოთაც ალგორითმებში მასივის ინდექსები ჩაწერილია ფრჩხილებში)

ალგორითმი 2.3.1. მუტაცია

ბიჯი 1. დავაგენერიროთ რაიმე შემთხვევითი რიცხვი: $rn \in [1, S]$ შუალედიდან, სწორედ rn ნომრიანი ქრომოსომა დაექვემდებარება მუტაციას;

ბიჯი 2. დავაგენერიროთ რაიმე შემთხვევითი რიცხვი: $r \in [0, 100]$ შუალედიდან, თუ $r < 50$, მაშინ მუტაციას დავუქვემდებაროთ $xch[rn]$ გენი. თუ $r \geq 50$, მაშინ მუტაციას დავუქვემდებაროთ $ych[rn]$ გენი.

(**შენიშვნა:** სხვა რაოდენობის ცვლადების შემთხვევაში ანალოგიური წესით განვსაზღვრავთ თუ რომელი გენმა უნდა განიცადოს მუტაცია: პროცენტულად r-ის მნიშვნელობის შესაბამისად. მაგ. ოთხი ცვლადის შემთხვევაში (x_1, x_2, x_3, x_4) შეგვიძლია ავირჩიოთ: თუ $r \leq 25$, ავირჩიოთ x_1, თუ $25 < r \leq 50$, ავირჩიოთ x_2, თუ $50 < r \leq 75$, ავირჩიოთ x_3, თუ $75 < r \leq 100$, ავირჩიოთ x_4)

ბიჯი 3. იმის მიხედვით თუ რომელი გენი განიცდის მუტაციას დავაგენერიროთ შემთხვევითი რიცხვი: $rk \in (0, k_1)$ ან $rk \in (0, k_2)$.

ბიჯი 3. გადავიყვანოთ არჩეული: $xch[rn]$ (ან $ych[rn]$) გენი გრეის კოდში, ვთქვათ ეს არის g;

ბიჯი 4. მოვახდინოთ g-ს დეკოდირება ორობით v ვექტორში;

ბიჯი 5. მოვახდინოთ $v[bit - rk - 1]$-ს ინვენტირება, ჩავწეროთ მასში 0, თუ მანამდე იქ ეწერა 1 და ჩავწეროთ 1, თუ ეწერა 0.

ბიჯი 6. მოვახდინოთ ორობითი v ვექტორის კოდირება $xch[rn]$ ან $ych[rn]$ გენში, იმის შესაბამისად, რომელმა გენმაც განიცადა მუტაცია.

მუტაციის შემდეგ ახალი პოპულაცია მიღებულია. ახლა უნდა შევამოწმოთ **გა**-ს დასრულების პირობა. თუ პირობა შესრულებულია, მაშინ მიღებულ პოპულაციაში საუკეთესო ინდივიდი იქნება ჩვენი ამოცანის ამონახსნი, წინააღმდეგ შემთხვევაში ალგორითმის ბიჯებს: სელექცია, შეჯვარება, მუტაციას ვიმეორებთ თავიდან გა-ს დასრულების პირობის შესრულებამდე. შემოთავაზებულ ნაშრომში გა-ს დასრულების პირობად აღებულია იტერაციების წინასწარ განსაზღვრული რაოდენობა.

ყოველი იტერაციის შემდეგ ვირჩევთ მიზნის ფუნქციის საუკეთესო მნიშვნელობას, ვადარებთ წინა და ახალი იტერაციის ბოლოს მიღებულ საუკეთესო მნიშვნელობებს და

ვიმახსოვრებთ უკეთესს. იტერაციების ამოწურვისას სწორედ ეს საუკეთესო იქნება ჩვენი ამოცანის ამონახსნი. ალგორითმი აპრობირებულია სატესტო ფუნქციებზე, ის კარგ შედეგებს იძლევა.

2.3.3. შედეგები

სატესტო ამოცანებზე ალგორითმის აპრობაციის შედეგები მოყვანილია ქვემოთ ცხრილებში.

ცხრილი 2.3.1. მეორე მოდიფიცირებული **გა**-ს გამოთვლის შედეგები სატესტო (ორი ცვლადის) ფუნქციებისათვის

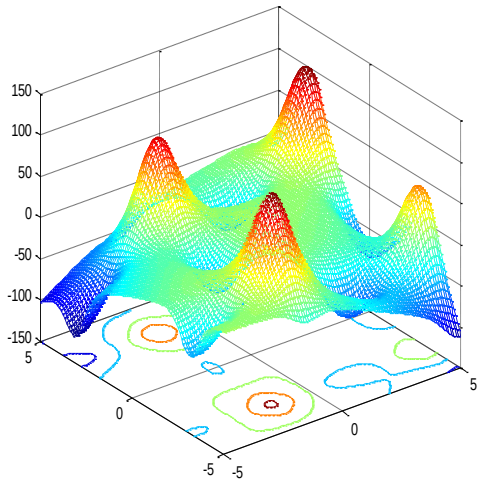
ფუნქციია	ფუნქციის მნიშვნელობა	x_1	x_2	თვლის დრო(წმ)	იტერაციების რაოდენობა
იაზონის (Eason)	0,999876	3.14934	3.13864	75,086	1822
რასტრიგინის (Rsatrigin)	-0,0082	-0.00598	0.002391	66,31	907
ექვსკუზიანი შავი აქლემი (Six-hump camel black function)	1.03142	0.09688475	-0.711739	90,007	1075
სამკუზიანი შავი აქლემი (Three-hump camel black function)	-0.00091212	-0.019783	-0.005187	62.865	3053
როზენბროკის (Rozenbrok)	0.00106851	1.011310	1.019681	69.251	121
შეფერის (Shaffer)	-3.2167e-005	-0.0029778	0.0048542	105.791	2859
ბერდის (Bird)	106.004 106.682	-1.57427 4.69286	-3.05588 3.17597	63.773 89.744	1299 608
მწვერვალები (Peaks)	-6.54949065	0.2411	-1.6212	119.88	1489

ცხრილი 2.3.2. მეორე მოდიფიცირებული **გა**-ს გამოთვლის შედეგები სატესტო (ოთხი ცვლადის) ფუნქციებისათვის

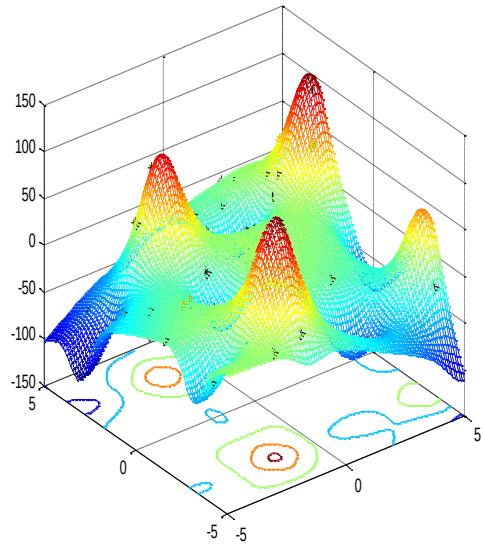
ფუნქცია	ფუნქციის მნიშვნელობა	x_1	x_2	x_3	x_4	თვლის დრო(წმ)	იტერაციების რაოდენობა
ვუდის (Wood)	-0.00604	0.99703	1.001102	1.002308	1.001232	124.86	2145
პაუელის (Powell)	-0.012726	0.12226	-0.003944	-0.00465	-0.02217	103.64	2969

მოვიყვანოთ რეალიზების ერთი საილუსტრაციო მაგალითი: ავაგოთ Bird ფუნქციის გრაფიკი და ვაჩვენოთ იტერაციული პროცესის მიმდინარეობა.

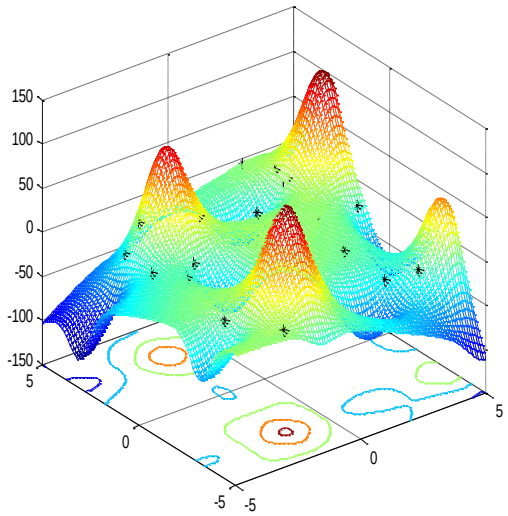
$$f(x,y) = -(x-y)^2 - e^{(1-\sin x)^2} \cdot \cos y - e^{(1-\sin y)^2} \cdot \cos x$$



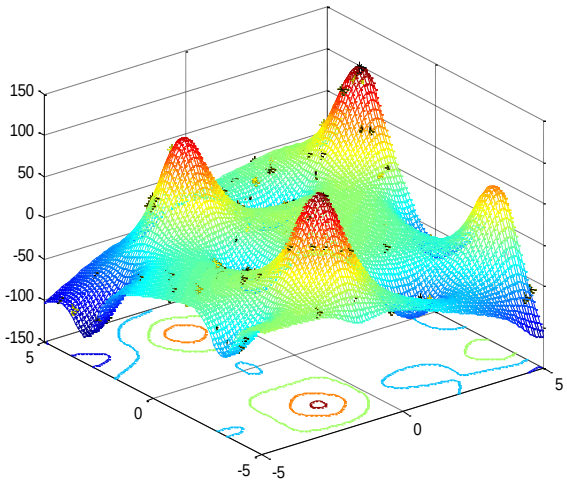
ნახ. 2.3.1. Bird ფუნქცია



ნახ. 2.3.2. 40-ე იტერაციაზე მიღებული მნიშვნელობები

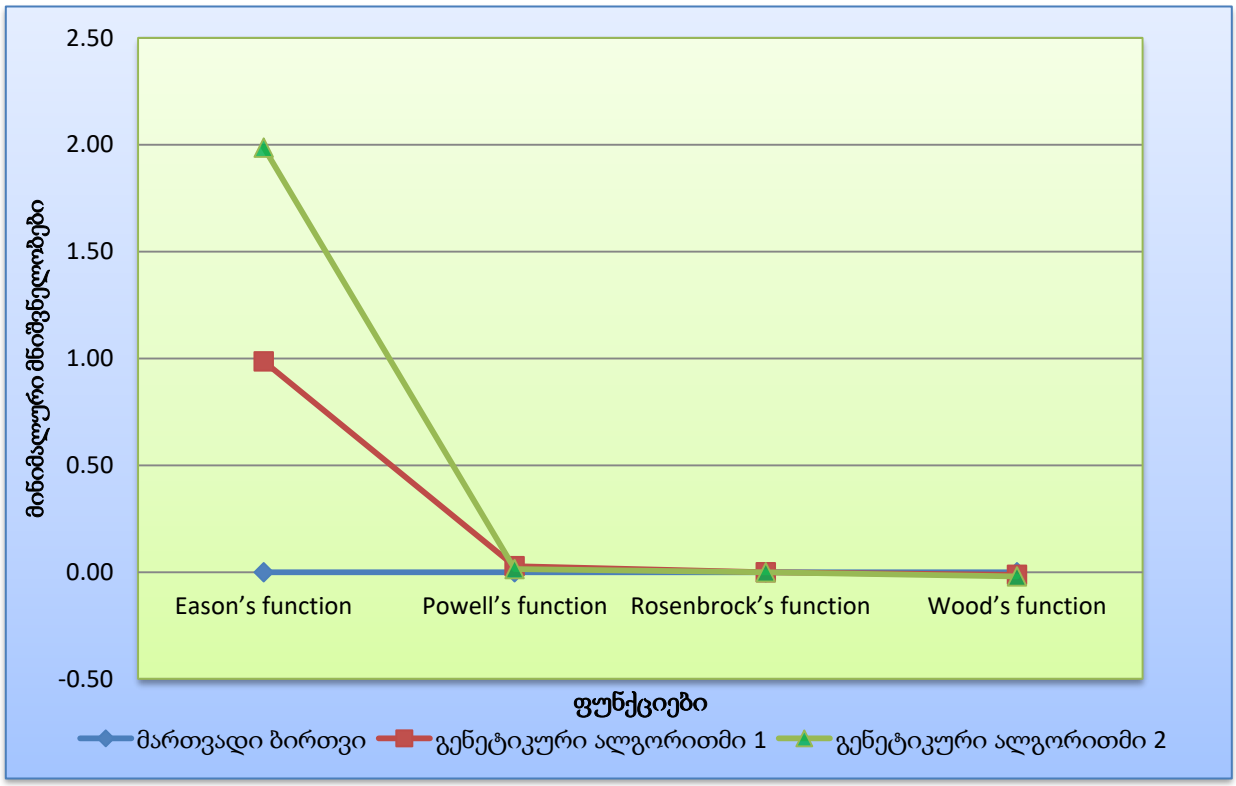


ნახ. 2.3.3. 60-ე იტერაციაზე მიღებული მნიშვნელობები



ნახ. 2.3.4. 450-ე იტერაციაზე მიღებული მნიშვნელობები

როგორც მოცემულია ცხრილი 2.3.1-ში, ოპტიმალური მნიშვნელობა მიღებულია 608-ე იტერაციაზე. შევადაროთ მართვადი მძიმე ბირთვის ალგორითმით და გენეტიკური ალგორითმებით მიღებული შედეგები:



ნახ. 2.3.5. მართვადი მძიმე ბირთვის ალგორითმით და გენეტიკური ალგორითმებით მიღებული შედეგების შედარებითი დიაგრამა.

დიაგრამიდან ჩანს, რომ გენეტიკურმა ალგორითმებმა უარესი შედეგები მოგვცა Eason-ის ფუნქციის შემთხვევაში, სხვაგან სამივე მეთოდით დაახლოებით ერთნაირი შედეგი მიიღება.

თავი 3. უმცირესი დაყოფისა და დაფარვის ამოცანების ამოხსნის შესახებ

3.1. პრობლემის შესახებ

ოპტიმიზაციის ამოცანების ნაწილი მიეკუთვნება კომბინატორულ ამოცანებს. ამ ამოცანების ამოხსნის ყველა არსებული ალგორითმების ბირთვს წარმოადგენს ამონახსნების სრული ან ნაწილობრივი გადარჩევა. გადარჩევის სტრატეგია სხვადასხვა ალგორითმში ხორციელდება სხვადასხვაგვარად. საუკეთესო ამონახსნის საძიებლად, სრულდება ამოცანის პარამეტრების ყველა შესაძლო მნიშვნელობების მიმართული, შემთხვევითი ან კომბინირებული გადარჩევა. ამ მიზნით შემუშავებული ალგორითმები როგორც ზუსტი-სრული გადარჩევის, ასევე ევრისტიული დღევანდელობამდე ვერ იღებდნენ საუკეთესო ამონახსნებს მისაღებ დროში.

ნაშრომში ყურადღება ეთმობა ალგორითმულ საკითხებს, რომლებიც დაკავშირებულია დაფარვის ამოცანის ზუსტ და მიახლოებით ამოხსნასთან. მოცემულია ინფორმაცია შემოთავაზებული ალგორითმების მუშაობის შესახებ სატესტო მაგალითებზე.

დაფარვის ამოცანები დისკრეტული ოპტიმიზაციის კარგად ცნობილი ამოცანებია. ისინი მიეკუთვნებიან NP-რთული ამოცანების კლასს. ამ ამოცანების ამოხსნა მისაღებ დროში მეტად მნიშვნელოვანია, რაგან დაფარვის ამოცანაზე მიიყვანება დისკრეტული ოპტიმიზაციის ბევრი ამოცანა. პრაქტიკაში დაფარვის ამოცანები წარმოიშვებიან მომსახურების პუნქტების განთავსებისას (Christofides, 1986; Нечепуренко, и др., 1990), ინფორმაციული ძიების სისტემებში, ტრანსპორტის დანიშვნისას, ქსელების დიაგნოსტირებისას (Christofides, 1986), ასევე რესურსების განაწილების დაგეგმვისას გლობალურ გამოთვლით ქსელებში. ამ ამოცანების ამოხსნისას მთავარი მოთხოვნა ალგორითმების ოპერატიული მუშაობა და მიღებული ამონახსნის სიზუსტეა, მიახლოებითი ამონახსნის მიღების შემთხვევაში კი, მინიმალური ცდომილება ზუსტ ამონახსნისთან შედარებით. უმცირესი დაფარვის ამოცანის კერძო შემთხვევას წარმოადგენს უმცირესი დაყოფის ამოცანა. ამოხსნის მეთოდების ნაწილი ხშირად შემუშავებულია დაყოფის ამოცანის ამოსახსნელად და შემდეგ ხდება მათი განვრცობა დაფარვის ამოცანისათვის. (Christofides, 1986; Balinski, 1965; Garfinkel , et al., 1969; Pierce, 1968; Gomory, 1963; Goldberg, 1989), ნაშრომებში აღწერილია ამ ამოცანების ამოხსნის მეთოდები. მარტივი მეთოდები (Pierce, 1968; Gomory, 1963) იყენებენ ძებნის ხეს. (Christofides, 1986)-ში შემოთავაზებულია ალგორითმი, რომელიც იყენებს ძებნის ხეს და წრფივ დაპროგრამებას. 0-1 პროგრამირებაში მიღებული პრინციპების მსგავსი მიდგომები განხილულია (Gomory, 1963)-ში.

წინასწარ შემოვიტანოთ რამდენიმე ცნება. $G = (V, E)$ გრაფისათვის დომინირებადი სიმრავლე ეწოდება იმ წვეროების ქვესიმრავლეს: $S \subseteq V$ -ს, რომ G გრაფის ყოველ x_j წვეროსათვის, რომელიც არ ეკუთვნის S-ს, არსებობს რკალი, რომელიც გამოდის S სიმრავლის რომელიღაცა x_i წვეროდან და მთავრდება x_j წვეროში. G გრაფის ყველა დომინირებად სიმრავლეებიდან უმცირესი სიმძლავრის მქონეს ეწოდება უმცირესი

დომინირებადი სიმრავლე, ხოლო ასეთი სიმრავლის სიმძლავრეს ეწოდება დომინირების რიცხვი. (სიმრავლის სიმძლავრეში იგულისხმება ელემენტების რაოდენობა ამ სიმრავლეში).

ვთქვათ, G გრაფის მეზობლობის მატრიცის ტრანსპონირებული მატრიცაა A^T მატრიცა. G გრაფის უმცირესი დომინირებადი სიმრავლის განსაზღვრის ამოცანა ეკვივალენტურია A^T მატრიცაში ისეთი უმცირესი რაოდენობის სვეტების პოვნისა, რომ ყოველი სტრიქონი შეიცავს ერთ 1-იანს მაინც ამ ამორჩეულ სვეტებში (და მხოლოდ ერთ 1-იანს უმცირესი დაყოფის ამოცანის შემთხვევაში). ეს უკანასკნელი ამოცანა - სტრიქონების "დამფარავი" უმცირესი რაოდენობის სვეტების პოვნისა, არის უმცირესი დაფარვის სახელით ცნობილი ამოცანა. ზოგადად დასმულ დაფარვის ამოცანაში არ არის აუცილებელი, რომ 0-ებისა და 1-იანებისაგან შედგენილი მატრიცა იყოს კვადრატული. გარდა ამისა, ყოველ j -ურ სვეტს (ჩვენს შემთხვევაში ყოველ x_j წვეროს) შეესაბამება რაიმე ღირებულება c_j და მოითხოვება უმცირესი საერთო ღირებულების დაფარვის პოვნა, რაც სხვა ტერმინოლოგიით - გრაფებისათვის ნიშნავს გრაფის დომინირებადი სიმრავლის მოძებნას. რამდენადაც წვეროების უმცირესი დომინირებადი სიმრავლის აგება არის კერძო შემთხვევა დაფარვის ამოცანისა, როდესაც $c_j = 1$, ყოველი $j = 1, \dots, M$-სათვის, ერთი შეხედვით შესაძლოა მოგვეჩვენოს რომ ასეთი ამოცანის ამოხსნა უფრო მარტივია, ვიდრე ზოგადი უმცირესი დაფარვის ამოცანის ამოხსნა.

ამოცანის დასმა.

უმცირესი დაფარვის ამოცანამ სახელი მიიღო თავისი თეორიულ-სიმრავლური ინტერპრეტაციიდან. უმცირესი დაფარვის ამოცანაში მოცემულია სასრული სიმრავლე: $R = \{r_1, r_2, \dots, r_N\}$ და $S_j \subset R$ ქვესიმრავლეების ნაკრები (ოჯახი): $\mathcal{L} = \{S_1, S_2, \dots, S_M\}$, სადაც ყოველ S_j-ს მიწერილი აქვს არაუარყოფითი წონა $c_j \geq 0$. ნებისმიერ ქვენაკრებს (ოჯახს) $\mathcal{L}' = \{S_{j_1}, S_{j_2}, \dots, S_{j_k}\}$-ს $\mathcal{L}$-დან ეწოდება R სიმრავლის დაფარვა თუ სრულდება შემდეგი პირობა:

$$\bigcup_{i=1}^{k} S_{j_i} = R \qquad (3.1)$$

S_{j_i}, $i = 1, 2, \dots, k$ სიმრავლეებს კი ეწოდება დამფარავი სიმრავლეები. თუ (3.1) ტოლობის გარდა სრულდება შემდეგი ტოლობებიც:

$$S_{j_h} \cap S_{j_l} = \emptyset, \quad \forall h, l \in \{1, 2, \dots, k\}, h \neq l \qquad (3.2)$$

ანუ თუ S_{j_i}, $i = 1, 2, \dots, k$ სიმრავლეები წყვილ-წყვილად თანაუკვეთები არიან, მაშინ $\mathcal{L}'$-ს R სიმრავლის ეწოდება დაყოფა.

ასეთ ქვენაკრებებს შორის უნდა შეირჩეს ის, რომლის წონების ჯამი $\sum_{j_i}^{k} c_{j_i}$ არის მინიმალური.

ხშირად უმცირესი დაფარვის ამოცანას წერენ მატრიცული ფორმით: ვთქვათ $A = (a_{ij})$, არის $N \times M$ ზომის მატრიცი, რომლისთვისაც $a_{ij} = 1$, თუ $i \in S_j$, და $a_{ij} = 0$ წინააღმდეგ შემთხვევაში. ყოველ S_j-ს შეესაბამება $c_j \geq 0$ წონა. უმცირესი დაფარვა კი გულისხმობს იმ A მატრიცის იმ სვეტების ამორჩევას რომლებიც დაფარავენ ყველა სტრიქონს და მათი ჯამური

წონა იქნება უმცირესი. ამოცანის ფორმულირება კი ასეთია: უნდა მოვახდინოთ მიზნის ფუნქციის მინიმიზაცია:

$$f(x) = \sum_{j=1}^{M} c_j \cdot x_j \,, \qquad (3.3)$$

შემდეგი შეზღუდვების გათვალისწინებით:

$$\sum_{j=1}^{M} a_{ij} \cdot x_j \geq 1, \;\; i = 1, \dots, N, \quad x_j \in \{0,1\}, \;\; j = 1, \dots, M, \qquad (3.4)$$

აქ x_j ცვლადი უდრის 1-ს, თუ S_j სიმრავლე შედის დაფარვაში და უდრის 0-ს წინააღმდეგ შემთხვევაში.

უმცირესი დაყოფის ამოცანაში დაფარვის ამოცანისაგან განსხვავებით (3.4) უტოლობების ნაცვლად გვექნება ტოლობები:

$$\sum_{j=1}^{M} a_{ij} \cdot x_j = 1, \;\; i = 1, \dots, N, \quad x_j \in \{0,1\}, \;\; j = 1, \dots, M, \qquad (3.5)$$

გამოყენების მაგალითები

თარჯიმნების არჩევა

ვთქვათ, დაწესებულებას ესაჭიროება დაიქირავოს თარჯიმნები, რომლებიც ფრანგული, გერმანული, ბერძნული, იტალიური, ესპანური, რუსული და ჩინური ენებიდან თარგმნიან ინგლისურ ენაზე და ჰყავთ 5 კანდიდატურა: A, B, C, D და E. თითოეული კანდიდატი ფლობს ზემოთ ჩამოთვლილი ენების სიმრავლიდან ენების საკუთარ ქვესიმრავლეს და ითხოვს გარკვეული რაოდენობის ხელფასს. დაწესებულებამ უნდა აირჩიოს ისეთი კანდიდატურები ამათგან, რომ ყველა ზემოთ ჩამოთვლილი ენიდან ჰყავდეს თარჯიმანი და თარჯიმნებზე ხელფასის სახით გაცემული თანხა იყოს უმცირესი. ცხადია, რომ ეს არის უმცირესი დაფარვის ამოცანა (Christofides, 1986).

ჩავთვლოთ რომ ყველას ერთნაირი თანხა უნდა გადავუხადოთ და თარჯიმნებიდან რომელი მათგანი რომელ ენებს ფლობს მოცემულია ქვემოთ მატრიცაში:

თარჯიმნები

ენა	A	B	C	D	E
ფრანგული	1	0	0	1	1
გერმანული	1	1	0	0	0
ბერძნული	0	1	0	0	0
იტალიური	1	0	0	0	1
ესპანური	0	0	0	1	0
რუსული	0	1	1	1	0
ჩინური	0	0	1	0	1

მაშინ, ცხადია რომ დაწესებულებამ უნდა აირჩიოს B, D და E კანდიდატურები.

თვითმფრინავის ფრენის მარშრუტები

ვთქვათ, არაორიენტირებული G გრაფის წვეროები აეროპორტებს წარმოადგენენ, ხოლო G გრაფის რკალები - თვითმფრინავის გადაფრენის ეტაპებია, რომლებიც ხორციელდება მოცემულ დროში. ნებისმიერი მარშრუტი ამ გრაფში შეესაბამება ფრენის რომელიღაც შესრულებად მარშრუტს. ვთქვათ, არსებობს ასეთი N მარშრუტი და ამ მარშრუტების ღირებულება გამოთვლილია რაიმე წესით. ამოცანა რომელიც მოითხოვს ისეთი მარშრუტების სიმრავლის განსაზღვრას, რომელთა ჯამური ღირებულება იქნება მინიმალური და თითოეული ეტაპი შევა ამორჩეული მარშრუტების სიმრავლიდან ერთში მაინც, არის უმცირესი დაფარვის ამოცანა (Christofides, 1986).

3.2. უმცირესი დაყოფის ამოცანის ამოხსნა

ნაშრომში თავდაპირველად განხილულია უმცირესი დაყოფის ამოცანის ზუსტი ამოხსნის ალგორითმი (Ananiashvili, 2014), შემდეგ კი მისი განვრცობა დაფარვის ამოცანისათვის (Ananiashvili, 2015). ალგორითმი ეფუძნება შტოების და საზღვრების მეთოდს.

ნაშრომში მოყვანილია ახალი მიდგომით განხორციელებული ალგორითმის გამოყენებით Or-Llibrary-დან (Beasley, 1990) აღებული სატესტო ამოცანების ამოხსნისას მიღებული შედეგები. უმცირესი დაყოფის ამოცანის ამოხსნის სარეალიზაციო პროგრამის ლოგიკური სტრუქტურა შეიძლება აღვწეროთ შემდეგნაირად: დაყოფის მატრიცა აღვნიშნოთ A-თი, სტრიქონების რაოდენობა N-ით, სვეტების რაოდენობა M-ით. პირველ ეტაპზე დაყოფის A მატრიცა იყოფა ბლოკებად, ისე როგორც ეს (Christofides, 1986)-შია. თითოეულ ბლოკში განთავსებულია ისეთი S_j სიმრავლის შესაბამისი სვეტი, რომ S_j სიმრავლე „ფარავს“ i-ურ წვეროს, ანუ: $i \in S_j$ და შესაძლოა $i+1, i+2, \ldots, N$-დან კიდევ ზოგიერთს, მაგრამ არა 1-დან $(i-1)$-მდე. სქემატურად ბლოკებად დალაგებული დაყოფის მატრიცას ექნება ცხრილი 3.1.-ის სახე:

ცხრილი 3.1. ბლოკებად დალაგებული დაყოფის მატრიცის სქემატური გამოსახულება

<table>
<tr><th></th><th>I ბლოკი</th><th>II ბლოკი</th><th>III ბლოკი</th><th>IV ბლოკი</th><th>V ბლოკი</th><th></th></tr>
<tr><td>r_1</td><td>1...1</td><td>0</td><td rowspan="2">0</td><td rowspan="3">0</td><td rowspan="3">0</td><td rowspan="6">...</td></tr>
<tr><td>r_2</td><td rowspan="5">0 ან 1</td><td>1....1</td></tr>
<tr><td>r_3</td><td rowspan="4">0 ან 1</td><td>1...1</td></tr>
<tr><td>⋮</td><td rowspan="3">0 ან 1</td><td>1...1</td><td>1...1</td></tr>
<tr><td></td><td rowspan="2">0 ან 1</td><td></td></tr>
<tr><td>r_N</td><td>0 ან 1</td><td></td></tr>
</table>

ბლოკებად დალაგების შემდეგ თითოეული სვეტი ჩავწეროთ კომპაქტურად - მოვახდინოთ მათი „შეფუთვა“ („შეფუთვის“ იდეა ჩვენ უკვე გამოვიყენეთ §2.2-ში რიცხვების ჩასაწერად ორობით კოდში, პროცედურაზე კი დაწვრილებით ქვემოთ იქნება საუბარი). ამის შემდეგ თითოეულ ბლოკში სვეტები გადავალაგოთ მათი წონების (ფასების) ზრდის მიხედვით, რაც ასევე მნიშვნელოვნად შეამცირებს გადარჩევების რაოდენობას და შესაბამისად რეალიზაციისათვის საჭირო დროს. შემდეგ კი გამოყენებულია ძებნის ხის ალგორითმი, რომელიც მოყვანილია (Christofides, 1986)-ში. დაყოფის მატრიცის სვეტებზე განხორციელებული ძირითადი ოპერაციები სრულდება ლოგიკური ოპერატორებით, რომელებიც ასევე რეალიზაციის დროის ეკონომიას იძლევა.

ქვემოთ შემოთავაზებული შეფუთვის პროცედურა პროგრამირების ტექნიკის არაარსებითი გართულების ხარჯზე საშუალებას იძლევა დაახლოებით $32 \cdot M$-ჯერ შემცირდეს თვლისთვის საჭირო დრო, რადგან ალგორითმისათვის საჭირო N ოპერაციის ნაცვლად პროგრამაში სრულდება $[N/32] + 1$ ოპერაცია (აქ $[N/32]$ აღნიშნავს N-ის 32-ზე გაყოფის მთელ ნაწილს). ეს ოპერაციებია მასივის ელემენტების შედარების ოპერაციები - როდესაც მოწმდება სვეტების თანაუკვეთობა, მინიჭების ოპერაციები - როდესაც ახალი სვეტი ემატება მიმდინარე დაყოფის სვეტებს, ან შესაძლოა იყოს ორივე ერთად: შედარების და მინიჭების ოპერაციები, როდესაც მიმდინარე ამონახსნიდან ვშლით ზოგიერთ სვეტს.

„შეფუთვის“ იდეა კი მდგომარეობს შემდეგში: ვთქვათ, A(N×M) დაყოფის მატრიცაა. მას შემდეგ, რაც A მატრიცა დაყოფილია ბლოკებად, მიზანშეწონილია რომ ამის შემდეგ „შეფუთული“ სახით ჩაიწეროს Unsigned Long ტიპის AP დინამიურ მატრიცაში. AP მატრიცის ზომაა $([N/32] + 1) \times M$, ანუ A მატრიცის ყოველი სვეტი შეიფუთება $[N/32] + 1$ მიმდევრობით უჯრებში და განთავსდება ოპერატიულ მეხსიერებაში AP მატრიცაში. რადგან ძებნის ხის ალგორითმში ყველა სვეტის განხილვა ხდება თითოჯერ მაინც და თითოეული სვეტისთვის თვლის დრო მცირდება მიახლოებით 32-ჯერ, ამიტომ შეგვიძლია ვთქვათ რომ უარეს შემთხვევაში თვლის დრო შემცირდება $32 \cdot M$-ჯერ.

A მატრიცის j-ური სვეტი აღვნიშნოთ A_j-ით. A_j სვეტის $(j = 1,2, \dots, M)$ შეფუთვა ხორციელდება შემდეგი ალგორითმით:

ალგორითმი 3.1. მასივის შეფუთვა

ბიჯი 1. N წარმოვადგინოთ შემდეგი სახით $N = 32 \cdot l + k$, სადაც $l \geq 0$ მთელი რიცხვია, ხოლო k მთელია და $0 \leq k < 32$;

ბიჯი 2. ავიღოთ: Pck[ii]=0, ii=0,…,l (Pck ვექტორი უნდა იყოს Unsigned Long ტიპის); i=1; $k_1 = 32$;

ბიჯი 3. ავიღოთ: p=1;

ბიჯი 4. თუ $A\big((i-1) * 32 + p - 1\big) = 1$, მაშინ ავიღოთ $c='0'$ (‘0’ –არის Unsigned Long ტიპის კონსტანტა), წინააღმდეგ შემთხვევაში ავიღოთ: c=’1’;

ბიჯი 5. ავიღოთ $c = c << (32 - p)$, (სადაც $c << (32 - p)$, არის მარცხნივ ძვრა c-ში $(32 - p)$ ორობით თანრიგზე);

$Pck(i-1) = Pck(i-1)|c$ (სადაც | -ლოგიკური ჯამია);

ბიჯი 6. ავიღოთ $p = p + 1$, თუ $p \le k_1$, მაშინ გადავიდეთ ბიჯი 4-ზე;

ბიჯი 7. ავიღოთ $i = i + 1$;

ბიჯი 8. თუ $i < l$, გადავიდეთ მე-3 ბიჯზე; თუ $i = l$ ავიღოთ $k_1 = k$ და გადავიდეთ მე-3-ე ბიჯზე; წინააღმდეგ შემთხვევაში($i = l + 1$) დავასრულოთ.

ალგორითმში ასევე რეალიზებულია შეფუთვის საპირისპირო ფუნქცია, რომელიც ახორციელებს შეფუთვის შებრუნებულ ოპერაციას - ვექტორის განფუთვას (განფუთვის იდეა უკვე გამოვიყენეთ §2.2-ში რიცხვების დეკოდირებისას), ეს ალგორითმი Unsigned Long ტიპის $[N/32] + 1$ ზომის მასივს გაშლის(განფუთავს) N ელემენტიან მთელი ტიპის B მასივში. მოვიყვანოთ განფუთვის ალგორითმი:

ალგორითმი 3.2. მასივის განფუთვა

ბიჯი 1. N წარმოვადგინოთ შემდეგი სახით $N = 32 \cdot l + k$, სადაც $l \ge 0$ მთელი რიცხვია, ხოლო k მთელია და $0 \le k < 32$;

ბიჯი 2. ავიღოთ $B_j = 0,\ jj = 1, \dots, N$; $k_1 = 32$, $i = 1$;

ბიჯი 3. ავიღოთ: j=1, $c_0 = '1'$ (‘1’ –არის Unsigned Long ტიპის კონსტანტა);

ბიჯი 4. ავიღოთ: $c = c_0 << (32 - j - 1)$, (სადაც $c_0 << (32 - j - 1)$, არის მარჯვნივ ძვრა c_0-ში $(32 - j - 1)$ ორობით თანრიგზე); ავიღოთ: $c_1 = Pck(i-1) \& c$ (სადაც & -ლოგიკური გამრავლებაა);

ბიჯი 5. თუ $c_1 \ne '0'$, ავიღოთ $B((i-1) * 32 + j) = 1$;

ბიჯი 6. ავიღოთ: $j = j + 1$, თუ $j \le k_1$, მაშინ გადავიდეთ ბიჯი 4-ზე;

ბიჯი 7. ავიღოთ: $i = i + 1$, თუ $i = l$ ავიღოთ $k_1 = k$ და გადავიდეთ მე-3 ბიჯზე; წინააღმდეგ შემთხვევაში ($i = l + 1$)-ს და დავასრულოთ.

უნდა აღინიშნოს რომ ჩვენს მიერ შემუშავებული ალგორითმის აპრობაციამ კარგი შედეგები გვიჩვენა, როგორც ჩვენს მიერ შემთხვევითი წესით შედგენილი მატრიცებისათვის, ასევე Or-Library-დან (Beasley, 1990) აღებული სატესტო ამოცანებისათვის. ქვემოთ შემოთავაზებულია Or Library-ის ამოცანების ამოხსნით მიღებული შედეგები ცხრილის სახით. სადაც მითითებულია ამოცანის შესაბამისი ფაილის სახელი, ამოცანის განზომილება-მატრიცის ზომა, f^*-ით კი აღნიშნულია ალგორითმის მუშაობის შედეგად მიღებული მიზნის ფუნქციის ოპტიმალური მნიშვნელობები:

$$f^* = min \sum_{j=1}^{M} c_j \cdot x_j$$

ცხრილი 3.2. ექსპერიმენტების შედეგები დაყოფის ამოცანებისათვის

ამოცანა	სტრიქონების რაოდენობა N	სვეტების რაოდენობა M	თვლის დრო (წამებში)	ოპტიმალური მნიშვნელობა f^*	შენიშვნა
sppnw06	50	6774	54.714	7810	
sppnw07	36	5172	0.24	5476	
sppnw08	24	434	0.063	35894	
sppnw09	40	3103	2981.71	67760	
sppnw10	24	853	0.195	68271	
sppnw12	27	626	0.188	14118	
sppnw16	139	148633	123.028	1181590	
sppnw20	22	685	0.06	16812	
sppnw21	25	577	0.044	7408	
sppnw24	19	1366	0.083	6314	
sppnw25	20	1217	0.228	5960	
sppnw31	26	2662	0.147	8038	
sppnw37	19	770	0.51	10068	
sppnw40	19	404	0.089	10809	
sppnw42	23	1079	0.037	7656	
sppkl01	55	7479	37.687	1113	მიახლოებითი მნიშვნელობა
sppnw03	59	43749	4347.16	25302	მიახლოებითი მნიშვნელობა
sppnw04	36	87482	568,216	17324	მიახლოებითი მნიშვნელობა

შევნიშნოთ რომ, ცხრილში მოყვანილი ამოცანებიდან ზოგიერთისთვის მივიღეთ მიახლოებითი მნიშვნელობა, მაგ.: sppkl01 ამოცანისთვის ჩვენ თვლა შევაჩერეთ მას შემდეგ, რაც მიზნის ფუნქციის მნიშვნელობა გახდა 1113, რადგან შემდეგ დაახლოებით 1 საათის თვლის განმავლობაში ეს შედეგი აღარ გაუმჯობესებულა. ყველა დანარჩენ შემთხვევაში მიღებულია ამოცანის ზუსტი ამონახსნი. აღსანიშნავია რომ ინტერნეტში ნაშრომების ძიებისას ზუსტი ალგორითმებით ამოხსნას მწირად ვხვდებით, თუმცა ამ ამოცანებისათვის ცნობილი ოპტიმალური მნიშვნელობების ცხრილი მოყვანილი აქვთ მაგ.: (Broderick Crawford, et al., 2013)-ში. ამ ნაშრომში გამოყენებულია ჰიბრიდული მეთოდები. საინტერესოა, რომ მათი შედეგები უმეტესად ემთხვევა ან ახლოსაა ოპტიმალურ მნიშვნელობებთან. თუმცა, მაგალითად sppnw06 ამოცანისათვის ოპტიმალური მნიშვნელობაა 7810, (Broderick Crawford, et al., 2013)-ში კი ერთ შემთხვევაში მიღებულია 9788 და გაუმჯობესებული 8038. ზოგიერთი ამოცანისთვის, მაგ.: sppnw08 (Broderick Crawford, et al., 2013)-ში ამონახსნი ვერ არის მოძებნილი. საინტერესოა აგრეთვე (Kong, 2007)-ის ნაშრომი, რომელშიც გამოყენებულია წრფივი პროგრამირების ალგორითმი, რომელიც თავის მხრივ იყენებს უმცირეს კვადრატთა მეთოდს. იგივე sppnw06 ამოცანისათვის ოპტიმალური მნიშვნელობა (Kong, 2007)-ში უდრის 7640. დანარჩენი ამონახსნები ძირითადად ემთხვევა ზემოთ მოყვანილ შედეგებს. აღსანიშნავია, რომ იქ სადაც, ზუსტი მნიშვნელობები გვაქვს მიღებული ამოხსნის სისწრაფე

ძირითადად ეთანადება (Kong, 2007)-ში მიღებულ სისწრაფეს. გარდა sppnw09 და sppnw16 ამოცანებისა, რომლებსაც ჩვენთან წუთზე მეტი დასჭირდათ.

3.3. უმცირესი დაფარვის ამოცანის ამოხსნა

უმცირესი დაფარვის ამოცანის ამოსახსნელად გამოყენებულია იგივე იდეა, რომელიც განხორციელდა დაყოფის ამოცანის ამოსხსნისას - დაფარვის მატრიცის სვეტები ჯერ უნდა ჩაიწეროს შეფუთული სახით, იმის ანალოგიურად როგორც ეს გაკეთდა დაყოფის ამოცანის ამოხსნის შემთხვევაში. შემდეგ საჭირო გახდა დაყოფის ამოცანის ამოსახსნელი ალგორითმის (პროგრამის) მოდიფიკაცია დაფარვის ამოცანის შესაბამისად. მოდიფიკაციის მნიშვნელოვანი მომენტები კი მდგომარეობს შემდეგში: დაფარვის მატრიცის სვეტები ჯერ დაიყოფა ბლოკებად ისე, როგორც ზემოთ გაკეთდა დაყოფის შემთხვევაში. მაგრამ, იმის გამო, რომ აქ არ გვაქვს დაფარვის სვეტების თანაუკვეთობის მოთხოვნა, საჭიროა ყოველი $S_j = \{r_{j_1}, r_{j_2}, \dots, r_{j_k}\}$ სიმრავლის შესაბამისი სვეტის შეტანა ყველა ბლოკში: $r_{j_1}, r_{j_2}, \dots, r_{j_k}$. ცხადია ამ უკანასკნელმა გამოიწვია დაფარვის მატრიცის ზომების საგრძნობი გაზრდა. მაგ.: 50×500 ზომის მატრიცა ერთ შემთხვევაში (ამოცანა scpe1) გახდა 50×3972 ზომის, სხვა ამოცანაში (ამოცანა scpe2) 50×4055 ზომის. თუმცა შემდეგ, ვინაიდან ალგორითმის მუშაობის პროცესში, დაყოფის ამოცანისაგან განსხვავებით, მიმდინარე ამონახსნში ყოველი ახალი სვეტის დამატებისას საჭირო აღარაა ამ სვეტის თანაუკვეთობის შემოწმება მიმდინარე ამონახსნში უკვე შესულ სვეტებთან, ამოცანის ამოხსნის დრო მნიშვნელოვნად არ გაიზარდა. ეს მოხდა იმ შემთხვევებში, როდესაც ამოცანის თავდაპირველი განზომილება არ იყო ძალიან დიდი (50×500). ქვემოთმოყვანილ ცხრილში "სვეტების რაოდენობა" სვეტში მითითებულია Or Library-დან აღებული მატრიცის სვეტების რაოდენობა და ფრჩხილებში კი მოდიფიცირებული ალგორითმით მიღებული სვეტების რაოდენობა. მოდიფიცირებულმა პროგრამამ გამოთვლებისას გამოიყენა სწორედ ამ ზომის მატრიცები.

ცხრილი 3.3. ექსპერიმენტების შედეგები დაფარვის ამოცანებისათვის

ამოცანა	სტრიქონების რაოდენობა N	სვეტების რაოდენობა M	თვლის დრო (წამებში)	ოპტიმალური მნიშვნელობა f^*	შენიშვნა
scpe1	50	500(3972)	85.319	8	
scpe2	50	500 (4055)	23.483	10	
scpe3	50	500(4066)	199.404	9	
scpe4	50	500(3991)	2.067	11	
scpe5	50	500(4063)	305.338	9	
scp41	200	1000(3120)	0.222	573	მიახლოებითი მნიშვნელობა
scp53	200	2000(6006)	0.062	353	მიახლოებითი მნიშვნელობა

scp63	200	1000(8762)	0.634	183	მიახლოებითი მნიშვნელობა
scp410	200	1000(3006)	0.217	752	მიახლოებითი მნიშვნელობა

ცხრილიდან ჩანს, რომ დაფარვის ალგორითმი საკმაოდ ეფექტურად მუშაობს შედარებით პატარა განზომილების (მიახლოებით 50×500)ამოცანების ამოხსნისას, სწრაფად მიიღება ზუსტი ამონახსნი. ასევე დიდი განზომილების ამოცანების ამოსახსნელად მიახლოებითი მნიშვნელობები მიიღება საკმაოდ სწრაფად, შემდეგ ეს ამონახსნები 1 საათიანი თვლის შემთხვევაშიც კი აღარ უმჯობესდება.

ტესტირება ჩატარდა სტანდარტული სიმძლავრის კომპიუტერზე მონაცემებით: Intel(R) Pentium (R) Dual CPU E2220 2.40 GHz, 2.00 GB of RAM.

მიღებული შედეგები გვიჩვენებს, რომ ნაშრომში შემოთავაზებული დაფარვის ალგორითმი საკმარისად ეფექტურად მუშაობს შედარებით მცირე განზომილების ამოცანებისათვის. იმ ამოცანებში, კი სადაც ამ ალგორითმით მიღებულია მიახლოებითი ამონახსნები, შესაძლოა ეს ამონახსნები გამოყენებული იქნეს სხვა ალგორითმებისათვის საწყის მიახლოებად.

თავი 4. დაფარვის ამოცანის ამოხსნა გენეტიკური ალგორითმის გამოყენებით

4.1. ამოცანის ამოხსნის ალგორითმი

როგორც უკვე ვნახეთ ნაშრომის მესამე თავში ზუსტი ალგორითმები უმცირესი დაფარვის ამოცანის სირთულიდან გამომდინარე უმეტესწილად უძლურებია რეალურ დროში ამოხსნას, ან თუნდაც თვლის რომელიმე შუალედურ ეტაპზე მიიღოს კარგი მიახლოებითი ამონახსნი. სწორედ ამიტომ, მნიშვნელოვანია ეფექტური ევრისტიული ალგორითმის შემუშავება, რომლითაც სტაბილურად კარგ მიახლოებებს მივიღებთ.

ერთ-ერთი პირველი საუკეთესო მიახლოებითი ალგორითმი შემოთავაზებულია ხვატალის მიერ (Chvatal, 1979), რომელიც სტანდარტულ დაფარვის ამოცანას ხსნის პოლინომიალურ დროში. (Grossman, et al., 1997)-ში განხილულია დაფარვის სტანდარტული ამოცანა, შემოთავაზებული ევრისტიული ალგორითმი, აღნიშნულია რომ ის რიგ ამოცანებზე იძლევა უკეთეს შედეგებს მანამდე არსებულ მეთოდებთან შედარებით. სტანდარტული დაფარვის მეტაევრისტიული ალგორითმია წარმოდგენილი (Guanghui Lan, 2007)-ში. (Ching Lih Lim, et al., 2014)-ში შემოთავაზებულია უმცირესი დაფარვის ამოცანის ამოხსნის ძუნწი ალგორითმი. ქვემოთ კი ჩემს მიერ შემოთავაზებულია ამ ამოცანის ამოხსნის გენეტიკური ალგორითმი (Ananiashvili, 2015; Ananiashvili, 2015), რომელიც კარგ შედეგებს გვაძლევს სატესტო ამოცანებზე.

კვლავ ვიხილავთ უმცირესი დაფარვის (3.3) ამოცანას, (3.4) შეზღუდვებით.

უმცირესი დაფარვის ამოცანის ამოხსნის შემოთავაზებული ალგორითმი ითვალისწინებს გენეტიკური ალგორითმების ზოგად პრინციპებს (Eremeev, 2000). განვიხილოთ სქემა, რომელიც გამოიყენება შემოთავაზებულ ალგორითმში. პირველი ეტაპი გენეტიკურ ალგორითმში არის კოდირების სქემის არჩევა. არჩეულია ორობითი კოდირება, რაც ნიშნავს რომ ყოველი ქრომოსომა წარმოადგენს M განზომილებიან $x^k = \left(x_1^k, x_2^k, \dots, x_M^k\right)$ ვექტორს, რომელშიც x_j^k-ური ელემენტი მიიღებს 1-ის ტოლ მნიშვნელობას, თუ S_j სიმრავლე შედის k-ურ ქრომოსომაში და $x_j^k = 0$, თუ S_j სიმრავლე არ შედის k-ურ ქრომოსომაში. ანუ x^k-ები $k = 1,2, \dots, s$ წარმოადგენენ ქრომოსომებს, რომლებიც შეესაბამებიან ინდივიდებს. x_j^k-ები გენებია, რომლებიც შეიძლება უდრიდნენ 0-ს ან 1-ს. S_j სიმრავლეები შეესაბამებიან გენოტიპებს. ინდივიდების (ქრომოსომების) s რაოდენობა პოპულაციაში დამოკიდებულია ამოცანის განზომილებაზე. საზოგადოდ იღებენ: $25 \le s \le 100$.

ვირჩევთ საწყის პოპულაციას. შემდეგი ეტაპი არის ქრომოსომებისათვის სარგებლიანობის ფუნქციის მნიშვნელობის გამოთვლა:

$$f_k = \sum_{j=1}^{M} c_j \cdot x_j^k \,, \quad k = 1, 2, \dots, s \tag{4.1.1}$$

შემდეგ ვირჩევთ მშობლებს. გენეტიკური ალგორითმების გამოყენებით ამოცანების ამოხსნისას მშობლების შერჩევისას იყენებენ რულეტკის ან რომელიმე სხვა მეთოდს

(Рутковская , и др., 2006). შემოთავაზებულ ნაშრომში კი არჩევა ხდება შემდეგნაირად: კენტ იტერაციაზე პოპულაციიდან ავირჩიოთ ის ორი ინდივიდი, რომლებსაც აქვთ სარგებლიანობის ფუნქციის მინიმალური მნიშვნელობები. ლუწ იტერაციაზე კი ავირჩიოთ ერთი ყველაზე უკეთესი - სარგებლიანობის ფუნქციის მინიმალური მნიშვნელობის მქონე ქრომოსომა, მეორე კი ყველაზე უარესი - სარგებლიანობის ფუნქციის მაქსიმალური მნიშვნელობის მქონე. ასე ავიცილებთ სწრაფ კრებადობას რომელიმე ლოკალური მინიმუმისაკენ. არჩეული ინდივიდების შეჯვარებისათვის გამოვიყენოთ ერთწერტილიანი შეჯვარების ოპერატორი. შეჯვარების შემდეგ მიღებული ინდივიდები განიცდიან მუტაციას. ხშირად გენეტიკურ ალგორითმებში ირჩევენ მუტაციის ოპერატორის ფიქსირებულ მნიშვნელობას: $\frac{1}{M}$. ნაშრომში შემოთავაზებულია მუტაციის ცვლადი მნიშვნელობის გამოყენება, შემდეგი მოსაზრებებიდან გამომდინარე: რადგან შეჯვარებისა და მუტაციის ოპერატორების დანიშნულებაა პოპულაციაში არსებული ინდივიდებისაგან განსხვავებული ინდივიდების მიღება, ამასთან ექსტრემუმთან მიახლოებისას ინდივიდები მცირედ განსხვავდებიან ერთმანეთისაგან, ამიტომ, რომ არ გვქონდეს მხოლოდ მონათესავე ინდივიდები, გამოვიყენოთ მუტაციის ოპერატორი ცვლადი მნიშვნელობით, რომელიც დამოკიდებული იქნება როგორც თვითონ ინდივიდზე, ასევე ამ ინდივიდის გენოტიპების თავისებურებაზე. კერძოდ კი გენოტიპში შემავალი 1-იანების და 0-ების რაოდენობაზე.

შეჯვარებისა და მუტაციის შედეგად მიღებული შვილი ქრომოსომებიდან ავირჩევთ მათ შორის უკეთესს, რომელსაც ღირებულება-სარგებლიანობის ფუნქციის მნიშვნელობა მინიმალური აქვს. მოვძებნოთ საწყის პოპულაციაში მაქსიმალური ღირებულების მქონე ინდივიდი და ჩავანაცვლოთ შვილებიდან არჩეული ინდივიდით. ამ პროცესს: მშობელი ქრომოსომების არჩევა, შეჯვარება, მუტაცია, პოპულაციაში ინდივიდის შვილობილით ჩანაცვლება ვიმეორებთ იტერაციების ამოწურვამდე.

4.1.1. საწყისი პოპულაციის ფორმირება

ვიგულისხმოთ რომ A მატრიცაში სვეტები დალაგებულია ფასების ზრდის მიხედვით.

აღვნიშნოთ s-ით პოპულაციაში ინდივიდების რაოდენობა, $L-$ით $s \times M$ ზომის პოპულაციის მატრიცი.

ალგორითმი 4.1. საწყისი პოპულაციის ფორმირება.

ნაბიჯი 1. ავიღოთ: $L(k,\ j) = 0,\ \ k = 1,2, \dots, s,\ \ j = 1,2, \dots, M;\quad R(\ell) = 0, \ell = 1,2, \dots, N;$

$i = 1;\ \ t = 1;$

L მატრიცის პირველი სტრიქონი, ანუ პოპულაციის პირველი ქრომოსომა ავაგოთ შემდეგნაირად:

ნაბიჯი 2. ვიპოვოთ A მატრიცის ის პირველივე სვეტი j_1 ნომრით, რომელიც ფარავს r_i-ურ წვეროს, ანუ $\mathrm{A}(1, \mathrm{j}_1) = 1$ და შევიტანოთ დაფარვაში, ავიღოთ: $L(1, j_1) = 1.$

ავიღოთ: $R(\ell) = R(\ell) + A(\ell, j_1),\ \ell = 1,2, \dots, N.$

ნაბიჯი 3. ვიპოვოთ R-ში პირველივე ნულოვანი ელემენტის ნომერი i_1.

ავიღოთ: $i = i_1$, გადავიდეთ მე-2 ნაბიჯზე. თუ ნულოვანი ელემენტი აღარ გვაქვს გადავიდეთ ნაბიჯ 4-ზე.

L მატრიცის მომდევნო სტრიქონები, ანუ პოპულაციის დანარჩენი ქრომოსომები ავაგოთ შემდეგნაირად:

ნაბიჯი 4. ავიღოთ $t = t + 1$, თუ $t > s$ დავასრულოთ, წინააღმდეგ შემთხვევაში ავიღოთ:

$$R(\ell) = 0, \ell = 1,2, \dots, N, \ i = 1.$$

ნაბიჯი 5. ვიპოვოთ A მატრიცის იმ სვეტების ნომრები $\{j_1, j_2, \dots, j_{h_i}\}$, რომლებიც ფარავს r_i-ურ წვეროს. $\{j_1, j_2, \dots, j_{h_i}\}$-დან შემთხვევითი წესით ავირჩიოთ რომელიმე j_u ნომერი, ანუ $A(i, j_u) = 1$ და შევიტანოთ მიმდინარე t-ურ ქრომოსომაში: $L(t, j_u) = 1$. ავიღოთ: $R(\ell) = R(\ell) + A(\ell, j_u), \ \ell = 1,2, \dots, N.$

ნაბიჯი 6. ვიპოვოთ R-ში პირველივე ნულოვანი ელემენტის ნომერი i_1. თუ ასეთი ელემენტი გვაქვს ავიღოთ $i = i_1$, გადავიდეთ მე-5 ნაბიჯზე. თუ ნულოვანი ელემენტი აღარ გვაქვს გადადი ნაბიჯ 4-ზე.

ასეთი წესით აგებულ პოპულაციაში სავსებით შესაძლებელია რომ თითოეულ ქრომოსომაში აღმოჩნდეს ჭარბი გენები, ანუ თითოეულ დაფარვაში შემავალი სვეტებიდან რომელიმე, ან რამდენიმე სვეტის წაშლის შემდეგ $R = \{r_1, r_2, \dots, r_N\}$ სიმრავლე ისევ აღმოჩნდეს დაფარული. J დაფარვას ვუწოდოთ ჩიხური დაფარვა, თუ ნებისმიერი $j \in J$-სათვის $J \setminus \{j\}$ აღარ წარმოადგენს დაფარვას. სწორედ ამიტომ საჭიროა მიღებული პოპულაციის თითოეული ინდივიდიდან წავშალოთ ზედმეტი სვეტები და გავხადოთ ისინი ჩიხური. ამისათვის გამოვიყენოთ შემდეგი ევრისტიული ალგორითმი:

ალგორითმი 4.2. ჭარბი სვეტების წაშლა k-ურ ინდივიდში.

ნაბიჯი 1. ყოველი $\ell = 1, 2, \dots, N$-ისათვის $R(\ell)$ განვსაზღვროთ შემდეგნაირად:

$R(\ell) = h_\ell$, სადაც h_ℓ პოპულაციის k-ურ ქრომოსომაში r_ℓ-ურ წვეროს დამფარავი სვეტების რაოდენობაა. ანუ სხვაგვარად რომ ვთქვათ: k-ურ ქრომოსომაში, რომელიც L მატრიცის k-ურ სტრიქონშია წარმოდგენილი, უნდა ვიპოვოთ იმ დამფარავი სვეტების $\{j_1, j_2, \dots, j_{h_\ell}\}$ რაოდენობა: h_ℓ, რომლებისთვისაც $A(\ell, j_p) = 1, \ p = 1, 2, \dots, h_\ell$.

ავიღოთ: $i = 1$.

ნაბიჯი 2. ვიპოვოთ უმცირესი ნომერი i_1: $i \le i_1 \le N$, რომ $R(i_1) > 1$, თუ ასეთი არ გვაქვს მაშინ დასასრული.

ავიღოთ: $u = R(i_1), \quad p = 1.$

ნაბიჯი 3. თუ ყველა $R(l) - A(l, j_p) > 0, \ l = 1,2, \dots, N$, მაშინ k-ური ქრომოსომიდან შესაძლებელია j_p სვეტის წაშლა. ავიღოთ:

$$R(l) = R(l) - A(l,\ j_p), \quad l = 1,2,\dots,N \quad \text{და} \quad L(k,\ j_p) = 0.$$

ნაბიჯი 4. ავიღოთ: $p = p + 1$, თუ $p \le u$ გადავიდეთ ნაბიჯი 3-ზე.

ნაბიჯი 5. ავიღოთ: $i = i_1 + 1$, თუ $i \le N$ გადადი ნაბიჯ 2-ზე, წინააღმდეგ შემთხვევაში დასასრული.

ამ ალგორითმით L პოპულაციის თითოეულ k-ურ ($k = 1, 2, \dots, s$) ინდივიდში წავშლით ჭარბ სვეტებს და მივიღებთ ჩიხურ დაფარვებს. მოვიყვანოთ პროგრამის ფრაგმენტი, რომელითაც შესაძლებელია x^{ch_1} გავხადოთ ჩიხური:

```
for i=1:N
    if R1(i)==1
        continue
    end
    max1=R(i);
it=1;
k=find(A(i,(ch1~=0)));
while it<=max1
      if (numel(find(R-A(:,k(it))==(-1|0)))==0)
         ch1(k(it))=0; R=R-A(:,k(it));
      end
end
end
```

4.1.2. შეჯვარება

პოპულაციის ფორმირების შემდეგ თითოეული ქრომოსომისათვის გამოვითვლით სარგებლიანობის ფუნქციის მნიშვნელობებს (2.1) ფორმულებით. ზემოთ 2.2. პარაგრაფში აღწერილი წესის მიხედვით ვარჩევთ ორ მშობელს-ორ ქრომოსომას. ვთქვათ, ესენია:

$$x' = (x'_1, x'_2, \dots, x'_M)$$
$$x'' = (x''_1, x''_2, \dots, x''_M)$$

შეჯვარების პოზიცია განვსაზღვროთ შემთხვევითი სიდიდით $k \in \{1,2,3,\dots,M-1\}$. შეჯვარების შემდეგ მივიღებთ შვილობილ ქრომოსომებს:

$$x^{ch_1} = (x'_1, x'_2, \dots x'_k, x''_{k+1}, \dots, x''_M)$$
$$x^{ch_2} = (x''_1, x''_2, \dots, x''_k, x'_{k+1}, \dots, x'_M)$$

ასეთი შეჯვარების შემდეგ შესაძლოა რომ შვილობილი ქრომოსომები ვეღარ ფარავდნენ $R = \{r_1,\ r_2, \dots, r_N\}$-ს. საჭიროა ამ ქრომოსომებში დაუფარავი წვეროებისათვის მათი დამფარავი ქვესიმრავლის მოძებნა და ჩამატება. მოვიყვანოთ ალგორითმი, რომელიც ყოველივე ამას უზრუნველყოფს:

ალგორითმი 4.3. ქრომოსომებში დაუფარავი წვეროების დამფარავი ქვესიმრავლის ჩამატება.

ალგორითმში გარკვეულობისათვის აღებულია $\mathrm{x}^{\mathrm{ch}_1}$ ქრომოსომა.

ნაბიჯი 1. ვიპოვოთ x^{ch_1} ქრომოსომაში არანულოვანი ელემენტების ნომრები: $\{\mathrm{j}_1, \mathrm{j}_2, \dots, \mathrm{j}_\mathrm{h}\}$- ეს არის ამ ქრომოსომის შესაბამისი დაფარვის სვეტების ნომრები. დაფარვის A მატრიცის $\{j_1, j_2, \dots, j_h\}$ სვეტების ყოველ ℓ-ურ სტრიქონში, $\ell = 1,2, \dots, N$-ისათვის, დავითვალოთ იმ სვეტების რაოდენობა: k_ℓ, რომლებიც ფარავენ r_ℓ-ურ წვეროს. ანუ რომლებისთვისაც:

$$A\left(\ell, j_{i_p}\right) = 1, \;\; p = 1,2, \dots, k_\ell \quad \text{და} \quad \left\{j_{i_1}, j_{i_2}, \dots, j_{i_{k_\ell}}\right\} \subset \{j_1, j_2, \dots, j_h\}$$

ავიღოთ: $R(\ell) = k_\ell$; $i = 1$;

ნაბიჯი 2. ვიპოვოთ უმცირესი ნომერი i_1: $i \le i_1 \le N$, რომლისთვისაც: $\mathrm{R}(i_1) = 0$. თუ R-ის ყველა ელემენტი არანულოვანია დასასრული.

ნაბიჯი 3. ვიპოვოთ A მატრიცის იმ სვეტების ნომრები $\{\mathrm{j}_1, \mathrm{j}_2, \dots, \mathrm{j}_\mathrm{v}\}$, რომელებიც ფარავს $\mathrm{r}_{\mathrm{i}_1}$ წვეროს. $\{\mathrm{j}_1, \mathrm{j}_2, \dots, \mathrm{j}_\mathrm{v}\}$-დან შემთხვევითი წესით ავირჩიოთ რომელიმე j_u ნომერი, ანუ $A(\mathrm{i}_1, \mathrm{j}_\mathrm{u}) = 1$. შევიტანოთ j_u სვეტი მიმდინარე ქრომოსომაში: $\mathrm{x}^{\mathrm{ch}_1}(\mathrm{j}_\mathrm{u}) = 1$.

ავიღოთ: $\mathrm{R}(\ell) = \mathrm{R}(\ell) + A(\ell, \mathrm{j}_\mathrm{u}),\ \ell = 1,2, \dots, N$.

ნაბიჯი 4. ავიღოთ $i = i_1$+1. თუ $\mathrm{i} \le N$, გადავიდეთ ნაბიჯი 2-ზე, წინააღმდეგ შემთხვევაში დასასრული.

ამ ალგორითმის გამოყენებით x^{ch_1} და x^{ch_2} ქრომოსომები კვლავ დაფარავენ $\mathrm{R} = \{r_1, r_2, \dots, r_N\}$ სიმრავლეს. შეჯვარების შემდეგ შვილობილი ქრომოსომები განიცდიან მუტაციას.

მოვიყვანოთ პროგრამის ფრაგმენტი, რომელითაც შესაძლებელია x^{ch_1} გავხადოთ დამფარავი:

```
t=true;
while t
      k0=numel(find(R==0,1,'first'));
   if k0==0
       break
   else
       k=find(R1==0,1,'first');
   end
   r=find(A(k,:)==1,1,'first');
   R=R+A(:,r);
   ch1(r)=1;
end
```

შენიშვნა: `find(R==0,1,'first')` აქ პოულობს `R` ვექტორში პირველი 0-ოვანი ელემენტის რიგით ნომერს, თუ ასეთი (0-ოვანი ელემენტი) არ გვხვდება `R` ვექტორში, მაშინ შედეგი იქნება ცარიელი. `numel` ფუნქცია განსაზღვრავს მასივის სიგრძეს, თუ აღმოჩნდა რომ `numel` ფუნქციის არგუმენტი ცარიელი მასივია, მაშინ დაგვიბრუნებს 0-ს.

4.1.3. მუტაცია

თითოეული შვილობილი x^{ch_1} და x^{ch_2} ქრომოსომისათვის შემთხვევითი წესით ავიღებთ რომელიმე რიცხვს, k-ს: $k \in \{1,2,3,\dots,M\}$ სიმრავლიდან და ვახდენთ ამ ქრომოსომის k-ური გენის მუტაციას. შევნიშნოთ რომ ამ ამოცანაში გენების მახასიათებელია როგორც მათი წონა, ასევე მის შესაბამის გენოტიპში 1-იანების და 0-ების რაოდენობა. A მატრიცის ყოველ j-ურ სვეტში: $j = 1,2,\dots,M$-სათვის დავითვალოთ 1-იანების რაოდენობა - $p_1(j)$ და 0-ების რაოდენობა - $p_0(j)$. ასევე ყოველი სვეტისათვის გამოვითვალოთ სიდიდეები:

$$E(j) = -p_0(j) \cdot log\big(p_0(j)\big) - p_1(j) \cdot log\big(p_1(j)\big),\ j = 1,2,\dots,M \qquad (4.1.2)$$

შემდეგ კი შევადგინოთ მუტაციის ალბათობის ვექტორი:

$$pmt(j) = \frac{\frac{1}{E(j)}}{\sum_{u=1}^{n}\frac{1}{E(u)}}\ ,\quad j = 1,2,\dots,M \qquad (4.1.3)$$

ავირჩიოთ τ გენი მუტაციისათვის, სადაც τ შემთხვევითი ნორმალიზებული მთელი რიცხვია $1 \le \tau \le M$ შუალედიდან. ჩამოვაყალიბოთ მუტაციის პირობა:

1. თუ სრულდება შემდეგი პირობები: $pmt(\tau) > \frac{1}{M}$, $x^{ch_1}(\tau) = 1$ და $p_0(\tau) > p_1(\tau)$ მაშინ τ გენი განიცდის მუტაციას, შესაბამისად გვექნება: $x^{ch_1}(\tau) = 0$.

2. თუ სრულდება შემდეგი პირობები: $pmt(\tau) > \frac{1}{M}$, $x^{ch_1}(\tau) = 0$ და $p_0(\tau) > p_1(\tau)$ მაშინ τ გენი განიცდის მუტაციას, შესაბამისად გვექნება: $x^{ch_1}(\tau) = 1$.

ანალოგიურად მოვახდენთ მეორე x^{ch_2} ქრომოსომის მუტაციას. ისევე როგორც შეჯვარებისას, მუტაციის შემდეგ შვილობილმა ქრომოსომებმა შესაძლოა ვეღარ დაფარონ $R = \{r_1, r_2, \dots, r_N\}$ სიმრავლე. გამოვიყენოთ ზემოთ მოყვანილი ალგორითმი 2.4. და x^{ch_1} და x^{ch_2}-ს ჩავუმატოთ სვეტები, რომლებიც დაფარავენ ყველა დაუფარავ წვეროს. სვეტების ჩამატების შემდეგ მიღებულ ქრომოსომები შესაძლოა აღარ წარმოადგენდნენ ჩიხურ დაფარვებს. გამოვიყენოთ ალგორითმი 4.2. და წავშალოთ თითოეულიდან ზედმეტი სვეტები. ამ პროცესს ვიმეორებთ იტერაციების ამოწურვამდე.

4.1.3. ექსპერიმენტების შედეგები

შემოთავაზებული ალგორითმები რეალიზებულია Matlab-ში. Matlab-ს გააჩნია ის სასარგებლო ფუნქციები, რომლებიც პროგრამის კომპაქტურად ჩაწერის საშუალებას იძლევიან. რაც ზემოთ მოყვანილ პროგრამულ ფრაგმენტებშიც ჩანს. ალგორითმის ეფექტურობის შესამოწმებლად გამოყენებულია სატესტო ამოცანები ცნობილი ბიბლიო-თეკიდან: Or-Library-დან (Beasley, 1990), რომელშიც ასეთი 11 კლასის ამოცანაა. ეს კლასებია: 4, 5, 6, A, B, C, D, E, F, G, H. ისინი შემთხვევითი წესით აგებული ამოცანებია $c_j \in [1, 100]$ წონებით და A მატრიცებით, რომლებშიც 1-იანების რაოდენობა 2%-დან 20%-მდეა.

ცხრილი 4.1.1. ამოცანების სახელები და მათი განზომილებები.

ამოცანის კლასი	სტრიქონების რაოდენობა	სვეტების რაოდენობა	ამოცანების რაოდენობა და ფაილების სახელები
4	200	1000	10 scp41, ..., scp410
5	200	2000	10 scp51, ..., scp510
6	200	1000	5 scp61, ..., scp65
A	300	3000	5 scpa1, ..., scpa5
B	300	3000	5 scpb1, ..., scpb5
C	400	4000	5 scpc1, ..., scpc5
D	400	4000	5 scpd1, ..., scpd5
E	500	5000	5 scpe1, ..., scpe5
F	500	5000	5 scpnrf1, ..., scpnrf5
G	1000	10000	5 scpnrg1, ..., scpnrg5
H	1000	10000	5 scpnrh1, ..., scpnrh5

თითოეულ ამოცანაზე ჩატარდა 10-10 ექსპერიმენტი, ყოველი ექსპერიმენტისათვის ირჩეოდა ახალი პოპულაცია. მიღებული შედეგებიდან საუკეთესოები მოყვანილია ცხრილი 4.1.2-ში და 4.1.3 -ში. შედეგები შედარებულია ამ ამოცანებისათვის ცნობილ უმცირესი დაფარვის ოპტიმალური წონების მნიშვნელობებთან. სულ განხილულია 65 ამოცანა. გამოთვლებისას პოპულაციების ზომა ყველგან აღებულია $s = 100$. იტერაციების რაოდენობა კი იცვლებოდა 550-დან 35250 ინტერვალში. 4-6 და A-D სერიის ამოცანებისათვის ცნობილია

ოპტიმალური მნიშვნელობები (Beaslay J. E., 1992), ხოლო E-H სერიის ამოცანებისათვის, რომლებიც დიდი განზომილების ამოცანებია, მათთვის შესადარებლად აღებულია ცნობილი საუკეთესო შედეგები (Jacobs , et al., 1993).

აღვნიშნოთ f^* - ექსპერიმენტებით მიღებული საუკეთესო მნიშვნელობა.

ცხრილი 4.1.2. ექსპერიმენტების შედეგები 4-6 სერიის ამოცანებისათვის, რომელშიც ნაჩვენებია საუკეთესო მნიშვნელობები ჩატარებული ექსპერიმენტებიდან:

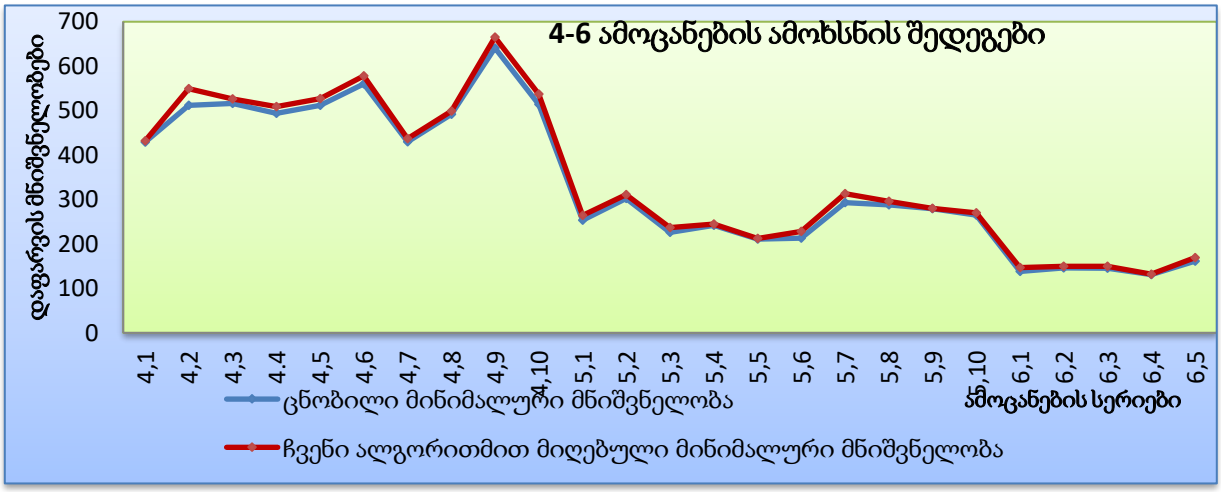
ამოცანა	ოპტიმალური ამონახსნი	მინიმალური მნიშვნელობა f^*	იტერაციების რაოდენობა	თვლის დრო(წმ)
4,1	429	432	850	5.748834
4,2	512	549	27250	189.521844
4,3	516	526	650	5.591801
4.4	494	509	5250	41.614141
4,5	512	527	650	7.285392
4,6	560	578	550	4.547840
4,7	430	437	650	4.874183
4,8	492	499	750	5.513872
4,9	641	665	15250	84.042923
4,10	514	537	750	5.267568
5,1	253	264	27250	184.503032
5,2	302	311	25250	160.479980
5,3	226	237	1250	10.480415
5,4	242	245	5250	38.623774
5,5	211	212	5250	36.555420
5,6	213	228	5250	35.234376
5,7	293	313	1250	11.987480
5,8	288	296	15250	106.514343
5,9	279	280	35250	254.069516
5,10	265	270	45250	345.253260
6,1	138	147	550	4.952619
6,2	146	150	35250	239.397738
6,3	145	150	5250	35.749140
6,4	131	132	5250	38.840746
6,5	161	169	1250	10.292615

ცხრილი 4.1.3. ექსპერიმენტების შედეგები A-D სერიის ამოცანებისათვის, რომელშიც ნაჩვენებია საუკეთესო მნიშვნელობები ჩატარებული ექსპერიმენტებიდან:

ამოცანა	ოპტიმალური ამონახსნი	მინიმალური მნიშვნელობა f^*	იტერაციების რაოდენობა	თვლის დრო (წმ)
A,1	253	255	1250	20.120029

A,2	252	267	850	15.381661
A,3	232	246	45250	699.561851
A,4	234	246	5250	79.033534
A,5	236	240	15250	279.40790
B,1	69	76	15250	260.158726
B,2	76	83	5250	114.132584
B,3	85	85	550	13.921434
B,4	79	83	5250	97.910171
B,5	72	75	650	12.822692
C,1	227	233	35250	886.189983
C,2	219	227	40250	835.990969
C,3	243	254	15250	367.848676
C,4	219	235	35250	844.615780
C,5	215	221	850	26.015785
D,1	60	60	35250	859.387968
D,2	66	70	550	23.790084
D,3	72	77	15250	452.277166
D,4	61	64	850	28.991679
D,5	62	64	550	18.933103

ცხრილი 4.1.1.-ის მიხედვით 4-6 ამოცანებისათვის ავაგოთ დიაგრამა, ჩვენი ალგორითმით მიღებული შედეგების ამ ამოცანების ცნობილ ოპტიმალურ ამონახსნებთან შესადარებლად.

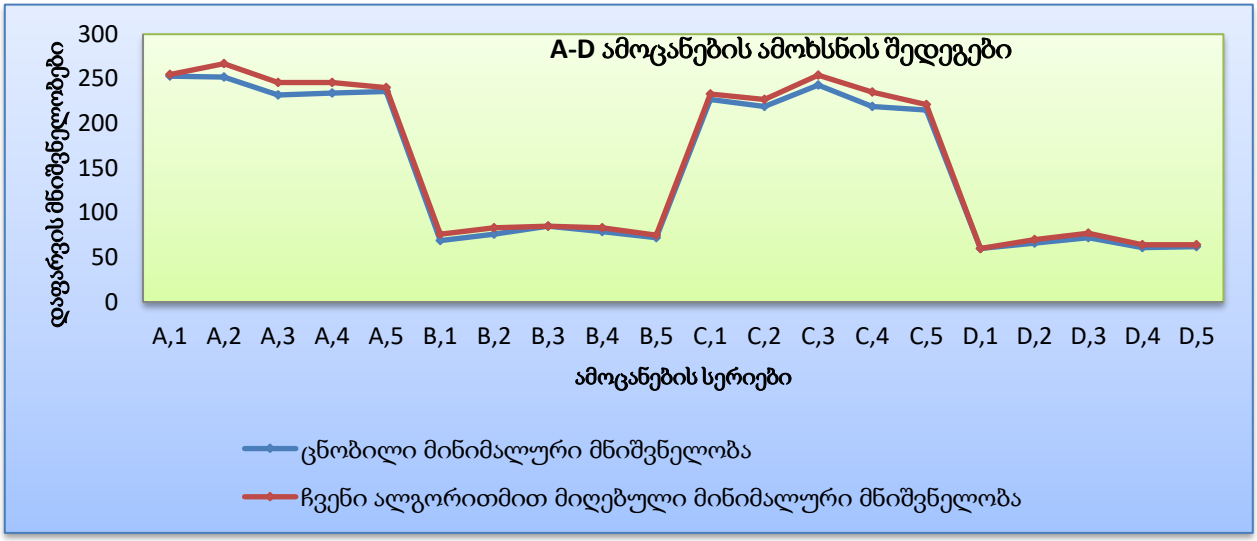


ნახ. 4.1.1. 4-6 ამოცანების ამოხსნის შედეგები

ყველაზე დიდი ცდომილება აქ არის 7%, უმეტეს შემთხვევაში კი ის იცვლება 0.3%-დან 1%-მდე. საშუალო ცდომილება გამოვიდა 3%.

ექსპერიმენტებში მიღებული საუკეთესო მნიშვნელობები A-D ამოცანებისათვის მოყვანილია ცხრილი 4.1.2-ში. ჩვენი ალგორითმით მიღებული შედეგების ამ ამოცანების

ცნობილ ოპტიმალურ ამონახსნებთან შესადარებლად. აქ (ნახ. 4.4.) ყველაზე დიდი ცდომილება არის 4%, საშუალო ცდომილება გამოვიდა 1%, ამონახსნი რიგ შემთხვევაში დაემთხვა ცნობილ ამონახსნებს.



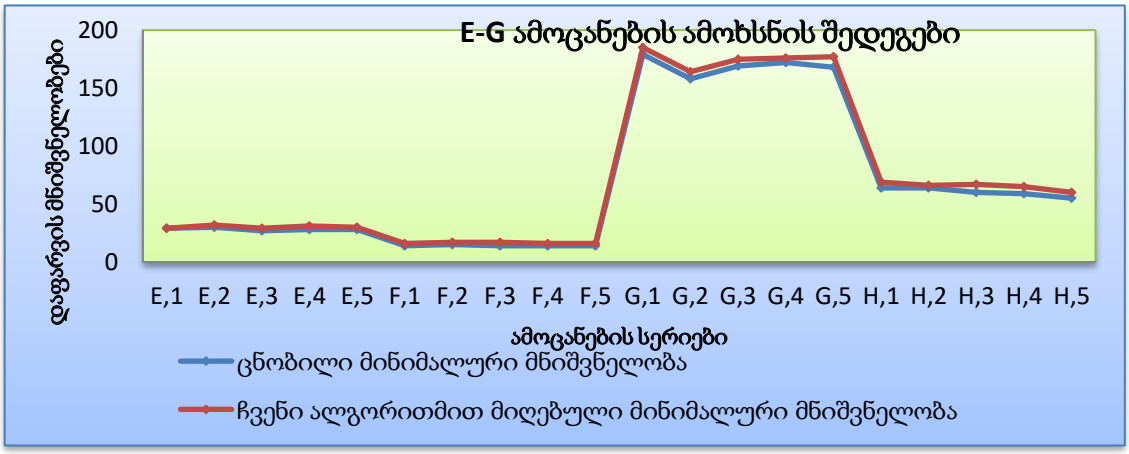
ნახ. 4.1.2. A-D ამოცანების ამოხსნის შედეგები

ცხრილი 4.1.4. ექსპერიმენტების შედეგები E-H სერიის ამოცანებისათვის, რომელშიც ნაჩვენებია საუკეთესო მნიშვნელობები ჩატარებული ექსპერიმენტებიდან:

ამოცანა	შემოთავაზებული მინიმალური მნიშვნელობა	მინიმალური მნიშვნელობა f^*	იტერაციების რაოდენობა	თვლის დრო (წმ)
E,1	29	29	650	37.089270
E,2	30	32	1250	55.733434
E,3	27	29	5250	222.218509
E,4	28	31	15250	795.258351
E,5	28	30	550	31.467857
F,1	14	16	550	38.747741
F,2	15	17	650	36.459649
F,3	14	17	550	32.964627
F,4	14	16	850	45.287650
F,5	14	16	550	31.603698
G,1	179	185	35250	4163.728506
G,2	158	164	25250	2921.625042
G,3	169	175	5250	932.124276
G,4	172	176	25250	3059.695269
G,5	168	177	35250	5870.136848

H,1	64	69	5250	706.016420
H,2	64	66	35250	4681.564954
H,3	60	67	1250	226.269730
H,4	59	65	550	111.810822
H,5	55	60	850	148.935323

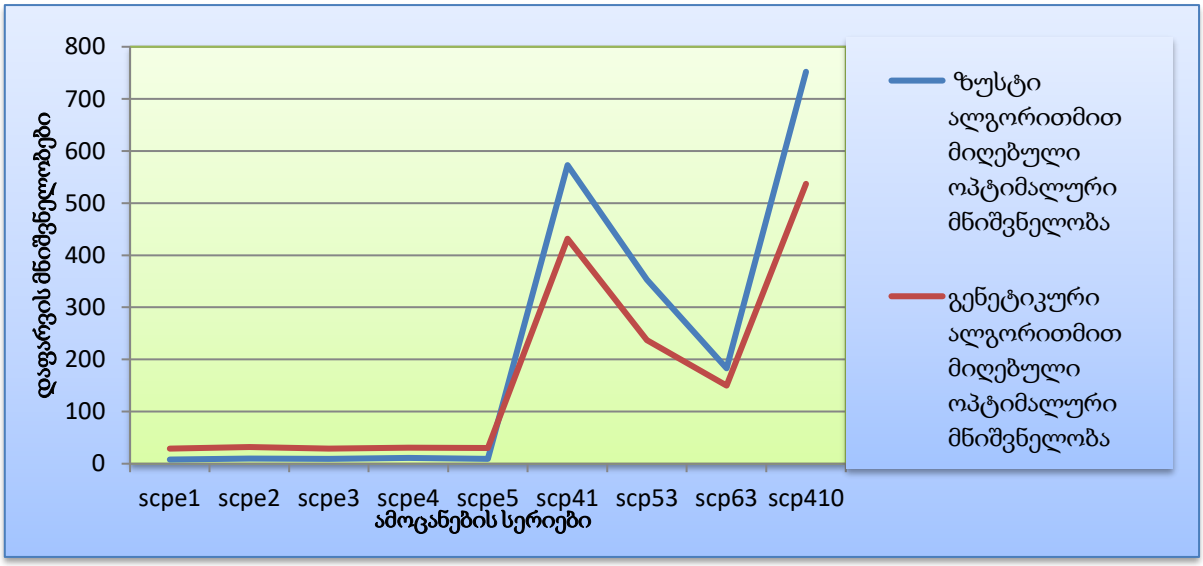
ცხრილი 4.1.3.-დან ავაგოთ დიაგრამა E-G ამოცანებისათვის ჩვენი ალგორითმით მიღებული შედეგების ცნობილ ოპტიმალურ ამონახსნებთან შესადარებლად.



ნახ. 4.1.3. E-G ამოცანების ამოხსნის შედეგები

აქ ყველაზე დიდი ცდომილება არის 11%, საშუალო ცდომილება გამოვიდა 8%, ამონახსნი ერთ შემთხვევაში დაემთხვა ცნობილ საუკეთესო ამონახსნს..

ტესტირება ჩატარდა სტანდარტული სიმძლავრის კომპიუტერზე მონაცემებით: Intel(R) Pentium (R) Dual CPU E2220 2.40 GHz, 2.00 GB of RAM. უმეტეს შემთხვევაში ამოცანების ამოხსნას ესაჭიროება წამები, ზოგიერთ შემთხვევაში რამდენიმე წუთი. მხოლოდ E-H სერიის რამდენიმე ამოცანას დასჭირდა საათზე მეტი დრო. შევადაროთ გენეტიკური ალგორითმით მიღებული შედეგები ზუსტი ალგორითმით მიღებულ შედეგებს, რომლებიც მოყვანილია წინა თავში იხ. §4.3.-ში ცხრილი 4.2.



ნახ. 4.1.4. დაფარვის ზოგიერთი ამოცანის ზუსტი და მიახლოებითი ალგორითმების შედეგების შედარებითი დიაგრამა.

დიაგრამიდან ჩანს რომ scpe1… scpe5 ამოცანებში ზუსტი ალგორითმით მიღებულია უკეთესი შედეგები, სხვა ამოცანებში სადაც ზუსტი ალგორითმით მიღებული გვქონდა მიახლოებითი მნიშვნელობები, გენეტიკური ალგორითმმა გაცილებით უკეთესი შედეგები აჩვენა. ამავე დროს უმეტეს შემთხვევაში ზუსტი ალგორითმით თუნდაც მიახლოებითი ოპტიმალური შედეგების მიღება შეუძლებელი ხდება რეალურ დროში.

ყოველივე ზემოთ თქმულიდან და ჩატარებული ექსპერიმენტების საფუძველზე შეგვიძლია დავასკვნათ რომ შემოთავაზებული გენეტიკური ალგორითმი საკმარისად ეფექტურად მუშაობს.

4.2. უმცირესი დაფარვის განზოგადებული ამოცანის ამოხსნა გენეტიკური ალგორითმით

4.2.1. ამოცანის დასმა

§3.1-ში დასმულ დაფარვის (3.3)-(3.4) ამოცანაში (3.4) პირობები შევცვალოთ ახალი პირობებით:

$$\sum_{j=1}^{M} a_{ij} \cdot x_j \geq b_i, \quad i = 1, \dots, N, \quad x_j \in \{0, 1\}, \quad j = 1, \dots, M, \qquad (4.2.1)$$

უმცირესი დაფარვის განზოგადებულ ამოცანაში მოითხოვება ისეთი დაფარვის მოძებნა, რომ კვლავ მოხდეს

$$f(x) = \sum_{j=1}^{M} c_j \cdot x_j$$

ფუნქციის მინიმიზება (4.2.1) შეზღუდვების პირობებში, სადაც $b_i, \ i = 1, \dots, N$ -ები წინასწარ მოცემული ნატურალური რიცხვებია, რომელთა დიაპაზონი არ არის დიდი (გაცილებით მცირეა N-ზე).

უმცირესი დაფარვის სტანდარტულ ამოცანაში, დაფარვაში მოიაზრებოდა A მატრიციდან არჩეული სვეტების ელემენტების თანრიგობრივი ლოგიკური ჯამი. ხოლო b ვექტორად აიღებოდა ვექტორი, რომლის ყველა ელემენტი 1-ის ტოლია. ვიტყვით რომ d ვექტორი ფარავს რაიმე b ვექტორს, თუ $d \geq b$, ანუ $d_i \geq b_i, i = 1, \dots, N$. მაშასადამე, უმცირესი დაფარვის განზოგადებული ამოცანა გულისხმობს A მატრიციდან ისეთი სვეტების ამორჩევას, რომელთა გაერთიანება ფარავს b ვექტორს და არჩეული სვეტებისათვის $f(x)$ ფუნქცია მინიმალურია (Ananiashvili, 2016).

აქ ჩამოყალიბებული ამოცანა პრაქტიკული გამოყენების თვალსაზრისით შესაძლოა წარმოიშვას ისეთი ცხოვრებისეული პრობლემების გადაჭრისას, როდესაც გვჭირდება მაგალითად მომსახურების ობიექტების ისეთი განლაგება, რომ ერთ მომხმარებელს შეეძლოს მასთან უახლოეს რამდენიმე ასეთი ობიექტიდან ერთ-ერთის მომსახურების არჩევა, ანუ მისთვის მოსახურების ობიექტი უნდა იყოს ერთი ან მეტი. მაგალითად ბანკების განლაგებისას, როდესაც საჭიროა რიგების თავიდან არიდება, ან თვითონ ბანკში მომსახურე ოპერატორების რაოდენობის განსაზღვრისას, და სხვა პრაქტიკულ ამოცანებში. გრაფების ენაზე კი, ასეთი ამოცანა შესაძლოა გამოყენებული იყოს ჯერადი ცენტრების ამოცანის ამოხსნისას (Minieka, 1978).

(Yossi Azar, et al., 2009)-ში ავტორები განიხილავენ უმცირესი დაფარვის განზოგადებულ პრობლემას, შემოთავაზებული აქვთ ლოგარითმული სიზუსტის მიახლოებითი ალგორითმი. (Nikhil Bansal, et al., 2010) შრომაში ავტორებმა გააუმჯობესეს (Yossi Azar, et al., 2009)-ის შედეგი. (Yang, et al., 2005)-ში და (Umetani, et al., 2013) შრომებში განიხილება განზოგადებული აწონილი დაფარვის პრობლემა, მოითხოვება ყოველი ელემენტის მრავალჯერადი დაფარვა. ავტორებს შედარებისათვის გამოყენებული აქვთ მთელრიცხვა

პროგრმირების ამოცანების ამომხსნელი - კომერციულ პროგრამა CPLEX (IBM, 2000), მათ მიერვე ფორმულირებულ სატესტო ამოცaნებზე, იმ მოტივით რომ მათთვის არ არის ცნობილი სხვა ავტორების მიერ მიღებული შედეგები. (Есипов, и др., 2014) ნაშრომში შემოთავაზებულია ორი ალგორითმი: ადიტიური და გენეტიკური, რომელთა ეფექტურობა შემოწმებულია ავტორების მიერ შეთხვევითი წესით აგებულ მატრიცებზე, წონების ვექტორიც ივსება შემთხვევითი წესით.

4.2.2. პრობლემის გადაწყვეტა და გამოთვლების შედეგები

ზემოთ დასმული უმცირესი დაფარვის განზოგადებული ამოცანა ამოვხსენი §4.1.-ში მოყვანილი გენეტიკური ალგორითმის გამოყენებით, რომელშიც საჭიროა (3.4)-პირობების ნაცვლად გავითვალისწინოთ (4.1) პირობები. კერძოდ:

- ალგორითმ 4.1-ში(საწყისი პოპულაციის ფორმირება) ნაბიჯი 3-ში და 6-ში პირობა: ვიპოვოთ R-ში პირველივე ნულოვანი ელემენტის ნომერი i_1-ის ნაცვლად უნდა ვიპოვოთ პირველივე ელემენტი, რომელიც აკმაყოფილებს პირობას: $R(i_1) < b(i_1)$;
- ალგორითმ 4.2-ში(ჭარბი სვეტების წაშლა k-ურ ინდივიდში) ნაბიჯი 2-ში პირობა: $R(i_1) > 1$ -ის ნაცვლად უნდა შემოწმდეს პირობა: $R(i_1) > b(i_1)$. ნაბიჯი 3-ში პირობა:$R(l) - A(l,\ j_p) > 0,\ l = 1,2,\dots,N$-ის ნაცვლად უნდა შემოწმდეს პირობა: $R(l) - A(l,\ j_p) > b_l,\ l = 1,2,\dots,N$;
- ალგორითმ 4.3-ში(ქრომოსომებში დაუფარავი წვეროების დამფარავი ქვესიმრავლის ჩამატება) ნაბიჯი 2-ში პირობა: $R(i_1) = 0$-ის ნაცვლად უნდა შემოწმდეს პირობა: $R(i_1) \le b(i_1)$;

ვინაიდან ჩვენს მიერ დასმული ამოცანისათვის ჯერ-ჯერობით სატესტო ამოცანები არ არსებობს, ბოლოდროინდელ ანალოგიურ კვლევებში კი ავტორებს ექსპერიმენტებისათვის გამოყენებული აქვთ საკუთარი - შემთხვევითი წესით ფორმირებული დაფარვის მატრიცები და შესაბამისი წონების მასივებიც, ამიტომ საექსპერიმენტოდ ამოცანები ავიღე Or-Library-დან, მაგრამ რადგანაც ამ ამოცანებში იგულისხმება რომ: $b_i = 1,\ i = 1,\dots,N$, ამიტომ გამოთვლებისას ავიღე შემთხვევითი რიცხვები: $b_i \in [1,3],\ i = 1,\dots,N$ (პროგრამაში: $b = randi([1,3],N,1);$). შემდეგ კი შედარებისათვის ჩემს მიერ ამოხსნილი ამოცანებისათვის შევადგინე AMPL-აპლიკაციის მოდელები და ამოვხსენი NEOS Server-ის გამოყენებით (Gurobi და CPLEX ამომხსნელები). იმ ამოცანებისათვის რომელთა დაფარვის მატრიცის ზომები 100×1000, 200×1000, 200×2000-ია Gurobi ამომხსნელით უკეთესი მნიშვნელობები მივიღეთ, უფრო დიდი ზომის შემთხვევაში 400×4000, 500×5000 ნაშრომში შემოთავაზებული ამოხსნები და Gurobi ამომხსნელით მიღებული მნიშვნელობები ახლოსაა ერთმანეთთან, scpd1 ამოცანისთვის უკეთესი შედეგი მივიღე GUROBI-სთან შედარებით. 1000×10000 ზომის შემთხვევაში ჩემი ალგორითმით მივიღე ამოხსნები, Gurobi ამომხსნელით კი ვერ მივიღე შედეგი (Ananiashvili, 2017).

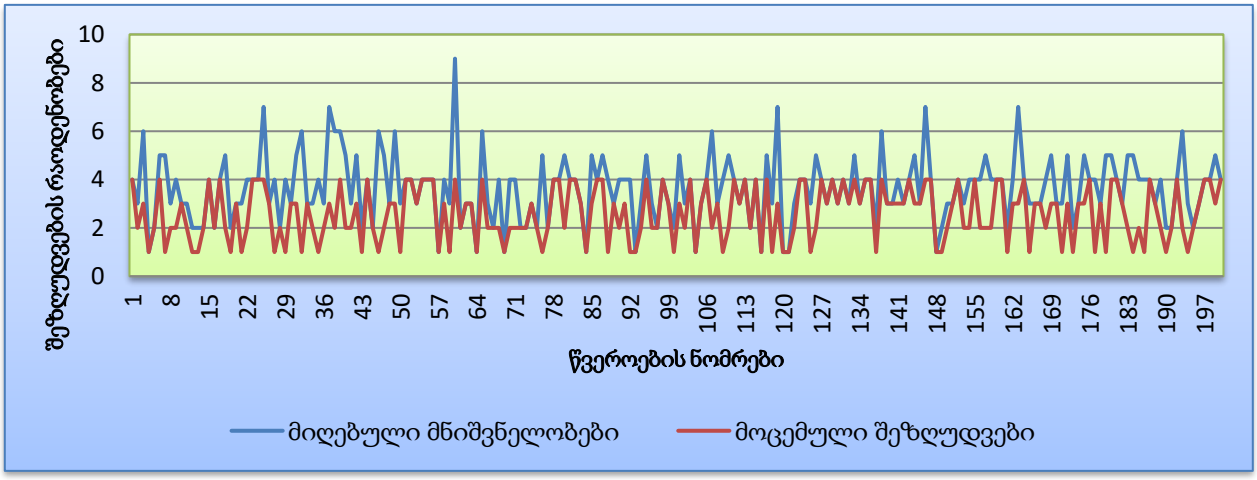
ამოხსნის შედეგები მოყვანილია ცხრილი 4.2.1.-ში, რომელშიც, f*-ით აღნიშნულია ზემოთ აღწერილი გენეტიკური ალგორითმის გამოყენებით მიღებული 10 ცდიდან საუკეთესო მნიშვნელობა, Fgurobi-ით კი Gurobi სოლვერით მიღებული მნიშვნელობა.

ცხრილი 4.2.1. ექსპერიმენტების შედეგები, დაფარვის განზოგადებული ამოცანისათვის:

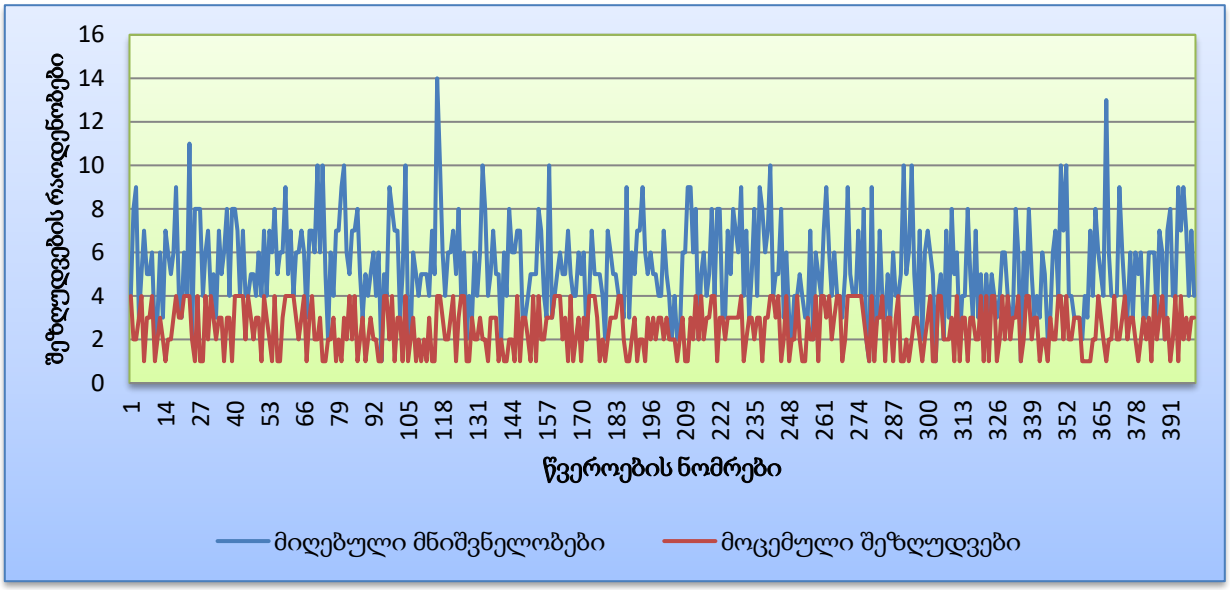
ამოცანა	f^*	იტერაციების რაოდენობა	თვლის დრო (წმ)	Fgurobi	თვლის დრო (წმ) GUROB-ით
4,1	1359	877	9.5218	1206	0.000546
4,2	1601	965	11.3247	1448	0.000571
4,3	1631	535	7.4521	1417	0.0005
5,1	841	5500	123.5824	718	0.005
5,2	851	575	10.7950	775	0.000625
5,9	959	895	7.1852	847	0.000584
6,1	360	575	10.9263	330	0.000595
6,2	381	775	10.0854	346	0.000576
6,3	440	9575	115.4195	375	0.000622
A,1	741	1575	50.0961	655	0.000584
B,1	189	874	7.5625	167	0.000533
C,1	771	885	8.2353	598	0.000633
D,1	150	875	8.9659	154	0.000569
E,1	10	1278	4.0702	9	0.000589
F,1	27	875	10.2762	25	0.000564
G,1	447	1876	524.7170	-	-
H,1	149	2276	864.1087	-	-

ტესტირება ჩატარდა სტანდარტული სიმძლავრის კომპიუტერზე მონაცემებით: Intel(R) Pentium (R) Dual CPU E2220 2.40 GHz, 2.00 GB of RAM. უმეტეს შემთხვევაში ამოცანების ამოხსნას ესაჭიროება წამები, ზოგიერთ შემთხვევაში რამდენიმე წუთი.

შევადაროთ რამდენიმე ამოცანისათვის ჩვენს მიერ შერჩეული შეზრუდვების: b_i, $i = 1, \dots, N$-ების მნიშვნელობები გამოთვლების შედეგად მიღებული დაფარვის შესაბამის $R(i)$, $i = 1, \dots, N$ -ების მნიშვნელობებს.



ნახ. 4.2.1. შედარებითი დიაგრამა scp41 ამოცანაში მოცემული შეზღუდვებისა და თვლის შედეგად მიღებული მნიშვნელობებისათვის.



ნახ. 4.2.2. შედარებითი დიაგრამა scpd1 ამოცანაში მოცემული შეზღუდვებისა და თვლის შედეგად მიღებული მნიშვნელობებისათვის.

თავი 5. ალგორითმების მუშაობის სისწრაფის შეფასება

ძალიან მნიშვნელოვანია შევაფასოთ ალგორითმები მუშაობის პროცესში რა რესურსს გამოიყენებენ - ეს იქნება შესაბამისი კოდის მიერ გამოყენებული ოპერატიული მეხსიერება თუ გამოთვლებისათვის საჭირო დრო. ალგორითმების სისწრაფის შეფასება ყოველთვის იყო აქტუალური (Garey, et al., 1979; Christofides, 1986; Minieka, 1978; Пападимитриу, и др., 1984; Cormen Thomas H., 2002; Касьянов , и др., 2003).

შემოვიღოთ პირობითი აღნიშვნები და შევაფასოთ ნაშრომში მოყვანილი ალგორითმების სისწრაფე: t_1-ით აღვნიშნოთ მინიჭების 1 ოპერაციისათვის საჭირო დრო, t_2-ით აღვნიშნოთ შედარების 1 ოპერაციისათვის საჭირო დრო, t_3-ით აღვნიშნოთ პირობითად 1 არითმეტიკული ოპერაციისათვის საჭირო დრო, τ-თი აღვნიშნოთ 1 ლოგიკური ოპერაციის (ლოგიკური "მვრა", "ან", "და" ოპერაციები) შესასრულებლად საჭირო დრო.

მეორე თავში მოყვანილია გენეტიკური ალგორითმი (ალგორითმი 2.2.1.), რომელიც რამდენიმე ალგორითმს მოიცავს, შევაფასოთ ამ ალგორითმის ის ბიჯები, რომლებზეც იხარჯება ძირითადი დრო და შემდეგ მთლიანი ალგორითმი:

P-პოპულაციების რაოდენობა, n-ამოცანის განზომილება (ცვლადების რაოდენობა), ω-იტერაციების რაოდენობა, რიცხვის კოდირებისათვის საჭირო მაქსიმალური თანრიგების რაოდენობა: $lbit = 32.$

შევაფასოთ რიცხვების კოდირებისათვის საჭირო ბიტების ზუსტი რაოდენობის განსაზღვრის დრო:

ცხრილი 5.1.1. ალგორითმი 2.2.1.-ის მუშაობის ბიჯი 3-ის სისწრაფის შეფასება:

ბიჯი	მინიჭების ოპერაციის რაოდენობა	შედარების ოპერაციის რაოდენობა	არითმეტიკული ოპერაციის რაოდენობა	ლოგიკური ოპერაცის რაოდენობა	ოპერაციის გამეორების შესაძლო რაოდენობა
3	2		3		1
3	1	3	1		$lbit$

$$T_1 = \left(2t_1 + 3t_3 + lbit \cdot (t_1 + 3t_2 + t_3)\right) \cdot P \cdot n$$

შევაფასოთ რიცხვების კოდირებისათვის საჭირო დრო:

ცხრილი 5.1.2. ალგორითმი 2.2.1.-ის მუშაობის ბიჯი 5-ის სისწრაფის შეფასება:

ბიჯი	მინიჭების ოპერაციის რაოდენობა	შედარების ოპერაციის რაოდენობა	არითმეტიკული ოპერაციის რაოდენობა	ლოგიკური ოპერაცის რაოდენობა	ოპერაციის გამეორების შესაძლო რაოდენობა
5	1		2		1
5	1	1	2		P

$$T_2 = \left(t_1 + t_3 + (t_1 + t_2 + 2t_3)\right) \cdot P \cdot n$$

ბინარულიდან გრეის კოდში გადაყვანა და პირიქით :

$$T_3 = (t_3 + \tau) \cdot P \cdot n$$

სელექციისათვის საჭირო პირობითი დრო:

$$T_4 = lbit \cdot t_2 \cdot P \cdot n$$

შეჯვარებისათვის საჭირო პირობითი დრო:

$$T_5 = t' \cdot P \cdot n$$

t'-პირობითი დროა (მინიჭებისა და ლოგიკური ოპერაციებისათვის საჭირო დროების ჯამი);

მუტაციისათვის საჭირო პირობითი დრო:

$$T_6 = lbit \cdot t' \cdot P \cdot n$$

მთლიანი ალგორითმის შესასრულებლად საჭირო დრო იქნება:

$$\begin{aligned} T = (T_1 + T_2 + T_3 + T_4 + T_5 + T_6) \cdot \omega \\ = \Big(\left(2t_1 + 3t_3 + lbit \cdot (t_1 + 3t_2 + t_3)\right) \cdot P \cdot n + \left(t_1 + t_3 + (t_1 + t_2 + 2t_3)\right) \cdot P \cdot n \\ + (t_3 + \tau) \cdot P \cdot n + lbit \cdot t_2 \cdot P \cdot n + t' \cdot P \cdot n + lbit \cdot t' \cdot P \cdot n\Big) \cdot \omega \end{aligned}$$

აქედან გამომდინარეობს:

$$T = O(P \cdot n \cdot \omega)$$

ანალოგიურ შეფასებას მივიღებთ მეორე მოდიფიცირებული ალგორითმისათვის.

მესამე თავში მოყვანილია უმცირესი დაყოფის და დაფარვის ალგორითმები, შევაფასოთ ამ ალგორითმების სისწრაფე. უმცირესი დაყოფის ამოცანის ამოსახსნელად ძირითად ალგორითმთან ერთად, რომელიც ექსპონენციალური სირთულისაა (Christofides, 1986), გამოყენებულია დამხმარე ალგორითმები: მასივების შეფუთვისთვის და განფუთვისათვის. შევაფასოთ დამხმარე ალგორითმები. განვსაზღვროთ ერთგანზომილებიანი მასივის შეფუთვის სისწრაფე.

ცხრილი 5.2. ალგორითმი 3.1.-ის მუშაობის სისწრაფე:

ბიჯი	მინიჭების ოპერაციის რაოდენობა	შედარების ოპერაციის რაოდენობა	არითმეტიკული ოპერაციის რაოდენობა	ლოგიკური ოპერაცის რაოდენობა	ოპერაციის გამეორების შესაძლო რაოდენობა
1	1		2		1
2	$[N/32]$ + 1+2				1
3	1				$[N/32] + 1$
4	1+1	1	4		$32 \cdot [N/32] + 1$
5	1+1		1	1+1	$32 \cdot [N/32] + 1$
6	1		1		$32 \cdot [N/32] + 1$

7	1		1		$[N/32]+1$
8	1	1+1			$[N/32]+1$

ამ ცხრილიდან და ზემოთ შემოტანილი აღნიშვნებიდან გამომდინარე, შეგვიძლია ვიპოვოთ ალგორითმი 3.1.-ის შესრულებისათვის საჭირო დრო. გავითვალისწინოთ, რომ ამ ალგორითმში გვაქვს ციკლში ჩადგმული ციკლები: ბიჯი 4, 5 და 6 ჩადგმულია 3-სა და ბიჯ 8-ს შორის, ეს ციკლები შესაძლოა გამეორდნენ უარეს შემთხვევაში $\left(\frac{N}{32}+1\right)$- ჯერ:

$$T_1 = t_1 + 2\cdot t_3 + \left(\frac{N}{32}+3\right)t_1 + (t_1 + (2t_1 + t_2 + 4t_3 + 2t_1 + t_3 + 2\tau + t_1 + t_3)\cdot 32 + \\ + t_1 + t_3 + t_1 + 2t_2)\cdot\left(\frac{N}{32}+2\right)$$

ამ ჯამს თუ შევაფასებთ მივიღებთ:

$$T_1 = \mathrm{O}(N)$$

რადგან ალგორითმი 3.1.-ით $1 \times N$ ზომის ვექტორის შეფუთვა ხდება, ცხადია რომ $\mathrm{N} \times M$ ზომის მატრიცის შეფუთვის სისწრაფე იქნება: $T = \mathrm{O}(N \cdot M)$

განფუთვის ალგორითმი 3.2.-ის მუშაობის სისწრაფე ისეთივე იქნება, როგორიც შეფუთვის ალგორითმის.

ამიტომ, ჩვენს მიერ განხილული ამოცანა (3.3)-(3.4) მიზნის (3.3) ფუნქციით არის NP-სრული. ამიტომ მესამე თავში მოყვანილი მთლიანი ალგორითმის სირთულე არის ექსპონენციალური.

მეოთხე თავში მოყვანილი გენეტიკური ალგორითმი წარმოადგენს ალგორითმების კომპლექსს. შევაფასოთ მასში შემავალი ალგორითმები სათითაოდ.

ცხრილი 5.3. ალგორითმი 4.1.-ის მუშაობის სისწრაფე:

ბიჯი	მინიჭების ოპერაციის რაოდენობა	შედარების ოპერაციის რაოდენობა	არითმეტიკული ოპერაციის რაოდენობა	ოპერაციის გამეორების შესაძლო რაოდენობა
1	$s\cdot M + N + 1 + 1$	M		*1*
2	$1 + N$		N	N
3	1	N		N
4	$1 + N + 1$	1	1	$(s-1)\cdot N$
5	1+1	M	N	$(s-1)\cdot N$
6	1	M		$(s-1)\cdot N$

ამ ცხრილიდან გამომდინარე, შეგვიძლია ვიპოვოთ ალგორითმი 4.2.-ის შესრულებისათვის საჭირო დრო. გავითვალისწინოთ, რომ ამ ალგორითმში ციკლში

ჩადგმული ციკლები არ გვაქვს და ბიჯები: 2, 3, 4, 5, 6 შესაძლოა გამეორდნენ უარეს შემთხვევაში N-ჯერ:

$$T_1 = (s \cdot M + N + 1 + 1) \cdot t_1 + M \cdot t_2 + \big((1 + N) \cdot t_1 + N \cdot t_3 + t_1 + N \cdot t_2\big) \cdot N + \big((1 + N + 1) \cdot t_1 + t_2 + t_3 + (1 + 1) \cdot t_1 + M \cdot t_2 + N \cdot t_3 + t_1 + N \cdot t_2\big) \cdot (s - 1) \cdot N$$

აქედან კი მარტივი გარდაქმნებით მივიღებთ, რომ ალგორითმის სისწრაფეა:

$$T_1 = \mathrm{O}(\mathrm{M} \cdot \mathrm{N} + \mathrm{N}^2 + (\mathrm{N} + M) \cdot (s - 1) \cdot \mathrm{N})$$

აღვნიშნოთ:

$$K = max(N, M) \tag{5.2}$$

გავითვალისწინოთ რომ: $s \ll K$, მაშინ საბოლოოდ მივიღებთ ალგორითმი 4.1.-ის მუშაობის სისწრაფე უარეს შემთხვევაში შეიძლება იყოს:

$$T_1 = \mathrm{O}(K^3)$$

ცხრილი 5.4. ალგორითმი 4.2.-ის მუშაობის სისწრაფე:

ბიჯი	მინიჭების ოპერაციის რაოდენობა	შედარების ოპერაციის რაოდენობა	არითმეტიკული ოპერაციის რაოდენობა	ოპერაციის გამეორების შესაძლო რაოდენობა
1	$N + 1$	M		1
2	$1 + 1$	N		N
3	$N + 1$	N	$2 \cdot N$	N
4	1	1		N
5	1	1		N

ამ ცხრილიდან გამომდინარე, შეგვიძლია ვიპოვოთ ალგორითმი 4.2.-ის შესრულე-ბისათვის საშირო დრო. გავითვალისწინოთ, რომ ამ ალგორითმში გვაქვს ციკლში ჩადგმული ციკლები: ბიჯი 3 და 4 ჩადგმულია 2 და ბიჯ 5-ს შორის, ეს ციკლები შესაძლოა გამეორდნენ უარეს შემთხვევაში N-ჯერ:

$$T_2 = (N + 1) \cdot t_1 + M \cdot t_2 + \big(2 \cdot t_1 + N \cdot t_2 + \big((N + 1) \cdot t_1 + N \cdot t_2 + t_1 + t_2\big) \cdot N + t_1 + t_2\big) \cdot N$$

აქედან კი მარტივი გარდაქმნებით მივიღებთ, რომ ალგორითმის სისწრაფეა:

$$T_2 = \mathrm{O}(\mathrm{N}^3)$$

შეჯვარების ოპერატორის განხორციელებისათვის საჭირო დრო:

1) (4.1) ფორმულების $f_k = \sum_{j=1}^{M} c_j \cdot x_j^k$, $k = 1, 2, \dots, s$ გამოთვლას დასჭირდება: $2M \cdot t_3 \cdot s$ პირობითი დრო;
2) ორი მაქსიმუმის ან ერთი მაქსიმალური და ერთი მინიმალური მნიშვნელობის პოვნას დასჭირდება: $2M \cdot t_2 \cdot s$ პირობითი დრო;
3) შვილობილი ქრომოსომების მიღებას დასჭირდება: $3N \cdot t_1$ პირობითი დრო;

შევკრიბოთ ეს დროები:

$$T_3 = 2M \cdot t_3 \cdot s + 2M \cdot t_2 \cdot s + 3N \cdot t_1$$

$$T_3 = O\big((M + N) \cdot s\big)$$

ცხრილი 5.5. ალგორითმი 4.3.-ის მუშაობის სისწრაფე:

ბიჯი	მინიჭების ოპერაციის რაოდენობა	შედარების ოპერაციის რაოდენობა	არითმეტიკული ოპერაციის რაოდენობა	ოპერაციის გამეორების შესაძლო რაოდენობა
1	$M + 1 + 1$	$M + M$		1
2		N		N
3	1	$N + M$	N	N
4		1	1	N

იგივე წესით გამოვითვლით და მივიღებთ:

$$T_4 = O(M \cdot N)$$

მუტაციის ოპერატორის განხორციელებისათვის საჭირო დრო:

1) (4.2) ფორმულების გამოთვლას დასჭირდება: $3M \cdot t_3$ პირობითი დრო;
2) (4.3) ფორმულების გამოთვლას დასჭირდება: $(1 + 2N + 1) \cdot M \cdot t_3$ პირობითი დრო;
3) მუტაციის პირობების შესრულებას დასჭირდება: $2M \cdot (t_1 + t_2 + t_1)$

მუტაციას სულ დასჭირდება:

$$T_5 = O(M)$$

ალგორითმი 4.1. მუშაობს მხოლოდ ერთხელ, ყველა დანარჩენი კი მეორდებიან იტერაციების ამოწურვამდე. ვთქვათ, იტერაციების რაოდენობა უდრის ω-ს, შევკრიბოთ T_1, T_2, T_3, T_4, T_5 იტერაციების რაოდენობების გათვალისწინებით:

$$T = K^3 + \mathrm{N}^2 + (\mathrm{N}^3 + (\mathrm{M} + \mathrm{N}) \cdot s + \mathrm{M} \cdot \mathrm{N} + \mathrm{M}) \cdot \omega$$

სადაც: $K = max(N, M)$, საბოლოოდ მივიღებთ ჩვენი გენეტიკური ალგორითმის სისწრაფეს:

$$T = \mathrm{O}(K^3 \cdot \omega)$$

როგორც ვხედავთ, მთლიანი ალგორითმი პოლინომიალური სირთულისაა.

ანალოგიურ შეფასებას მივიღებთ განზოგადებული დაფარვის ამოცანისათვის.

დასკვნა და ძირითადი შედეგები

სადისერტაციო ნაშრომში შემოთავაზებული ”მძიმე ბირთვის” ახალი უწყვეტი დაგვიანებული მოდელი - მართვადი მძიმე ბირთვი, რომელიც საინტერესოა მათემატიკური მოდელირების და რიცხვითი ოპტიმიზაციის თვალსაზრისით. მძიმე ბირთვის მეთოდის კრებადობის საკითხი დამუშავებულია გლუვი ფუნქციების შემთხვევაში. სხვა მეთოდებთან შედარებაც გლუვ ფუნქციებზე განხორციელდა. თუმცა მისი გამოყენება შესაძლებელია არაგლუვი ფუნქციებისთვისაც. ტესტირების შედეგებმა აჩვენა, რომ შემოთავაზებული - მართვადი ბირთვის ალგორითმი საიმედოდ მუშაობს და სტაბილურად აჩვენებს გაცილებით უკეთეს შედეგებს მძიმე ბირთვის კლასიკურ ვარიანტთან შედარებით.

ნაშრომში შემოთავაზებულია ოპტიმიზაციის ამოცანების ამოხსნის გენეტიკური ალგორითმები, რომლებიც დაფუძნებულია კლასიკური გენეტიკური ალგორითმის ძირითად პრინციპებზე. ახალი ალორითმების სარეალიზაციო კოდებით სატესტო ამოცანებზე მიღებულია მისაღები შედეგები. აღსანიშნავია, რომ რამდენიმე ამოცანისათვის, რომლებისთვისაც კლასიკური მეთოდები ვერ აღწევს შედეგს, განხილული ალგორითმები პოულობს ამონახსნს. ალგორითმების მუშაობის სისწრაფე შეესაბამება გენეტიკური ალგორითმებისათვის მისაღებ სისწრაფეს.

ნაშრომში განხილულია ოპტიმიზაციის კარგად ცნობილი უმცირესი დაყოფისა და დაფარვის ამოცანები, რომლებსაც გამოყენებების უაღრესად ვრცელი სპექტრი გააჩნიათ. ნაშრომში შემოთავაზებულია ამ ამოცანების ზუსტი და მიახლოებითი ამოხსნის ალგორითმები, რომლებიც საჭიროებენ ნაკლებ დროს და ნაკლებ მეხსიერებას. ამ ალგორითმებისათვის დაწერილია სარეალიზაციო პროგრამული კომპლექსები, რომლებიც გასინჯულია სატესტო ამოცანებზე. შედეგებიდან ჩანს რომ ამოხსნის დრო შეესაბამება მიახლოებითი ალგორითმებისათვის მისაღებ სისწრაფეს; ზუსტი მეთოდებით, სატესტო ამოცანების უმეტესობისთვის წამებში მიიღწევა ოპტიმალური მნიშვნელობები; მიახლოებითი მეთოდები შედეგიანად მთავრდება ყველა გასინჯულ სატესტო ამოცანაზე და თვლის დროც რამდენიმე წამიდან წუთებამდეა.

დისერტაციაში წარმოდგენილი კვლევის შედეგები ასახულია შემდეგ სამეცნიერო პუბლიკაციებში:

1. Gelashvili K., Alkhazishvili L., Khutsishvili I., Ananiashvili N., “On the modification of heavy ball method”, A. Razmadze Mathematical Institute, Ivane Javakhishvili Tbilisi State University, Tbilisi, Journal, ISSN 1512-0007, Vol. 161, 2013.
2. Ananiashvili N., "Several dynamic visualization method for unconstrained optimization" *http://conference.ens-2013.tsu.ge/uploads/50ffc040d844bnatela-ananiashvili-ENG.pdf*
3. Ananiashvili N. "About the one solution of the set partition problem",// V Annual inter¬nati-onal conference of the Georgian mathematical union". - Batumi, 2014. - p. 60.
4. Ananiashvili N. "Solution Of Problem Of Set Covering By Means Of Genetic Algorithm"- Batumi : VI Annual international conference of the Georgian mathematical union, 2015. - p. 75.

5. Ananiashvili N., "Solution of problem of set covering by means of genetic algorithm", Gesj:Computer Sciences And Telecommunications, Reviewed Electronic Scientific Journal, 2015|No.2(46):16-23.
6. Ananiashvili N., "Solution of Problem of Unconditional Optimization by Means of Genetic Algorithm". International Journal of Engineering and Innovative Technology (IJEIT) Volume 5, Issue 4, October 2015.
7. Ananiashvili N., "Solution of problems of minimal set partition and set covering". (2015)Bull. Georg. Natl. Sci., 9, 1: 38-43
8. Ananiashvili Natela About Solution of Generalized Problems of Minimal Set Covering // VII Annual international conference of the Georgian mathematical union. - Batumi , 2016. - p. 77.
9. Ananiashvili N., "Solution of a Generalized Problem of Covering by means of a Genetic Algorithm".(2017), Baltic J. Modern Computing, Vol. 5, No. 4, 351-361

ბიბლიოგრაფია

Ananiashvili N. About the one solution of the set partition problem [Conference] // V Annual inter¬nati-onal conference of the Georgian mathematical union. - Batumi : [s.n.], 2014. - p. 60.

Ananiashvili N. Solution of a Generalized Problem of Covering by means of a Genetic Algorithm [Journal] // Baltic J. Modern Computing. - 2017. - 4 : Vol. 5. - pp. 351-361.

Ananiashvili N. Solution of problem of set covering by means of genetic algorithm [Journal] // Gesj:Computer Sciences And Telecommunications. - [s.l.] : Reviewed Electronic Scientific Journal, 2015. - No.2(46).

Ananiashvili N. Solution Of Problem Of Set Covering By Means Of Genetic Algorithm [Conference]. - Batumi : VI Annual international conference of the Georgian mathematical union, 2015. - p. 75.

Ananiashvili N. Solution of Problem of Unconditional Optimization by Means of Genetic Algorithm [Journal] // International Journal of Engineering and Innovative Technology (IJEIT), ISSN: 2277-3754. - 2015. - Issue 4 : Vol. 5.

Ananiashvili N. Solution of problems of minimal set partition and set covering [Journal]. - Tbilisi : Bull. Georg. Natl. Sci., 9, 2015. - pp. 38-43. - 1.

Ananiashvili Natela About Solution of Generalized Problems of Minimal Set Covering [Conference] // VII Annual international conference of the Georgian mathematical union. - Batumi : [s.n.], 2016. - p. 77.

Balas E. Ho A. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. [Journal] // Mathematical Programming 12. - 1980. - pp. 37-60.

Balinski M. Integer programming: methods, uses, computation [Book]. - [s.l.] : Man. Sci., 1965. - p. 253. - 12.

Beaslay J. E. Jonsten K. Enhancing an algorithm for set covering problems [Journal] // Europen J. Oper. Res.. - 1992. - Vol. 58. - pp. 293-300. - N 2.

Beasley J. E. Distributing Test Problems by Electronic Mail,The Journal of the Operational Research Society [Journal] // OR-Library. - November 1990. - Vol. 41. - pp. 1069-1072. - No. 11.

Berge C. Theory of graphs and its applications [Book]. - Dunod, Paris : [s.n.], 1958.

Bhaya A., Pazos, F. and Kaszkurewicz, E The controlled conjugate gradient type trajectory-following neural net for minimization of nonconvex functions [Conference] // Proceedings of the IEEE 2010 International Joint Conference on Neural Networks (IJCNN). - Spain, Barcelona : [s.n.], 2010. - pp. 1-8.

Broderick Crawford [et al.] A Hybrid Soft Computing Approach for Subset Problems [Journal]. -[s.l.]: Hindawi Publishing Corporation. Mathematical Problems in Engineering, 2013. - Vol. 2013.

Brown G.W. and Neumann J.von Solutions of Games by Differential Equations, Contributions to the Theory of Games [Journal] // Annals of Mathematics Studies. - [s.l.] : TPrinston University Press, 1950. - pp. 73-79. - 24.

Brown G.W. Iterative solution of games by fictitious play. Activity analysis of production and allocation [Journal]. - [s.l.] : New York: Wiley, 1951. - Issue: 1 : Vol. 13. - pp. 374–376.

Cabot A. Engler H., Gadat S. On the long time behavior of second order differential equations with asymptotically small dissipation [Book]. - [s.l.] : Trans. Amer. Math. Soc., 361(11), 2009. - pp. 5983–6017.

Ching Lih Lim, Alistair Moffat and Anthony Wirth Lazy and Eager Approaches for the Set Cover Problem [Conference] // Proceedings of the Thirty-Seventh Australasian Computer Science Conference. - Auckland, New Zealand : [s.n.], 2014.

Christofides Nicos Graph Theory. An Algorithmic Approach, Computer science and applied mathematics [Book]. - London : Academic Press, 1986.

Chvatal V. A greedy heuristic for the set-covering problem. [Journal] // Mathematics of Oper. Res.. - 1979. - 3 : Vol. 4. - pp. 233-235.

Cormen Thomas H. Leiserson Charles E., Rivest Ronald L., Clifford Stein Introduction to Algorithms, Second Edition [Book]. - [s.l.] : The MIT Press Cambridge, Massachusetts London, England, 2002.

Eiselt H.A. and Sandblom C.L. The bounce algorithm for mathematical programming [Journal]. - [s.l.] : Math Meth Oper Res, 2000. - pp. 173-183. - 52.

Eremeev A.V. Modeling and analysis of genetic algorithm with tournament selection [Journal] // Artificial Evolution. - Berlin : Springer, 2000. - pp. 84-95.

Fisher M. L. Kedia P. Optimal solution of set covering/partitioning problems using dual heuristics [Journal] // Management Science 36. - 1990. - pp. 674-688.

Garey M. R. and Johnson D. S. Computers and intractability. A guide to the theory of NP-completeness [Book]. - [s.l.] : San Francisco: W. H. Freeman and Co., 1979.

Garfinkel G.L. and Nemhauser R.S. The set partitioning problem: set covering with equality constraints [Book]. - [s.l.] : Ops.Res., 1969. - p. 848. - 17.

Gelashvili K. [et al.] On the modification of heavy ball method [Journal]. - Tbilisi : A. Razmadze Mathematical Institute, Ivane Javakhishvili Tbilisi State University, 2013. - ISSN 1512-0007 : Vol. 161.

Goldberg D.E. Genetic algorithms in search, optimization, and machine learning [Book]. - [s.l.] : MA.: Addison-Wesley, 1989.

Gomory R. An algorithm for integer solutions to linear programs, Recent Advances in mathematical Programming [Book]. - New York : Graves and Wolfe,Eds., McGraw-Hill, 1963.

Goudou X. and Munier J. The gradient and heavy ball with friction dynamical systems: the quasiconvex case [Journal]. - [s.l.] : Math. Program., Ser. B, 2009. - pp. 173-19. - 116.

Grossman T. and Wool A. Computational experience with approximation algorithms for the set covering problem [Journal] // European J. Oper. Res.. - 1997. - Vol. 101. - pp. 81-92. - N1.

Guanghui Lan Gail W. DePuy , Gary E. Whitehouse An effective and simple heuristic for the set covering problem [Journal] // European Journal of Operational Research. - 2007. - Vol. 176. - pp. 1387–1403.

Hajba Tamas Optimizing second-order differential equation systems [Journal] // Electronic Journal of Differential Equations. - 2011. - Vol. 2011. - pp. 1–16. - 44.

Holland J.H. Adaptation in Natural and Artificial Systems [Book]. - [s.l.] : Ann Arbor: University of Michigan Press, 1975.

IBM ILOG CPLEX 7.0-User's Manual, ILOG [Book]. - California : View, Mountain, 2000.

Jacobs L. W. and Brusco M.J. A simulated annealing-based heuristic for the set covering problem [Journal]. - [s.l.] : Working paper, Operations Management and Infprmation Systems Department, Northern Illinois University, Dekalb, IL 60115, USA, 1993.

Jorge J. More, Burton S. Garbow and Kenneth E. Hillstr Testing Unconstrained optimiza-tion software. [Journal] // ACM Transactions on Mathematical Software. - 1981. - Vol. 7. - No 1.

Kong Seunghyun Linear Programming Algorithms Using Least-squares Method [Journal]. - [s.l.] : Georgia Institute of Technology, 2007.

Minieka E. Optimization Algorithms For Networks And Graphs [Book]. - New York : M. Dekker, 1978.

Nikhil Bansal, Anupam Gupta and Ravishankar Krish Generalized min sum set cover [Conference] // In ACM-SIAM Symposium on Discrete Algorithms. - 2010.

Paul-Emile Maingé Asymptotic convergence of an inertial proximal method for unconstrained quasiconvex minimization [Journal] // Journal of Global Optimization. - December 2009. - Issue 4 : Vol. 45.

Pierce J.F. Application of combinatorial programming to a class of all-zero-one integer programming problems [Book]. - [s.l.] : Man. Sci., 1968. - p. 191. - 15.

Polyak Boris T. Introduction to Optimization. Optimization Software [Book]. - New York : Translated from the Russian. With a foreword by Dimitri P. Bertsekas. Translations Series in Mathematics and Engineering. Optimization Software, Inc., Publications Division, 1987.

Press W.H. [et al.] Numerical Recipes [Book]. - Cambridge University Press : [s.n.], 2007.

Randy L. Haupt Sue Ellen Haupt "Practical genetic algorithms"-2nd ed. [Book]. - New Jersey : Published by John Wiley & Sons, Inc., Hoboken, 2004.

Ravindran A., Ragsdell K. M. and Reklaitis G. V. Engineering Optimization: Methods and Applications, Second Edition [Journal]. - New Jersey : Publisher: John Wiley & Sons, Inc., Hoboken, 2006.

Soo Y. Chang and Katta G. Murty The steepest descent gravitational method for linear programming [Journal] // Discrete Applied Mathematics . - 1989. - pp. 211-239. - 25.

Umetani S., Arakawa M. and Yagiura M. A heuristic algorithm for the set multicover problem with generalized upper bound constraints. [Conference] // In Proceedings of Learning and Intelligent Optimization Conference (LION). - 2013. - pp. 75–80.

Yang J and Leung J.Y.T. A generalization of the weighted set covering problem [Journal] // Naval Research Logistics. - 2005. - Vol. 52(2). - pp. 142–149.

Yossi Azar, Iftah Gamzu and Xiaoxin Yin Multiple intents re-ranking [Conference] // Proceedings of the41st Annual ACM Symposium on Theory of Computing. - New York : NY,USA,. ACM, 2009. - pp. pages 669-678.

Гладков Л. А., Курейчик В. В. и Курейчик В. М. Генетическме алгоритмы [Книга]. - [б.м.] : Ростов на дону, 2004. - Учебное пособие.

Есипов Б.А. и Муравьев В.В. Исследование Алгоритмов Решения Обобщенной Задачи О Минимальном Покрытии [Журнал] // Известия Самарского научного центра Российской академии наук, т. 16, №4(2). - 2014 г..

Касьянов В.Н. и Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение [Книга]. - Санкт-Петербург : БХВ-Петербург, 2003.

Нечепуренко М.И., Попков В.К. и Майнагашев С.М. Алгоритмы и прграммы решения задач на графах и сетях [Книга]. - Новосибирск : Наука, 1990.

Панченко Т.В. Генетическме алгоритмы [Книга]. - [б.м.] : издательский дом "Астраханский универ-ситет", 2007.

Пападимитриу Х. и Стайглиц К. Комбинаторная Оптимизация, Алгоритмы и слож-ность [Книга]. - Москва : (перевод с англ.), изд. «МИР», 1984.

Рутковская Д., Пилинский М. и Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы [Книга]. - Москва : Горячая линия-Телеком, 2006.