BULUT GÖREV ÇİZELGELEMESİ İÇİN
BENZETİLMİŞ TAVLAMA TABANLI
BİR OPTİMİZASYON YAKLAŞIMI

Esra ÇELİK

Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Dr. Öğr. Üyesi Deniz DAL
2018

ATATÜRK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

BULUT GÖREV ÇİZELGELEMESİ İÇİN BENZETİLMİŞ TAVLAMA TABANLI BİR OPTİMİZASYON YAKLAŞIMI

Esra ÇELİK

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ERZURUM
2018

T.C.
ATATÜRK ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü Müdürlüğü

TEZ ONAY FORMU

BULUT GÖREV ÇİZELGELEMESİ İÇİN BENZETİLMİŞ TAVLAMA TABANLI BİR OPTİMİZASYON YAKLAŞIMI

Dr. Öğr. Üyesi Deniz DAL danışmanlığında, Esra ÇELİK tarafından hazırlanan bu çalışma,12/09/2018 tarihinde aşağıdaki jüri tarafından Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Bilim Dalı'nda Yüksek Lisans tezi olarak **oybirliği / oy çokluğu (3./0)** ile kabul edilmiştir.

Başkan: Dr. Öğr. Üyesi Deniz DAL İmza :

Üye : Dr. Öğr. Üyesi Tolga AYDIN İmza :

Üye : Dr. Öğr. Üyesi Mustafa CANIM İmza :

Yukarıdaki sonuç;

Enstitü Yönetim Kurulu'nun 08/11/2018 tarih ve ..44.../..11..... nolu kararı ile onaylanmıştır.

Prof. Dr. Mehmet KARAKAN
Enstitü Müdürü

Not: Bu tezde kullanılan özgün ve başka kaynaklardan yapılan bildiriş, çizelge, şekil ve fotoğrafların kaynak olarak kullanımı, 5846 sayılı Fikir ve Sanat Eserleri Kanunundaki hükümlere tabidir.

ÖZET

Yüksek Lisans Tezi

BULUT GÖREV ÇİZELGELEMESİ İÇİN BENZETİLMİŞ TAVLAMA TABANLI BİR OPTİMİZASYON YAKLAŞIMI

Esra ÇELİK

Atatürk Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Deniz DAL

Her geçen gün ilerleyen teknoloji, kapasitesi hızla artan ve geleneksel algoritmalar/donanımlar ile işlenemeyen büyük veriyi beraberinde getirmektedir. Bu verinin depolanması, makul sürelerde işlenebilmesi ve analiz edilmesi için de dağıtık büyük veri kümelerine ihtiyaç duyulmaktadır. Bu tür veri kümelerine ise bulut bilişim hizmeti veren altyapılarda sıklıkla rastlanmaktadır. Görev çizelgeleme, bu veriyi analiz etmek için kullanılacak görevlerin tamamının söz konusu küme düğümleri (sunucuları) üzerinde en kısa sürede işletilmesine imkân verecek görev-sunucu eşleştirmesi işleminin adıdır. Başka bir deyişle çizelgeleme, NP-hard olarak da kategorize edilen ve global minimum arayan bir optimizasyon problemidir. Dolayısıyla bu problemin optimuma yakın (eğer mümkünse optimum) değerleri makul sürelerde üretebilecek şekilde çözüme kavuşturulması için metasezgisel yaklaşımların yardımına ihtiyaç duyulmaktadır.

Bu tez kapsamında görev çizelgeleme için benzetilmiş tavlama tabanlı bir metasezgisel yaklaşım geliştirilmiştir ve bu yaklaşımın seri ve paralel versiyonları C++ programlama dili kullanılarak bir bilgisayar programına dönüştürülmüştür. Paralel versiyon için aynı zamanda OpenMP kütüphanesinden faydalanılmıştır. Benzetilmiş tavlama isimli metasezgisel, esin kaynağını metalurji biliminden almaktadır. Yüksek sıcaklıklara kadar ısıtılan metaller rastgele sıvı hale dönüşmekte ve uygun bir şekilde yavaşça soğutulduklarında düzenli bir kristal yapıya kavuşmaktadır. Bu gözlemden esinlenen benzetilmiş tavlama yöntemi de eldeki problemi temsil eden rastgele bir ilk çözümü yüksek sıcaklıklardan başlayarak ve bu sıcaklığı her adımda yavaşça azaltarak arzulanan global çözüme yaklaştırmayı hedeflemektedir.

Geliştirilen yaklaşımın yetkinliği görev çizelgeleme algoritmalarının performanslarını karşılaştırmak için kullanılan ve Braun modeliyle oluşturulmuş on iki meşhur benchmark ile test edilmiştir. Önerilen yaklaşımın hem seri hem de paralel versiyonu, tüm benchmarklar için, literatürde şu ana kadar bir sezgisel veya metasezgisel kullanılarak rapor edilen en iyi gecikme değerlerinden daha iyi sonuçları 90 saniye kısıtı içerisinde üretebilmiştir. Geliştirilen bilgisayar programının çalışma süresinin azaltılması ve üretilen çözümlerin kalitesinin arttırılması için, benzetilmiş tavlamanın ihtiyaç duyduğu farklı rastgele sayı üretme ve pertürbasyon teknikleri, veri yapıları, döngü sonlandırma koşulları, keşif-sömürme oranları ve derleyici etkisi yine bu tez kapsamında detaylı olarak analiz edilmiştir.

2018, 128 sayfa

Anahtar Kelimeler: Bulut Bilişim, Görev Çizelgeleme, Benzetilmiş Tavlama, Metasezgisel, Paralel Hesaplama

ABSTRACT

Master Thesis

A SIMULATED ANNEALING-BASED OPTIMIZATION APPROACH FOR CLOUD TASK SCHEDULING

Esra ÇELİK

Atatürk University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Asst. Prof. Dr. Deniz DAL

The technology advancing with each passing day brings a huge data along the way that grows very rapidly and cannot be processed with traditional algorithms/hardware. Storing, processing and analyzing these data in a reasonable amount of time require distributed data clusters. Such data clusters are frequently encountered in infrastructures that provide cloud services. Task scheduling is the name of the task-server mapping process that will allow all of the tasks to be used to analyze this data to be run on those cluster nodes (servers) as soon as possible. In other words, scheduling is an optimization problem categorized as NP-hard and seeking global minimum. Therefore, there is a need for metaheuristic approaches to solve this problem in such a way that it can produce near-optimum (if possible optimal) values at reasonable times.

In this thesis, a simulated annealing based metaheuristic approach was developed for task scheduling and serial and parallel versions of this approach were transformed into a computer program using C ++ programming language. For the parallel version, the OpenMP library was also used. The metaheuristic called simulated annealing takes its inspiration from metallurgical science. Metals heated up to high temperatures are converted into a liquid state and, if properly cooled slowly, have a regular crystal structure. The simulated annealing method, inspired by this observation, aims at bringing a first initial solution to a desired global solution, starting at high temperatures and gradually decreasing this temperature at every step.

The effectiveness of the proposed approach has been tested by twelve famous benchmarks, which are formed by the Braun model to compare the performance of task scheduling algorithms. Both the serial and the parallel version of the proposed approach were able to produce better results for all benchmarks than the best latency values reported in the literature within the time constraint of 90 seconds. In order to decrease the execution time of the developed computer program and to improve the quality of the solutions produced, different random number generation and perturbation techniques, data structures, early termination conditions, exploration-exploitation rates and compiler effect required by simulated annealing were also analyzed in detail within the scope of this thesis.

2018, 128 pages

Keywords: Cloud Computing, Task Scheduling, Simulated Annealing, Metaheuristic, Parallel Computing

TEŞEKKÜR

Yüksek lisans tez danışmanlığımı üstlenen, tez konusu seçiminde isteklerimi göz önünde bulunduran, çalışmanın planlanmasından oluşumuna kadar kıymetli zamanını ayırarak sabırla ve büyük bir ilgiyle engin bilgi ve tecrübelerini benimle paylaşan, çalışmamı bilimsel temeller ışığında şekillendiren, her sorun yaşadığımda yanına çekinmeden gidebildiğim, güler yüzünü ve samimiyetini benden esirgemeyen ve gelecekteki mesleki hayatımın her aşamasında bana verdiği değerli bilgilerden faydalanacağıma inandığım kıymetli hocam Sayın Dr. Öğr. Üyesi Deniz DAL'a sonsuz teşekkürlerimi sunarım.

Bilgi ve deneyimleri ile yol göstererek, değerli görüşlerini paylaşan Atatürk Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü öğretim üyelerinden Sayın Dr. Öğr. Üyesi Tolga AYDIN'a ve Erzurum Teknik Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü öğretim üyelerinden Sayın Dr. Öğr. Üyesi Mustafa CANIM'a teşekkürlerimi sunarım.

Ayrıca sevgi ve saygı çerçevesinde beni yetiştirerek bugünlere getiren, hayatım boyunca maddi ve manevi hiçbir desteği benden esirgemeyen, eğitim hayatımda ideallerime ulaşmayı başarabileceğim konusunda bana cesaret katan, yaşadığım her anı değerli kılan ve bu hayattaki en büyük şansım olan aileme sonsuz teşekkürler.

Esra ÇELİK
Eylül, 2018

İÇİNDEKİLER

SİMGELER ve KISALTMALAR DİZİNİ

CLS	Bulut Listesi Çizelgeleme (*Cloud List Scheduling*)
CHC	Çapraz Heterojen Katalizmik (*Cross Heterogeneous Cataclysmic*)
cMA	Hücresel Memetik Algoritma (*Celluler Memetic Algorithm*)
CMMS	Bulutta Min-Min Çizelgeleme (*Cloud Min-Min Scheduling*)
CPR	Yeniden Başlatma ile Sütun Fiyatlandırması (*Column Pricing with Restarts*)
CPD	Yokuş Aşağı Sütun Fiyatlandırması (*Column Pricing Downhill*)
CTPS	Bulut Görev Bölümleme Çizelgeleme (*Cloud Task Partitioning Scheduling*)
DAS	Doğrudan Bağlı Depolama (*Direct Attached Storage*)
ETC	Beklenen Hesaplama Süresi (*Expected Time Compute*)
GA	Genetik Algoritma (*Genetic Algorithm*)
GCC	GNU Derleyici Koleksiyonu (*GNU Compiler Collection*)
HCSP	Heterojen Hesaplama Çizelgeleme Problemi (*Heterogeneous Computing Scheduling Problem*)
HDFS	Hadoop Dağıtık Dosya Sistemi (*Hadoop Distributed File System*)
HUX	Yarım Aynı Çaprazlama (*Half Uniform Crossover*)
LB	Alt Sınır (*Lower Bound*)
LJFR	En Hızlı Kaynak için En Uzun İş (*Longest Job to Fastest Resource*)
LMCTS	Yerel Minimum Tamamlama Süresi Değişimi (*Local Minimum Completion Time Swap*)
LTH	Yerel Tabu Sıçraması (*Local Tabu Hop*)
MA	Memetik Algoritma (*Memetic Algorithm*)
MEMAX	Medyan MAX (*MEdian MAX*)
MCT	Minimum Tamamlama Süresi (*Minimum Completion Time*)
MCC	Bulutta Minimum Tamamlama Çizelgesi (*Minimum Completion Cloud Scheduling*)
NAS	Ağa Bağlı Depolama (*Network Attached Storage*)
NSGA-II	Baskın Olmayan Sıralama Genetik Algoritması (*Non-Dominated Sorting Genetic Algorithm*)

PALS	Problemden Haberdar Yerel Arama (*Problem Aware Local Search*)
p-CHC	Paralel CHC (*Parallel Cross Heterogeneous Cataclysmic*)
pμ -CHC	Paralel Mikro CHC (*Parallel Micro Cross Heterogeneous Cataclysmic*)
PGA	Paralel Genetik Algoritma (*Parallel Genetic Algorithm*)
SA	Benzetilmiş Tavlama (*Simulated Annealing*)
SAN	Depolama Alan Ağı (*Storage Area Network*)
SBTS	Düzgünleştirme Tabanlı Görev Çizelgeleme (*Smoothing Based Task Scheduling*)
SGA	Mücadele Genetik Algoritması (*Struggle Genetic Algorithm*)
SJFR	En Hızlı Kaynağa En Kısa İş Görev (*Shortest Job to Fastest Resource*)
TPB	Zorlu Ceza Tabanlı (*Tenacious Penalty Based*)
TS	Tabu Arama (*Tabu Search*)
TSTT	İki Aşamalı Görev Transferi (*Two Stages Tasks Transfer*)
μ-GA	Mikro Genetik Algoritma (*Micro Genetic Algorithm*)

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Gün geçtikçe ilerleyen teknoloji ile birlikte bilgisayar ve mobil cihaz kullanımının yaygınlaşması, depolama alanlarının ucuzlaması ve yüksek kapasiteli ağların kullanım alanlarının artışı beraberinde büyük boyutlu verileri getirmiştir. Giderek artan bu büyük çaplı verilerin ihtiyaca bağlı olarak en kısa sürede işlenmesi ve kullanıcıya sunulması gerekliliği bu gelişmelerin bir sonucu olarak karşımıza çıkmaktadır. Bu ihtiyaca yönelik bir kavram olarak 1950'li yıllarda ortaya çıkan, 1994 yılına kadar interneti temsil amaçlı kullanılan ve 2006 yılında bilişim hizmeti olarak kabul edilen bulut, kendi bünyesinde bilişim hizmeti veren alt bileşenleri sayesinde eldeki büyük çaplı verinin analiz edilmesinden depolanmasına kadar gerekli olan tüm altyapıyı sağlamaktadır.

Artan ihtiyaçlara karşı kısa sürede ve doğru bir şekilde dönüş yapılmasına olanak sağlayan bulut bilişim hizmeti veren altyapılar, mevcut büyük veriyi bilgiye dönüştürme sürecinde arka planda çok sayıda yöntemi bir arada kullanmaktadır. Bu yöntemler arasında var olan kaynakların etkin bir şekilde kullanılmasına yönelik veri-bilgi dönüşümünü sağlayan en temel yöntem sıkı bir görev çizelgeleme yapısından geçmektedir.

Görev çizelgeleme, büyük veriyi analiz etmek için kullanılacak görevlerin tamamının söz konusu küme düğümleri (sunucuları) üzerinde en kısa sürede işletilmesine imkân verecek görev-sunucu eşleştirmesi işleminin adıdır. Kullanılacak görev ve düğüm sayısına bağlı olarak çok sayıda farklı çizelgeleme oluşturmak mümkündür. Örneğin T görev sayısı, N ise görevlerin dağıtılacağı düğüm sayısı olmak üzere tüm muhtemel kombinasyonlar dahilinde (N^T) farklı görev çizelgeleme yapılabilmektedir. Aşağıda okuyucuya bir fikir vermesi açısından farklı N'ler ve T'ler için (Wolfram|Alpha 2018)'dan alınan sonuçlar ile çözüm uzayının büyüklüğüne ait birkaç örnek paylaşılmıştır.

$8^{32} =$ 79228162514264337593543950336 (29 basamak)

$8^{64} =$ 627710173538668076383578942320766641610235544446403451289 6 (58 basamak)

16^{128} = 134078079299425970995740249982058461274793658205923933777235614
437217640300735469768018742981669034276900318581864860508537538
828119465699464336490060840 96 (155 basamak)

16^{256} = 179769313486231590772930519078902473361797697894230657273430081
157732675805500963132708477322407536021120113879871393357658789
768814416622492847430639474124377767893424865485276302219601246
094119453082952085005768838150682342462881473913110540827237163
350510684586298239947245938479716304835356329624224137216
(309 basamak)

16^{512} = 323170060713110073007148766886699519604441026697154840321303454
275246 551388678908931972014115229134636887179609218980194941195
591504909210950881523864482831206308773673009960917501977503896
521067960576383840675682767922186426197561618380943384761704705
816458520363050428875759154106580860755239912393038552191433338
966834242068497478656456949485617603532632205807780565933102619
270846031415025859286417711672594360371846185735759835115230164
590440369761323328723122712568471082020972515710172693132346967
854258065669793504599726835299863821552516638943733554360213543
322960464531847860495214819355585361105959623065 6 (617 basamak)

16^{1024} = 104438888141315250669175271071662438257996424904738378038423348
328395390797155745684882681193499755834089010671443926283798757
573438185793607263236087851365277945956976543709998340361590134
383718314428070011855946226376318839397712745672334684344586617
496807908705803704071284048740118609114467977783598029006686938
976881787785946905630190260940599579453432823469303026696443059
025015972399867714215541693835559885291486318237914434496734087
811872639496475100189041349008417061675093668333850551032972088
269550769983616369411933015213796825837188091833656751221318492
846368125550225998300412344784862595674492194617023806505913245
610825731835380087608622102834270197698202313169017678006675195
485079921636419370285375124784014907159135459982790513399611551
794271106831134090584272884279791554849782954323534517065223269
061394905987693002122963395687782878948440616007412945674919823
050571642377154816321380631045902916136926708342856440730447899
971901781465763473223850267253059899795996090799469201774624817
718449867455659250178329070473119433165550807568221846571746373
296884912819520317457002440926616910874148385078411929804522981
857338977648103126085903001302413467189726673216491511131602920
781738033436090243804708340403154190336 (1234 basamak)

Elde edilen çözüm uzayı büyüklüklerinden yola çıkarak doğru çizelgelemeye kaba kuvvete (*brute force*) dayalı algoritmalarla polinomsal zamanda ulaşmanın mümkün olmadığı, dolayısıyla çözümü makul zamanda üretecek bir yaklaşıma ihtiyaç duyulduğu açıkça görülmektedir. Çünkü yapılacak görev çizelgeleme, NP-hard olarak da kategorize edilen ve global minimum arayan bir optimizasyon problemidir. Dolayısıyla bu problemin optimuma yakın (eğer mümkünse optimum) değerleri makul sürelerde üretebilecek şekilde çözüme kavuşturulması için metasezgisel yaklaşımların yardımına ihtiyaç duyulmaktadır.

Bu tez kapsamında görev çizelgeleme için benzetilmiş tavlama tabanlı bir metasezgisel yaklaşım geliştirilmiştir ve bu yaklaşımın seri ve paralel versiyonları C++ programlama dili kullanılarak bir bilgisayar programına dönüştürülmüştür. Paralel versiyon için aynı zamanda OpenMP kütüphanesinden faydalanılmıştır. Benzetilmiş tavlama isimli metasezgisel, esin kaynağını metalurji biliminden almaktadır. Yüksek sıcaklıklara kadar ısıtılan metaller rastgele sıvı hale dönüşmekte ve uygun bir şekilde yavaşça soğutulduklarında düzenli bir kristal yapıya kavuşmaktadır. Bu gözlemden esinlenen benzetilmiş tavlama yöntemi de eldeki problemi temsil eden rastgele bir ilk çözümü yüksek sıcaklıklardan başlayarak ve bu sıcaklığı her adımda yavaşça azaltarak arzulanan global çözüme yaklaştırmayı hedeflemektedir.

Geliştirilen yaklaşımın yetkinliği görev çizelgeleme algoritmalarının performanslarını karşılaştırmak için kullanılan ve (Anonymous 2012)'den alınan Braun modeliyle oluşturulmuş on iki meşhur benchmark ile test edilmiştir. Önerilen yaklaşımın hem seri hem de paralel versiyonu, tüm benchmarklar için, literatürde şu ana kadar bir sezgisel veya metasezgisel kullanılarak rapor edilen en iyi gecikme değerlerinden daha iyi sonuçlar üretmiştir. Ayrıca önerilen yaklaşımda literatürde sezgisel veya metasezgisel yaklaşımlara ait rapor edilen çalışma süresi için kullanılan maksimum 90 saniye kısıtlaması da dikkate alınmıştır. Öte yandan literatürde yer alan ve şimdiye kadar bulunmuş en iyi çözümleri rapor eden çalışmada kalburüstü sunuculardan oluşan bir hesaplama kümesi kullanılmasına rağmen bu çalışmada özel bir donanımdan ya da paralel hesaplama kümesinden faydalanılmamıştır; aksine test ortamı olarak herhangi bir son

kullanıcının rahatlıkla erişebildiği bir dizüstü bilgisayardan faydalanılmıştır. Geliştirilen bilgisayar programının çalışma süresinin azaltılması için, benzetilmiş tavlamanın ihtiyaç duyduğu farklı rastgele sayı üretme ve pertürbasyon teknikleri, veri yapıları, döngü sonlandırma koşulları, keşif-sömürme oranları ve derleyici etkisi yine bu tez kapsamında detaylı olarak incelenmiştir.

Bu tezin geri kalan bölümü ise şu şekilde organize edilmiştir. 1. bölümde temel bazı kavramlar açıklığa kavuşturulmuştur. 2. bölümde kaynak özetlerine yer verilmiştir. Materyal ve yöntem 3. bölümde incelenmiştir. 4. bölümde araştırma bulguları tartışılmıştır. 5. ve son bölümde sonuçlar detaylı grafikler eşliğinde analiz edilmiştir.

1.1. Veri Nedir

Veri bilgiye dönüşmek için değerlendirilmeye ihtiyaç duyan bir yapıdır. Bu sebeple veri analiz edilerek bir anlam kazandırılmalıdır. (Erbay ve Kör 2016)'e göre veri gerçeklerin, fikirlerin ve komutların insan veya makine tarafından iletişim, yorumlama veya işlemeye uygun olarak belli bir düzende soyutlanmasıdır.

1.2. Büyük Veri Nedir

Her geçen gün küçük, orta ve büyük çaplı sistemler tarafından üretilen oldukça çeşitli, aynı zamanda üretim hızı yüksek olan bir veri topluluğu hayatımıza katılmaktadır. Geleneksel algoritmalar/donanımlar ile işlenemeyecek durumda olan bu ham veri topluluğu büyük veri olarak isimlendirilmektedir. Bu veri topluluğunun analiz edilmesinden, bir anlam kazanmasına kadar geçen süreç hızla büyüyen veri yığınının getirdiği başlıca sorunlar arasında yer almaktadır. (Patel *et al.* 2012)'e göre büyük veri yoğun veri teknolojilerinin kullanımının artışını sağlayan bir toplumda ortaya çıkmakta ve yönetiminde yapılandırılmamış veriler, erişilebilirlik, gerçek zamanlı analiz, hata toleransı gibi çeşitli zorlukları beraberinde getirmektedir.

1.2.1. Büyük verinin oluşumu

Büyük verinin oluşumuna yönelik çok sayıda temel unsur sıralamak mümkündür. Bunlar arasında en çok dikkat çeken teknolojik cihazların kullanımına ve sınırsız internet erişimine bağlı olarak;

- sosyal medya uygulamaları,
- online mesajlaşma uygulamaları,
- bu uygulamalara bağlı olarak anlık fotoğraf ve video paylaşımlarındaki hızlı artış,
- GPS teknolojisi ile konuma bağlı veri elde edilmesi gibi unsurlar yer almaktadır.

2012 yılında, The Human Face of Big Data'nın büyük miktarda veriyi gerçek zamanlı olarak toplayan, görselleştiren ve analiz eden küresel proje istatistiklerine göre;

- Facebook'un 70 dil, 140 milyar fotoğraf yükleme, 125 milyar arkadaş bağlantısı, her gün 30 milyar içerik ve 2,7 milyar beğeni ve yorum kullanılarak 955 milyon aylık aktif hesap yayınladığı, her dakika 48 saat video yüklendiği ve her gün YouTube'da 4 milyar görüntüleme gerçekleştirildiği belirtilmiştir (Sagiroglu and Sinanc 2013).

1.3. Bulut Bilişim (Cloud Computing)

Geniş çevrelerde ortaya çıkışı Amerikalı bilgisayar bilimcisi John McCarthy'nin 1960'larda "Bir gün hesaplama işlemleri geniş kamusal ağlar üzerinde gerçekleşecek" görüşüne dayandırılan bulut bilişim, internet üzerinden erişimde bulunulan yazılım uygulamaları, veri depolama hizmetleri ve işlem kapasitesi olarak tanımlanmaktadır.

Bulut bilişim, büyük ölçekli ve karmaşık hesaplamalar yapmaya imkan veren güçlü bir teknolojidir. Büyük veri ile bulut bilişim birbiriyle bağlantılıdır. Büyük verinin ele alınması zorlu ve zaman isteyen bir süreçtir. Verinin birden fazla veri kümesine dağıtılması, işlenmesi ve sonuç kümelerinin zamanında iade edilmesi için büyük bir hesaplama altyapısı gerekmektedir. Bulut bilişim bunu büyük ölçüde altyapısında yer

alan dağıtık veri işleme platformlarından oluşan bir sınıf olan Hadoop'un kullanımıyla sağlamaktadır (Hashem *et al.* 2014).

1.3.1. Bulut altyapısı

Bulut bilişim ön kısım ve arka kısım olmak üzere iki ana bileşenden oluşmaktadır. Bu iki bileşen birbirine bir ağ ile bağlıdır. Ön kısım bilgisayar kullanıcısı veya müşteri tarafı, arka kısım ise sistemin bulut kısmıdır ve bulutun altyapısını oluşturmaktadır. Şekil 1.1'de görüldüğü gibi ön kısımda kullanıcı bilgisayarları, buluta erişim için gerekli uygulamalar yer alırken arka kısım ise çok sayıda bilgisayar, sunucu ve veri depolama birimini içermektedir.

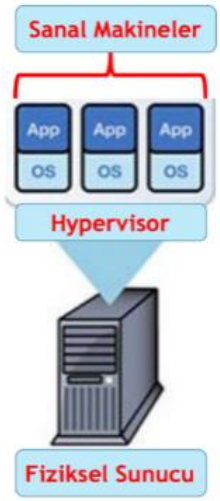
Şekil 1.1. Bulut altyapısı

Bulutun altyapısında sanallaştırma teknolojisi kullanılmaktadır. Sanallaştırma fiziksel bir sunucuyu her biri kendi bağımsız işletim sistemini çalıştıran çoklu sunucularmış gibi kullanma işlemidir. 1970'li yıllarda IBM firması tarafından hayatımıza giren daha sonra Intel ve AMD işlemcilerine eklenen ve günümüzde kurumların sistem odalarında uygulanmaya devam eden bir teknolojidir. Bu yapı üzerinde kullanılan sanal sunucuların her biri sanal makine olarak adlandırılmaktadır.

(Hashem *et al.* 2014) dağıtılmış bilgi işlem bileşenlerine büyük bir veri ortamında erişmek, depolamak, analiz etmek ve yönetmek için gereken birçok platform özniteliğinin temelinin sanallaştırma yoluyla sağlandığını ifade etmiştir.

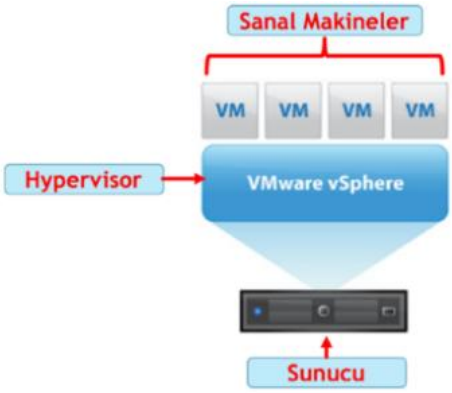
Sanallaştırma kavramı kaynak kullanımı ve paylaşımı anlamında, başta donanım maliyetlerininin düşürülmesi ve sistem kaynaklarının verimli kullanımının sağlanması noktasında çok önemli bir yer tutmaktadır. Günümüzde yapılan veri merkezi konsolidasyonlarında en öncelikli olarak değerlendirilen sanallaştırma teknolojileri, bulut bilişim kavramının temelini oluşturmaktadır (Okutucu, 2012).

Fiziksel bir sunucu üzerinde sanallaştırma yapılabilmesine imkân sağlayan yapı hipervizör (hypervisor) katmanıdır. Hipervizör katmanı sunucunun temel fiziksel kaynaklarını verimli şekilde kullanmasına olanak tanımaktadır. Bu olanağı da kapasitesi yüksek sunucular içerisine sanal olarak aynı sistemleri kurabilme imkânı veren yazılımlarla sağlamaktadır. Böylelikle hem maliyet hem de alan açısından büyük bir avantaj elde edilmektedir. (Okutucu, 2012)'ya göre hipervizör katmanı fiziksel sunucunun sahip olduğu kaynakların, bu sunucu üzerinde yapılandırılacak sanal sunuculara istenilen miktarda atanması için donanım ve sanal işletim sistemleri arasında kullanılmaktadır. Sanallaştırma ve hipervizör yapısı Şekil 1.2'de gösterilmiştir.



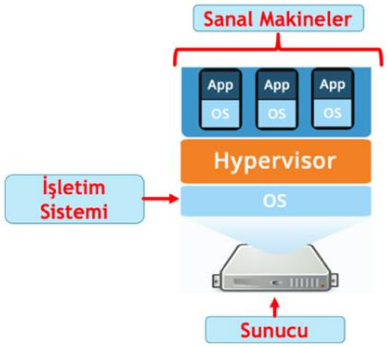
Şekil 1.2. Sanallaştırma ve hipervizör katmanı

Hipervizör katmanı sisteme iki farklı şekilde entegre edilmektedir. İlki direk donanım üzerine kurulmasını, diğeri ise bir işletim sistemi üzerine kurulmasını içermektedir. Şekil 1.3'de görüldüğü gibi hipervizör katmanının direk donanım üzerine kurulduğu durumlarda sanal makineler de bu katmanın üzerine kurulmaktadır. Sanal makinelerin donanım üzerinde çalışmaları sebebiyle fiziksel sunucunun kapasitesinin tamamından yararlanılmaktadır. Yoğun çalışma ortamlarında tercih edilen bu yapıya uygun yazılımlar arasında Microsoft Hyper-V ve VMware vSphere yer almaktadır.



Şekil 1.3. Direk donanım üzerine kurulu hipervizör yapısı

Hipervizör katmanının çalışan bir işletim sistemi üzerine kurulduğu durumlarda ise yazılım işletim sistemi üzerinde çalışmaktadır. Fiziksel sunucunun kapasitesinden tam olarak yararlanılmamakla birlikte basit çalışma ortamlarında kullanılan bu yapıya uygun yazılımlar arasında Microsoft Virtual Server ve Microsoft Virtual PC yer almaktadır. Şekil 1.4'de işletim sistemi üzerine kurulu hipervizör yapısı gösterilmiştir.



Şekil 1.4. İşletim sistemi üzerine kurulu hipervizör yapısı

1.3.2. Depolama birimleri

Geçmiş yıllarda birçok bilgisayarın tek bir sabit sürücüye sahip olduğu bilinmektedir. Zamanla eldeki verilerin böyle bir sabit sürücüye sığmadığı durumlarla karşılaşılınca harici sürücülere gereksinim duyulmuştur. Ancak yıllar içerisinde harici disklerin de depolama ihtiyacını karşılayamaz duruma geldiği ve bu durum için farklı depolama birimlerinin ortaya çıktığı görülmektedir. Bu depolama birimleri aşağıda yer almaktadır.

- DAS
- NAS
- SAN

DAS fiziksel sunucu üzerinde bulunan veya sunucuya doğrudan bağlı olan bir depolama birimidir. Sunucuya doğrudan bağlantı harici bir kablo ile yapılmaktadır. Diğer birimlerden en temel farkı depolama için herhangi bir ağa ihtiyaç duyulmamasıdır. (Hashem *et al.* 2014)'na göre DAS küçük çapta birbirine bağlı sunucular için uygundur.

NAS doğrudan ağ üzerine yerleştirilmekte ve ihtiyaç duyulduğunda Ethernet ya da kablosuz ağ ile erişim sağlanabilmektedir. En büyük avantajı ağ üzerindeki kullanıcıların tamamının erişim hakkına sahip olmasıdır. (Hashem et al. 2014)'na göre NAS sunucusunun ağ üzerinden bir depolama aygıtına dolaylı olarak erişebileceği düşünüldüğünde, bir NAS sunucusundaki I/O yükü, DAS sunucusundan önemli ölçüde daha hafiftir.

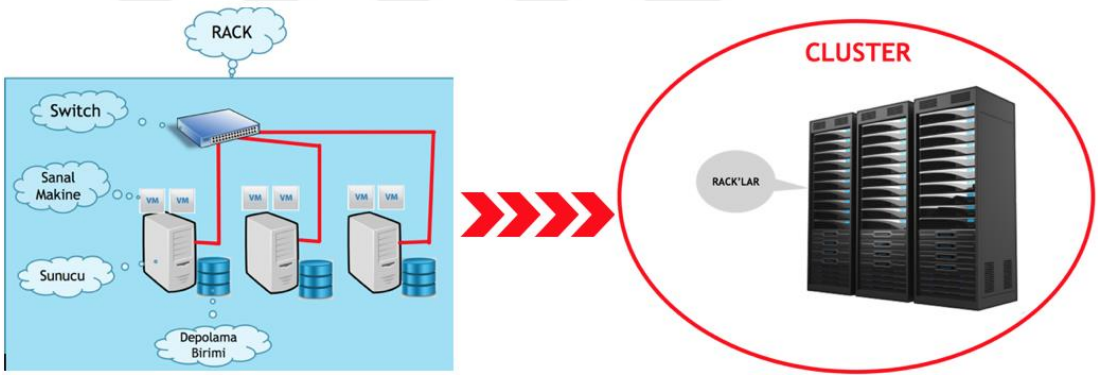
SAN ise ağa bağlı depolama biriminin yetersiz kaldığı durumlarda bir depolama birimi yerine birbirine bağlı depolama aygıtlarının kullanılmasına olanak sağlayan bir birimdir. Bağlantı için bant genişliği sınırlamalarından etkilenmeksizin uzun mesafeli veri paylaşımına olanak tanıyan bir fiber kanaldan faydalanılmaktadır.

Bu üç tür depolama biriminden ayrı, günümüzde sadece birkaç yazılımın kullanıcı sistemine yüklenmesi ile kurulum ve bakıma ihtiyaç duymadan çok sayıda veriyi ücretsiz

bir şekilde saklamaya imkân sağlayan bulut depolama alanları mevcuttur. Bu depolama alanları için iCloud, Dropbox ve Google Drive yapıları birer örnektir. Bu depolama alanlarının en önemli dezavantajları internet kesilmesi durumunda veriye erişim imkânının olmamasıdır.

1.3.3. Küme (Cluster) yapısı

Çok sayıda sanal makinenin, sunucunun, depolama biriminin, sunucular arasında bağlantıları sağlayan switch ve bağlantı kablolarının tamamının içerisinde yer aldığı yapıya şasi (*rack*) adı verilmektedir. Şekil 1.5'de görüldüğü gibi şasi olarak adlandırılan yapıların bir araya gelmesi sonucu küme (*cluster*) oluşmaktadır.

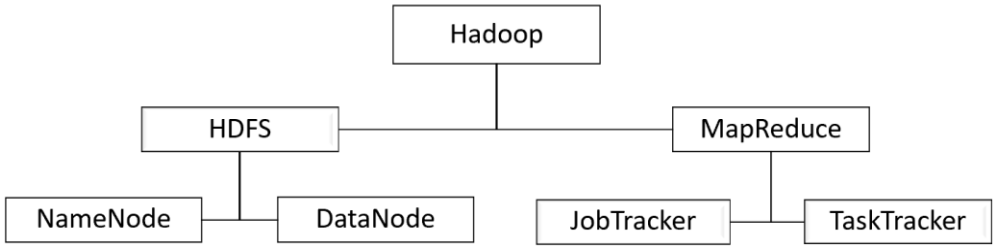


Şekil 1.5. Küme (*Cluster*) yapısı

1.4. Hadoop Nedir

Hadoop sıradan sunuculardan oluşan küme (*cluster*) üzerinde büyük verileri depolayacak ve işleyecek ilgili uygulamaları çalıştıran ve Java ile geliştirilmiş açık kaynaklı bir kütüphanedir (İlter 2013). (Anonymous 2014) kaynağına göre Apache Hadoop yazılım kütüphanesi Apache tarafından "basit bir programlama modelini kullanarak bilgisayar kümeleri arasında büyük veri kümelerinin dağıtımının yapılmasına izin veren bir çerçeve" olarak tanımlanmakta ve her biri yerel hesaplama ve depolama sunan binlerce makineye ölçeklendirmek için kullanılmaktadır.

HDFS ve MapReduce olmak üzere iki kısımdan oluşan Hadoop yapısı Şekil 1.6'da yer almaktadır.



Şekil 1.6. Hadoop yapısı

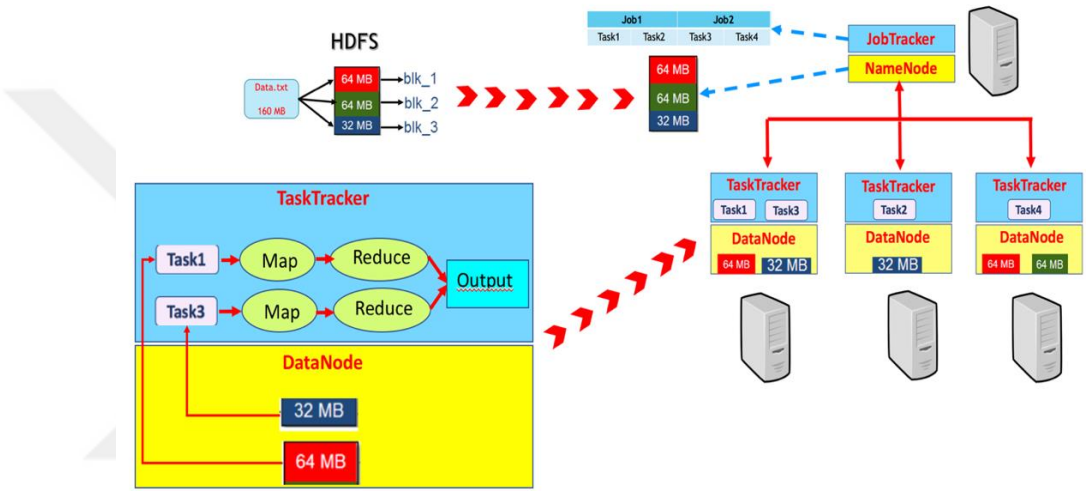
HDFS, cluster verisinin depolandığı disk alanıdır ve cluster içerisindeki sunucuların yerel disklerini kullanarak büyük verilerin saklanmasına olanak tanımaktadır. HDFS büyük verileri bloklar halinde saklamakta ve Şekil 1.7'de görüldüğü gibi blok boyutları ihtiyaca bağlı olarak 64 MB ya da 128 MB olarak seçilebilmektedir. HDFS'de verilerin makinelere dağıtılmasını mümkün kılan, tek bir NameNode ve birden fazla DataNode içeren iki çeşit yapı bulunmaktadır.

NameNode HDFS üzerindeki dosyalar hakkında bilgileri saklayan ve yöneten birimdir. Blokların sunucular üzerine dağıtılmasından, silinmesinden, bloklarda sorun meydana geldiği zaman tekrar oluşturulmasından, kısacası her türlü dosya erişiminden sorumludur.

DataNode ise NameNode tarafından gönderilen blokları saklayan yardımcı birimdir. Her DataNode kendi yerel diskindeki veriden sorumludur. Veri kaybını önlemek amacıyla her DataNode yedek verileri de barındırmaktadır.

Hadoop'un diğer bileşeni olan MapReduce ise Map ve Reduce olmak üzere iki fonksiyona ve JobTracker ve TaskTracker olmak üzere iki sürece sahiptir. Map fonksiyonu bir işin çözümüne yönelik gerekli olan verileri bloklardan çekip filtrelemek için kullanılırken, Reduce fonksiyonu ise Map fonksiyonu ile elde edilen verileri işin mantığına göre birleştirerek ilgili sonucu üretmektedir.

JobTracker yapısı görevlerin küme üzerinde dağıtılmasından ve çalıştırılmasından sorumludur. Dağıtılan görevlerin çalışması ile ilgili bir problem olursa o görevin sonlandırılması ya da yeniden başlatılmasını da sağlamaktadır. Şekil 1.7'de görüldüğü gibi TaskTracker DataNode'ların bulunduğu sunucularda çalışmakta ve JobTracker'dan görev talep etmektedir. JobTracker, NameNode'un yardımıyla DataNode'dan alınan veriye uygun Map görevini TaskTracker'lara iletmekle sorumludur.



Şekil 1.7. Hadoop çalışma şekli

1.5. Çizelgeleme (Scheduling)

Çizelgeleme veya görev çizelgeleme, büyük veriyi analiz etmek için kullanılacak görevlerin tamamının söz konusu küme düğümleri (sunucuları) üzerinde en kısa sürede işletilmesine imkân verecek görev-sunucu eşleştirmesi işleminin adıdır. Çizelgeleme, statik çizelgeleme ve dinamik çizelgeleme olmak üzere iki gruba ayrılmaktadır.

1.5.1. Dinamik çizelgeleme (Dynamic scheduling)

Dinamik çizelgeleme görev sayısının başlangıçta bilinmediği, başka bir deyişle çizelgelemenin görev geldikçe yapılması mantığına dayanmaktadır. Bu çizelgelemenin amacı hizmet önceliği sağlamak için görev önceliği ve görev performansı gibi ölçümleri kullanarak görevlerin tamamlanma süresini minumum seviyeye indirmektir.

1.5.2. Statik çizelgeleme (Static scheduling)

Statik çizelgeleme algoritmasında sistemin yapısına dair bilgilerin önceden belirlenmiş olması gerekmektedir. Bu bilgiler içerisinde sistemde yer alan makine ve görev sayısı, makineler ve görevler arasındaki yürütme maliyeti ve makinelerin birbirinden bağımsızlığı yer almaktadır.

Statik çizelgelemede işlemcilerin görevleri yürütme süresi program başlamadan önce bilinmektedir. Bu çizelgelemenin amacı ise tüm görevlerin tamamlanma süresini en aza indirmektir. İlk giren ilk çıkar çizelgeleme (*first in first out scheduling*), adil çizelgeleme (*fair scheduling*) ve kapasite çizelgeleme (*capacity scheduling*) statik çizelgelemeye örnek olarak verilebilir (Senthilkumar and Ilango 2016) ve bu çizelgelemeler Hadoop tarafından da kullanılmaktadır.

İlk giren ilk çıkar çizelgeleme mantığı kuyruk yapısını kullanarak görevleri kuyruğa yerleştirmektedir. Her bir görev parçacığını işlemek üzere TaskTracker'lar boş olan makinelere verilmektedir. Önce gönderilen görevler daha sonra gönderilen görevlere tercih edilmekte ve işin tamamlanması için makinelerin boşalması veya önceki işin tamamlanması gerekmektedir. Bu yaklaşım Hadoop tarafından varsayılan çizelgeleme olarak kullanılmakta ve kullanıcı görevlerinin yürütme sırasının hiçbir önemi olmadığı durumlarda çoğunlukla tercih edilmektedir (Senthilkumar and Ilango 2016). (Nagina and Dhingra 2016) ise bu çizelgelemenin görevin bir zaman dilimi içerisinde tamamlanması ve her göreve daha iyi cevap verme süresi sağlanmasına ihtiyaç duymakta olduğunu ifade etmiştir.

Adil çizelgeleme Facebook tarafından geliştirilmiştir ve amacı Hadoop kümesindeki kaynakları her görev için eşit şekilde paylaştırmak olarak tanımlanmıştır (Senthilkumar and Ilango 2016). (Nagina and Dhingra 2016) adil çizelgeleme sayesinde kapasite altındaki havuzlara görev vermek için kapasite üstündeki havuzlarda çalışan görevlerin sonlandırılması ile adil paylaşım yapıldığını ifade etmiştir.

Yahoo'nun geliştirdiği kapasite çizelgeleme ise kaynakları adil bir şekilde kullanmak için bir kuyruk yapısını kullanarak ilk giren ilk çıkar yaklaşımına dayalı olarak görevlere öncelik değerleri atamaktadır. Daha sonra yüksek önceliği olan görevlerin, daha az öncelikli görevlere kıyasla kaynaklara erişimini sağlamaktadır (Senthilkumar and Ilango 2016).

1.6. Problemin Tanımı

Her geçen gün ilerleyen teknoloji, kapasitesi hızla artan ve geleneksel algoritmalar/donanımlar ile işlenemeyen büyük veriyi beraberinde getirmektedir. Bu verinin depolanması, makul sürelerde işlenebilmesi ve analiz edilmesi için de dağıtık büyük veri kümelerine ihtiyaç duyulmaktadır. Bu tür veri kümelerine ise bulut bilişim hizmeti veren altyapılarda sıklıkla rastlanmaktadır. Bu altyapılarda sıkça tercih edilen Hadoop yapısı göz önüne alındığında büyük verinin bloklar halinde dağıtıldığı bulut düğümleri üzerinde veriyi işlemek amaçlı çalışacak olan çok sayıda görevin varlığı söz konusudur. Her bir görevin düğümlerde çalışma süresindeki farklılık (sunucuların konfigürasyonlarından ve depolama birimleriyle sunucular arasındaki mesafeden kaynaklanan farklılık) dikkate alındığında görevlerin düğümlere dağılımının farklı şekillerde gerçekleştirilebileceği görülmektedir. Büyük çaptaki veriyi analiz etmek için kullanılacak görevlerin tamamının söz konusu küme düğümleri (sunucuları) üzerinde en kısa sürede işletilmesine imkân verecek görev-sunucu eşleştirmesi işlemi NP-hard olarak da kategorize edilen ve global minimum arayan bir optimizasyon problemi olarak ortaya çıkmaktadır ve bu tez kapsamında bu problemin çözümüne yönelik yeni bir öneri getirilmektedir.

1.7. Amaç

$m_1, m_2, m_3, \dots, m_n$ sunucu düğümlerini ve $t_1, t_2, t_3, \dots, t_m$ de bu düğümlerde çalıştırılacak görevleri temsil ediyor olsun. Görev çizelgelemenin amacı bu görevlerin her birinin hangi düğümde işletileceğine karar vermek ve bu eşleştirmeyi son görevin tamamlanma süresini minimum yapacak şekilde gerçekleştirmektir. Çizelgeleme görev ve düğüm sayısı arttıkça yüksek maliyetli bir soruna dönüşmektedir. Burada maliyetten

kasıt çözüm uzayının büyüklüğüdür ve kaba kuvvete (*brute force*) dayanan bir çözümün makul sürelerde sonuç üretemeyecek oluşunu ifade etmektedir. Bu sebeple her bir görevin tamamlanma zamanını ($ETC[t,m]$) dikkate alarak bu problemi optimuma yakın (eğer mümkünse optimum) değerleri makul sürelerde üretebilecek şekilde çözüme kavuşturacak bir metasezgisel yaklaşımla görev çizelgeleme işleminin gerçekleştirilmesi amaçlanmaktadır.

1.8. Kapsam

Bu çalışmada büyük veri analizini gerçekleştirebilecek çoklu görev ve makine yapısına sahip bulut bilişim hizmetini kullanan altyapılar için benzetilmiş tavlama tabanlı bir metasezgisel yaklaşım geliştirilmiştir ve bu yaklaşımın seri ve paralel versiyonları C++ programlama dili kullanılarak bir bilgisayar programına dönüştürülmüştür. Paralel versiyon için aynı zamanda OpenMP kütüphanesinden faydalanılmıştır. Geliştirilen bilgisayar programının çalışma süresinin azaltılması için, benzetilmiş tavlamanın ihtiyaç duyduğu farklı rastgele sayı üretme ve pertürbasyon teknikleri, veri yapıları, döngü sonlandırma koşulları, keşif-sömürme oranları ve derleyici etkisinin analizi bu tezin kapsamında yer almaktadır.

2. KAYNAK ÖZETLERİ

Bu bölümde literatürde görev çizelgeleme problemini ele alan sezgisel ve metasezgisel yaklaşımlardan, doğrusal (*lineer*) programlamaya dayalı çalışmalardan ve çoklu bulut ortamında yapılan farklı çizelgeleme yöntemlerinden bahsedilecektir.

2.1. HCSP

Son yıllarda işlem gücü yüksek bigisayarların ve kapasitesi yüksek ağların hızlı artışı karmaşık problemlerin çözümü için dağıtık hesaplama ortamlarının yayılmasını beraberinde getirmiştir. Bu hesaplama ortamları önceleri aynı işlem birimlerinden oluşan ve birbiri ile bağlantılı görevleri işleten homojen bir yapıya sahipken süreç içerisinde farklı özelliklere ve işlem gücüne sahip hesaplama birimlerinden oluşan heterojen yapılara yerini bırakmıştır. HCSP heterojen hesaplama sistemlerinde en son tamamlanan görevin süresini minimuma indirmek için yapılacak çizelgeleme problemini ifade etmektedir. HCSP'nin homojen ortamlarda çözülmesi düşük sayıda işlemci içeren yapıların kullanılmasından kaynaklanan performans kayıpları nedeniyle zordur. Bunun yerine yoğun hesaplama gerektiren işlemler için dağıtık yüksek performanslı makineleri barındıran ve kaynak kullanımı gibi birçok hedefi karşılamak için makinelere uygun görev ataması sağlayan heterojen hesaplama sistemleri kullanılmaktadır. (Anonymous 2012; Gogos *et al.* 2016).

Heterojen hesaplama sistemlerine zaman içerisinde gelişen bulut bilişim hizmeti sağlayan altyapılarda da sıkça rastlanmaktadır. Görev çizelgeleme bu altyapılarda bulut düğümleri (sunucuları) üzerinde yapılmaktadır. Her bir düğümün her bir görevi çalıştırma süresi birbirinden farklı olduğu için bulutta görev çizelgeleme problemi HCSP ile heterojen ortamlarda çizelgeleme yapılması ve minimum süre arayışı açısından benzerlik göstermektedir. Literatürde HCPS ve heterojen bulut sisteminde çizelgeleme problemi konusunda çok sayıda çalışma bulunmaktadır. Ancak (Li *et al.* 2012) birleşik çoklu bulut sisteminde çizelgeleme konusunda ilk çalışmayı gerçekleştirdiklerini ifade etmişlerdir.

Bulutta görev çizelgeleme sorunu modern bulut bilgi işlem tabanı (*IT*) altyapısında oldukça önemli bir araştırma alanıdır. Geleneksel bilgi işlem ortamında, bir görevin yürütme süresi CPU ve gerekli tüm kaynaklar kendisine ayrıldıktan sonra sabitlenir. Ancak, bulut bilişim ortamında, bir görevin tamamlanma süresi, hesaplamanın yapıldığı düğüme bağlıdır. Tamamlanma süresindeki bu değişiklik, verilerin bilgisayar düğümüne olan konumundan kaynaklanmaktadır. Bulut altyapısında kullanılan, sanallaştırma teknolojisi ve hipervizör yazılımları ile desteklenen ve çok sayıda sanal makineyi barındıran şasi yapıları farklı coğrafik konumlarda bulunmaktadır. Hesaplama ve veri düğümleri arasındaki bu mesafe, herhangi bir görevi yürütmek için gereken gerçek yürütme süresinde önemli bir rol oynamaktadır (Syed *et al.* 2015).

2.2. Sezgisel ve Metasezgisel Yaklaşımlar

2.2.1. Min-Min

Görev çizelgeleme için geliştirilmiş olan bu sezgisel yöntem herhangi bir i görevinin makinelerdeki tamamlanma süresini kullanarak söz konusu bu i görevi için, en erken tamamlanma süresini veren makine m_i'yi bulmakta ve tamamlanma matrisinin (çizelgenin) i. satırına eklemektedir. Daha sonra, en erken tamamlanma süresine sahip olan görev i_k seçilmekte ve önceden hesaplanmış makineye m_k atanmaktadır. En sonunda i_k görevi, görevler arasından çıkarılmaktadır. Süreç atanacak hiç bir görev kalmayıncaya kadar tekrarlanmaktadır (Xhafa *et al.* 2007b).

2.2.2. Sufferage

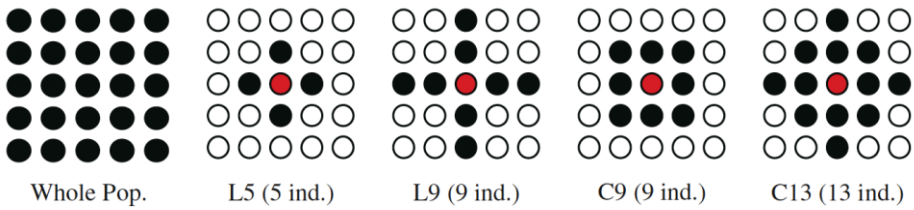
Görev çizelgeleme için kullanılan Sufferage sezgisel yönteminin arkasındaki temel fikir bir görevin iki makineden hangisinde çalıştırılmasına karar vermek için *sufferage* parametresinin kullanılmasıdır. Bu parametre bir görevin herhangi iki makinede tamamlanma süreleri arasındaki farkı ifade etmektedir. Her iterasyonda çözümden alınan her bir görev için *sufferage* değeri hesaplanmakta ve en yüksek *sufferage* değerine sahip olan görev en hızlı makineye atanmaktadır (Xhafa *et al.* 2007b).

2.2.3. MCT

MCT sezgisel yöntemi rastgele sıralanmış görev kümesini dikkate almaktadır. Kümeden alınan her görev bu görevin minimum yürütme süresine sahip makineye atanmaktadır (Braun *et al.* 2001).

2.2.4. cMA

cMA popülasyon tabanlı sezgisel bir yöntemdir. Yüksek kaliteli çözümlerin çok kısa sürede bulunması konusunda önemli bir yere sahiptir. cMA'da bireyler popülasyona dağınık olarak yerleştirilmekte ve evrimsel işlemler komşulara uygulanarak yeni bir popülasyon oluşturulmaktadır. Algoritmada popülasyon iki boyutla temsil edilen bir grid yapısındadır. Bu yapıya ait ilk birey LJFR–SJFR yöntemi kullanılarak oluşturulmaktadır. Yöntem ilk olarak en hızlı makinelere en uzun görevleri atamakta, daha sonra kalan görevleri işini en hızlı bitiren makineye ya en uzun görevi (LJFR) ya da en kısa görevi (SJFR) atayarak tamamlamaktadır. Geri kalan bireyler ilk bireyin pertürbasyonlarından rastgele elde edilmektedir. Algoritmada Şekil 2.1'de yer alan farklı türdeki komşuluk yapıları arasından görev çizelgeleme problemi için C9 (9 birey) tercih edilmektedir (Xhafa *et al.* 2008a).



Şekil 2.1. cMA komşuluk yapısı (Xhafa *et al.* 2008a)

Algoritmada N-turnuva (*Ntour*) seçimi kullanılarak her iterasyonda üç birey seçilmekte ve seçilen bireylere tek noktada (*one-point*) çaprazlama işlemi uygulanmaktadır. Daha sonra mutasyon işlemi ile aşırı yüklenmiş makinelerdeki yükler yük miktarı az olan makinelere transfer edilerek yeniden dengeleme (*rebalancing*) yapılmaktadır.

Algoritmanın son aşamasında LMCTS yöntemi ile farklı makinelere atanan ve tamamlanma süresinde en iyi azalmayı sağlayan görev çiftlerinin değiştirilmesiyle kaliteli çözümler oluşturulmaktadır (Xhafa *et al.* 2008a).

2.2.5. TS

Tabu arama diğer sezgisel yöntemlere rehberlik etmek için Glover tarafından geliştirilen yüksek seviyeli bir algoritmadır. Amacı yerel optimuma takılmayı önlemek için arama alanını akıllıca keşfetmektir. Algoritmada çözümü temsil etmek için çizelgede yer alacak olan görev sayısı boyutunda bir vektör kullanılmakta ve çözüm Min-Min sezgisel yöntemi ile doldurulmaktadır. Başlangıç çözümünden komşu çözümler üretmek için eşit olasılıkla uygulanan transfer ve takas (*swap*) hareketleri kullanılmaktadır. Önceden ziyaret edilen çözümleri cezalandırmak (*tabu*) için uzun, her bir görevin makinelere atanma sıklığını tutmak için kısa bellek yapısı kullanılmaktadır. Daha iyi bir çözüme ulaşılması durumunda uzun bellekte tutulan tabu durumları kaldırılmaktadır. Mevcut çözümden daha iyi çözüm oluşturmak için en iyi çözümlerin ödüllendirilmesi, yük faktörüne dayalı yaklaşım ve eşit sayıda yapılan transfer ve takas işleminin yerine önce bir takas dizisi ardından tek bir transfer işlemi olmak üzere üç farklı yoğunlaştırma (*insentification*) yönteminin kombinasyonu kullanılmaktadır. Yoğunlaştırma işleminin ardından çözümlerde çeşitlendirme (*diversification*) işlemi ile hafif bir tedirginlik oluşturularak en iyi çözüm belirlenmektedir (Xhafa *et al.* 2008b).

2.2.6. MA

MA da popülasyon tabanlı bir algoritmadır. Genetik ve evrimsel algoritmaya benzemekte ancak onlardan farklı olarak ara popülasyon kullanmamaktadır. Çeşitlilik oluşturmak için yeniden birleştirme ve mutasyon işlemi popülasyonun tamamına uygulanmakta, bu nedenle popülasyonun boyutu her bir iterasyonda artmaktadır. Seçim aşamasında mevcut popülasyon başlangıçtaki popülasyona indirgenerek sınırsız büyüme önlenmektedir. MA'da çözümü temsil etmek için görev sayısı boyutunda vektör yapısı kullanılmakta ve içerisi M makine sayısı olmak üzere [1, M] aralığında yer alan doğal sayılar ile MCT ve

LJFR-SJFR sezgisel yöntemleri kullanılarak doldurulmaktadır. Daha sonra popülasyondan ikişer çözüm alınarak tek noktada çaprazlama yapılmaktadır. Mutasyon aşamasında makinelerin yüklerini dengelemek için yeniden dengeleme (*rebalancing*) yöntemi kullanılmaktadır. Yeniden dengeleme ile mutasyon işlemi yapılırken yük miktarı fazla olan makine ve o makinede çalışan iki görev seçilmekte daha sonra çalışma süresi yüksek olan görev o makineden alınarak başka bir makineye atanmaktadır. Seçim için ise MA'da iki yöntem kullanılmaktadır. İlk olarak LMCTS yöntemi ile rastgele bir görev seçilmekte, daha sonra bu görev çalışma zamanını düşürecek bir makinedeki görev ile takas yapılmaktadır. İkinci seçim ise LTH yöntemi ile yapılmaktadır. Bu yöntem Tabu algoritmasına dayalı bir seçim yöntemidir ve lokal optimumlara takılmayı önlemektedir. Yöntem ya en iyi çözümleri sömürmekte ya da kısa süreli en kötü çözümleri kabul etmektedir. MA'nın LTH ile çalıştırılması MA+TS algoritması olarak adlandırılmaktadır (Xhafa 2007a).

2.2.7. GA

Genetik algoritma geniş çözüm alanında aramalar yapmak için kullanılan bir yaklaşımdır. GA'da çözümü temsil etmek için bir popülasyon oluşturulmaktadır. Popülasyon içerisindeki her bir kromozom görev odaklı kodlama kullanılarak vektör yapısı ile temsil edilmektedir. Burada da popülasyonu doldurmak için Min-Min sezgisel yaklaşımı kullanılmaktadır. Popülasyon, her kromozoma atanan görevlere bağlı olarak bir uygunluk değeri (*fitness value*) ile değerlendirilmektedir. Daha sonra çaprazlama işlemi için rastgele iki kromozom şeçilmekte ve yine rastgele seçilen bir noktada çaprazlanmaktadır. Mutasyon üç farklı aşamadan oluşmaktadır. İlkinde çalışma süresi yüksek olan bir makineden görev alınarak başka bir makineye atanmaktadır. İkincisinde çalışma süresi en az olan görev en düşük makineye atanmaktai üçüncüsünde ise çalışma süresi en fazla olan görev en yüksek makineye atanmaktadır. Belli bir yürütme süresine veya iterasyon sayısına ulaşılması durumunda algoritma sonlandırılmaktadır (Nesmachnow*et al*. 2010).

2.2.8. PGA

Arama verimliliğini arttırmak için GA'da paralelleştirme yapılmaktadır. GA'nın paralelleştirilme yaklaşımlarından biri popülasyonun q parçaya ayrılmasıdır. Burada q fiziksel işlem elemanını (*physical processing elements-PPEs*) göstermektedir. Parçalanmış her alt popülasyon çok sayıda kromozom içermektedir. Her bir PPE orijinal GA algoritmasını kullanarak kendine ayrılmış olan popülasyon parçasını çalıştırmaktadır. Böyle bir paraleleştirme yapısına iri taneli (*coarse-grained*) PGA denilmektedir. İri taneli PGA, her bir PPE arasında bilgi alışverişi yapılmasını farklı şekilde sağlamaktadır. Bunun için bağlı ada modelini (*connected island model*) kullanmaktadır. Bu modele göre tüm PPE'ler o ana kadar buldukları çözümler ile ilgili bilgi alışverişinde bulunmak için eşzamanlı iletişim ortamı oluşturmaktadırlar. Böylelikle tüm PPE'lerden gelen çözümler karşılaştırılarak en iyi çözüm tekrar PPE'lere yönlendirilmektedir. GA'da olduğu gibi belli bir yürütme süresine veya iterasyon sayısına ulaşılması durumunda algoritma sonlandırılmaktadır (Nesmachnow*et al.* 2010).

2.2.9. SGA

Görev çizelgeleme için geliştirilmiş genetik algoritmanın karma tabanlı bir türüdür. SGA, GA'ya benzer şekilde çalışmaktadır. SGA'da yeni bir nesil oluşturmak için, mevcut popülasyonun sadece bir kısmı yeni bireyler ile değiştirilmektedir. Popülasyondaki en kötü bireyi değiştirmeyi amaçlayan SGA yeni bir bireyi ancak değiştirilecek olandan daha iyi bir uygunluk değerine sahip olması durumunda kabul etmektedir. Algoritmada iki bireyin ne kadar benzer olduğunu göstermek için bir benzerlik ölçütü kullanılmaktadır. SGA'da yeni bir bireye benzer bir bireyi bulmak, mevcut neslin tüm bireylerini karşılaştırmayı gerektirdiğinden yüksek bir hesaplama maliyetine sahiptir. Amaç daha iyi bir bireye ulaşmayı sağlarken optimizasyon hızını korumaktır. SGA'nın uygulanması basittir. Bununla birlikte, SGA'nın performansı, popülasyonun iyi çeşitliliğine bağlı olmasının yanı sıra, genetik operatörlerin geri kalanına da bağlıdır. Bu nedenle, çaprazlama ve düşük mutasyon oranı, SGA'nın çok iyi bir performans göstermesini sağlamaktadır (Xhafa *et al.* 2008c).

2.2.10. NSGA-II

NSGA-II görev çizelgeleme problemlerini çözmek amacıyla oluşturulmuş genetik algoritmanın çok amaçlı kullanılan bir türüdür. Başlangıçta N bireye sahip bir popülasyon üretilmektedir. Bu N birey seçim, çaprazlama ve mutasyon işlemleri kullanılarak oluşturulmaktadır. NSGA-II nüfus içindeki bireylerin çeşitliliğini sağlamak için baskın olmayan sıralama ve kalabalık mesafesini kullanmaktadır. Birleşik sıralama olarak adlandırılan bu sıralama bireylerin tamamını 2D yapısında bir popülasyonda birleştirmektedir. Kalabalık mesafesi, bireyin komşularına ne kadar yakın olduğunu göstermektedir. Bu mesafe değeri yüksek olan bireylere ait oldukları yere dayalı yüksek bir uygunluk değeri, düşük olanlara ise daha düşük bir uygunluk değeri tahsis edilmektedir. Seçim sırasında ise N boyutlu popülasyondan uygunluk değeri düşük olanlar tercih edilmektedir. Böylelikle NSGA-II, etkili karşılaştırmalar yaparak popülasyondaki en iyi çözümü belirlemektedir (Subashini *et al.* 2010).

2.2.11. CHC

CHC kısaltması çapraz nesil elit seçimi, heterojen rekombinasyon ve kataklizik mutasyon (*cross generational elitist selection, heterogeneous recombination, and cataclysmic mutation*) anlamına gelmektedir. Geleneksel GA'nın özelleştirilmiş bir halidir. CHC, başlangıç çözümü için 120 kromozoma sahip popülasyon yapısını kullanmaktadır. Her bir kromozom Şekil 2.2'de görüldüğü gibi görev odaklı kodlamaya (*task oriented encoding*) dayalı olarak tek boyutlu bir vektörden oluşmaktadır. Kromozomların içerisi Min-Min sezgisel yaklaşımına kullanılarak doldurulmaktadır (Nesmachnow *et al.* 2010).

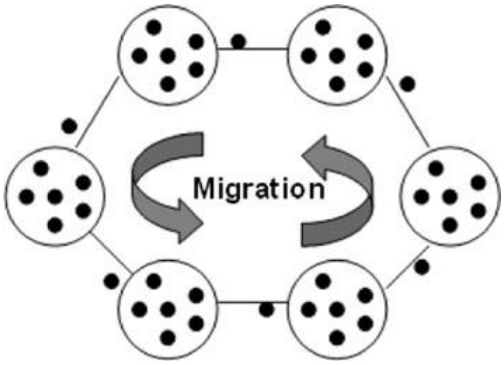
m_1	m_2	m_3	m_0	m_3	...	...	...	m_j	...	m_k	...	...
t_1	t_2	t_3	t_4	t_5	...	...	...	t_i	...	t_k	...	...

Şekil 2.2. Görev odaklı kodlama (Nesmachnow*et al.* 2010)

CHC algoritması popülasyonun en iyi bireylerini sürdürmek için sadece birbirinden çok sayıda bit ile ayrılan ebeveynlerin çoğalmasına izin vermektedir. Ayrıca algoritma popülasyonda çeşitliliği sağlamak için yeniden birleştirme ve yeniden başlatma yöntemlerini kullanmaktadır. Yeniden birleştirme işlemi iki kromozom arasında farklılık gösteren bitlerin tam olarak yarısının rastgele değiştirilmesini içeren HUX yaklaşımı ile yapılmaktadır. Yeniden başlatma işleminde ise popülasyonda yer alan en iyi kromozoma hareket (*move*) veya takas (*swap*) yöntemleri uygulanmaktadır. Tüm bu işlemlerin ardından son olarak iyileşme kontrol edilmekte ve iterasyon sayısına bağlı olarak algoritma sonlandırılmaktadır (Nesmachnow *et al.* 2010).

2.2.12. p-CHC

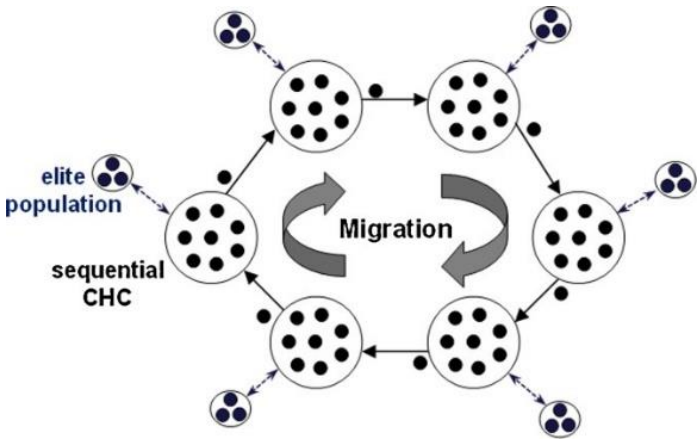
CHC algoritmasının verimliliğini arttırmak, yüksek kaliteli çözümler elde etmek ve zor optimizasyon problemlerine çözüm bulmak amacıyla paralelleştirme yapılmaktadır. CHC'yi paralelleştirmek için mevcut popülasyon alt popülasyonlara ayrılmaktadır. Her bir alt popülasyon ardışık olarak CHC algoritmasını çalıştırmaktadır. Her birey sadece içerisinde bulunduğu alt popülasyondaki bireylerle etkileşim içinde olmaktadır. Bireyler arasında paralel olarak işlem yapılması göç (*migration*) operatörü ile sağlanmaktadır. Göç operatörü algoritmada çeşitlilik oluşturmak için ara sıra alt popülasyonlar arasından seçtiği bireyleri birbirleri ile değiştirmektedir. Göç operatörü, bireylerin ne zaman değiştirileceğini belirleyen gönderici göçmenler (*sendmigrants*) ve yabancı bir alt popülasyondan gelen bireylerin kabul edilip edilmeyeceğini belirleyen alıcı göçmenler (*recvmigrants*) olmak üzere iki kısma ayrılmaktadır. Göçmenler (*migrants*) bir alt popülasyondan diğerine göç edecek olan bireyleri göstermektedir. Göç gönderme (*sendmigration*) ve göç alma (*recvmigration*) operatörleri her alt popülasyon arasında iletişimi sağlamak için çoğunlukla Şekil 2.3'de görüldüğü üzere tek yönlü halka (*unidirectional ring*) çizgesi ile bağlantıyı sağlayarak popülasyonlar arası bireylerin değiştirilmesine olanak tanımaktadır (Nesmachnow *et al.* 2010).



Şekil 2.3. Tek yönlü halka çizgesi (Nesmachnow *et al.* 2010)

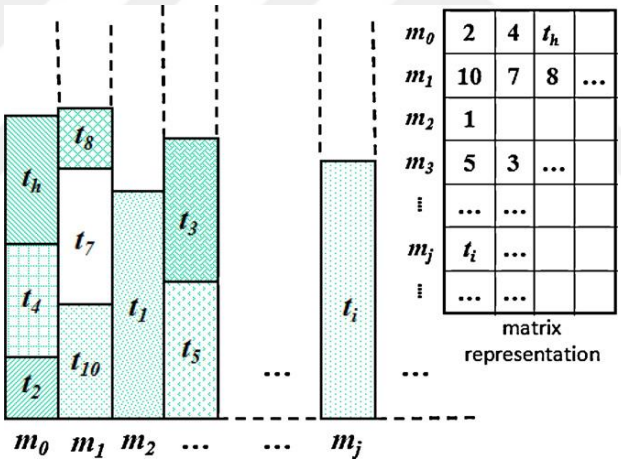
2.2.13. pμ-CHC

p-CHC'de mevcut popülasyonu alt popülasyonlara ayırarak algoritma paralel hale getirilmektedir. Ancak alt popülasyonların kullanılması çözümün hızlıca çeşitliliğini kaybetmesine ve aramada durgunluğa sebep olmaktadır. Çeşitlilik kaybının üstesinden gelmek için μ-GA algoritması kullanılarak CHC algoritması tasarlanmıştır. μ-GA algoritmasında arama boyunca bellekte saklamak için iki popülasyon kullanılmaktadır. İlki Şekil 2.4'de yer alan CHC'nin ana popülasyonu, ikincisi aramada bulunan en iyi çözümleri saklamak için kullanılan seçkin popülasyon (*elite population*)'dur. Seçkin popülasyon belli sayıda iterasyondan sonra popülasyonun yeniden başlatılmasını o ana kadar bulunan en iyi bireyleri kullanarak düşük bir hesaplama maliyeti ile sağlamaktadır (Nesmachnow *et al.* 2012).



Şekil 2.4. p-CHC tek yönlü halka çizgesi (Nesmachnow*et al.* 2012)

pμ-CHC algoritması orijinal CHC'nin paralel modelinde kullanılan alt popülasyonlara ayırma işlemi ile μ-GA'nın iki temel kavramını kullanarak oluşturulmuştur. Bu temel kavramlardan ilki arama sırasında seçkin çözümleri depolamak için ayrı bir popülasyon oluşturmak, ikincisi her alt popülasyondaki çeşitliliği sağlamak için en iyi bilinen yerel arama yönteminin rastgele versiyonunu kullanarak yeniden başlatmayı hızlandırmaktır. pμ-CHC'nin her bir alt popülasyonunda sekiz birey, her bir seçkin popülasyonunda üç birey yer almaktadır. Seçkin popülasyona birey eklemek için her zaman popülasyondaki en kötü olan birey çıkarılmaktadır. Ayrıca başlangıç popülasyonu Şekil 2.5'da yer alan makine odaklı kodlama (*machine-oriented encoding*) kullanılarak oluşturulmaktadır. Makine odaklı kodlama her bir makinede çalıştırılacak olan görev gruplarını temsil etmek için 2D yapısını kullanmaktadır (Nesmachnow *et al.* 2012).



Şekil 2.5. Makine odaklı kodlama (Nesmachnow *et al.* 2012)

pμ-CHC algoritması Min-Min ve Sufferage sezgisel yaklaşımlarının olasılık versiyonunu kullanarak kendi sezgisel yaklaşımını oluşturmaktadır. Bunun için ilk olarak rastgele sayıda görevi popülasyonda görevlendirmekte daha sonra kalan görevleri MCT sezgisel yaklaşımı ile popülasyona yerleştirmektedir. İki çözümü birleştirmek amacıyla HUX yaklaşımını kullanan algoritma yerel bir optimum değere takılmayı önlemek için en iyi birey kullanarak yeniden başlatma işlemi gerçekleştirmektedir. pμ-CHC'da yeniden başlatma işlemi orijinal CHC ile aynıdır. Ancak pμ-CHC yeniden başlatmayı PALS sezgisel arama yaklaşımı ile birleştirerek kullanmaktadır. PALS tek bir çözüm kullanarak

maliyet fonksiyonunun değerini yerel olarak iyileştirmeyi amaçlayan hareketler uygulayan yinelemeli bir geliştirme yöntemidir. En önemli adımı belirli bir hareket uygulandığı zaman maliyet fonksiyonunun da hesaplanmasıdır. PALS sezgisel yaklaşımı pμ-CHC'de belirli bir iterasyonun sonrasında yeniden başlatmayı takiben algoritmaya uygulanmakta ve bu sayede eşleşme prosedürü ile yeni bir birey eklemeye gerek kalmamaktadır (Nesmachnow *et al*. 2012).

2.3. Doğrusal (Lineer) Yaklaşım

2.3.1. CPD ve CPR

CPD ve CPR, birbirinden bağımsız görevlerin yer aldığı heterojen ortamlarda matematiksel model ve sezgisel yöntemler kullanılarak çizelge yapmak için geliştirilmiş birer yaklaşımdır. Bu yaklaşımlarda çözümü oluşturmak için TPB yöntemi kullanılmaktadır. TPB ilk olarak her bir görevi en hızlı çalıştıran makineye atamaktadır. Sonra yük miktarı en fazla olan makineden en az olan makineye görev aktarımı yaparak her iterasyonda makineleri mevcut yüklerine göre sıralamaktadır. Daha sonra matematiksel model kullanılarak her bir makinenin çalıştırdığı görevlerin sayısını hesaplamaktadır. Tüm makinelere ait görev sayıları hesaplandıktan sonra doğrusal programlama çözücüsü kullanılarak problem çözülmekte ve yeni çözüm üretilmektedir. Yeni çözümü oluşturmak için dört farklı sezgisel yöntem kullanılmaktadır. İlk sezgisel yöntemde en fazla yüklü makineden bir görev başka bir makineye atanmaktadır. İkinci sezgisel yöntemde en fazla yüklü makineden bir görev daha düşük bir görev ile takas edilmektedir. Üçüncü sezgisel yöntemde en fazla yüklü makineden başka bir makinede daha hızlı çalışacak bir görev seçilmekte ve taşınmaktadır. Dördüncü sezgisel yaklaşımda ise yine en fazla yüklü makineden bir görev yükü daha az olan makinelerden birisine aktarılmaktadır. Bu dört farklı sezgisel yaklaşım sayesinde bir önceki çözümden daha iyi bir çözüm elde edilmesi sağlanmaktadır. Program iki farklı şekilde sonlandırılmaktadır. İlk sonlandırma Java ile hazırlanmış ve Gurobi 6.0.3 Integer Programming (*IP*) çözücüsü olarak bilinen bir matematik çözücüsüne (*Math Solver*) ait alt sınıra (*MS Bound*) ulaştığında yapılmaktadır. Programın çalıştırılıp bu şekilde sonlandırılması ise CPD

yaklaşımı olarak adlandırılmaktadır. İkinci sonlandırma ise programın yeniden başlatılması ile gerçekleştirilmekte ve CPR yaklaşımı olarak isimlendirilmektedir (Gogos *et al.* 2016).

2.4. Heterojen Çoklu Bulut Ortamında Çizelgeleme Yaklaşımları

2.4.1. CLS

Bulutta görev dağılımı yapmak için geliştirilmiş statik bir çizelgeleme algoritmasıdır. Bulut heterojen bir yapıya sahip olduğu için sanal makineler (*VM*) üzerinde çalışan görevlerin yürütme süreleri aynı değildir. Bu sebeple algoritmada başlangıçta görevlerin en erken ve en geç tamamlanma süreleri dikkate alınarak görevlere öncelik değerleri atanmakta daha sonra bu önceliklere bağlı görev listesi oluşturulmaktadır. Görev listesi oluşturulduktan sonra görevler listede yer alan sıraya göre yerleştirilmektedir. Listenin en üst kısmına en erken tamamlanma süresine sahip görev yerleştirilmekte, liste tamamen dolup bulut kaynakları hazır olduğunda görevler listeden kaynaklara dağıtılmaktadır. Kaynaklara gönderilen her bir görev listeden çıkarılmakta ve liste tamamen boşalıncaya kadar işlem devam etmektedir (Li *et al.* 2012).

2.4.2. CMMS

CMMS algoritması CLS algoritması gibi bulutta görev dağılımı için geliştirilmiş statik bir çizelgeleme algoritmasıdır. Bu algoritmanın çalışması açgözlü algoritma (greedy algorithm) ve Min-Min sezgisel yaklaşımına benzemektedir. CMMS de Min-Min yaklaşımı gibi görevleri en erken çalıştırma süresine sahip olan bulut kaynaklarına atamakta ve kaynaklar boşaldıkça kalan görevler için aynı işlemi devam ettirerek algoritmayı çalıştırmaktadır (Li *et al.* 2012).

2.4.3. MCC

MCC algoritması çoklu bulut ortamında görev çizelgeleme yapmak için geliştirilmiştir. Algoritma kuyruk veri yapısını kullanmakta ve ilk olarak görevleri ilk gelen ilk hizmet

alır yaklaşımı ile kuyruğa yerleştirmektedir. Daha sonra kuyrukta yer alan görevleri bulut düğümlerine dağıtmaktadır. Her bir bulut düğümüne atanan görevlerin tamamlanma süresini hesaplayarak minimum tamamlanma süresini veren bulut-görev çiftini belirlemekte ve görevi kuyruktan çıkarmaktadır. Kuyruk boşalınca algoritma sonlandırılmaktadır. MCC algoritmasının çalışması MCT sezgisel yöntemine benzemekte olup görevlerin hazırlanma süresini kullanma açısından MCT ile farklılık göstermektedir (Panda *et al.* 2015).

2.4.4. MEMAX

MEMAX iki aşamalı bir zamanlama algoritmasıdır. Algoritmanın çalışması iki aşamalıdır. İlk aşamada, bulut düğümlerinde tüm hazır görevlerin yürütülme sürelerinin ortalaması hesaplanmaktadır. İkinci aşamada ise maksimum ortalama değere sahip olan görev seçileerek en düşük yürütme süresine sahip olan düğüme atama yapılmaktadır (Panda *et al.* 2015).

2.4.5. CTPS

Bulutta görev bölümleyen çizelge olarak tanımlanan CTPS algoritması, heterojen çoklu bulut ortamı için geliştirilmiş çevrimiçi bir algoritmadır. Algoritmanın çalışma şekli ön işleme ve işleme olmak üzere iki aşamadan oluşmaktadır. İlk aşamada, bir görevin tüm bulut düğümlerindeki ön işleme ve maksimum hazırlanma süresi, ikinci aşamada ise aynı görevin bulut düğümlerindeki tamamlanma süresi hesaplanmaktadır. Ardından, görev minimum tamamlanma süresini veren düğüme atanmaktadır. İşlem her bir görev için tekrarlanmaktadır. Bu algoritmanın diğerlerinden farkı aynı anda bir görevin iki bulutta çalışmasına olanak tanımasıdır. Bunu hem ön işleme hem de işleme aşamasında iki farklı bulut düğümüne aynı görevi atayarak sağlamaktadır. Algoritmanın gerçek tamamlanma süresi *k* tane iterasyon, *n* tane görev, *m* tane bulut düğümü ve sabit bir *o* değeri için O (knmo)'dir. (Panda *et al.* 2017).

2.4.6. SBTS

SBTS algoritması heterojen bulut ortamında görev çizelgeleme yapmak için geliştirilmiş bir başka algoritmadır. Algoritma düzgünleştirme ve çizelgeleme olmak üzere iki aşamadan oluşmaktadır. Düzgünleştirme aşamasında tüm görevler yürütme sürelerine bağlı olarak çeşitli gruplara ayrılmaktadır. Gruplandırma işlemi her bir görevin bulut düğümlerinde yürütülme süresine ($ETC[i,j]$) ve en küçük tamsayıya (*Count*) bağlı kıyaslama ile gerçekleştirilmektedir. Kıyaslama için $ETC[i,j] < 10^{Count}$ koşulu kullanılmakta ve şartı sağlayan tüm görevler aynı grupta yer almaktadır. Belirlenen her bir değer için kıyaslama yapılarak gruplar oluşturulmaktadır. Çizelgeleme aşamasında ise gruplar azalan sırada birbiri ardına bulut düğümlerine gönderilmektedir. Çizelgelemedeki amaç bulut kaynaklarında görevlerin çalıştırılması için harcanan maksimum süreyi en aza indirmektir (Panda *et al.* 2014).

2.4.7. TSTT

Dağıtık bilgi işlem sistemlerinde hızlı görev yürütme süresine duyulan ihtiyaç hızla artmıştır. Bu ihtiyaç, yürütme süresini azaltmak ve kaynaklar arasındaki kullanımı arttırmak için etkili bir algoritma olan TSTT'yi beraberinde getirmiştir. TSTT iki aşamalı bir algoritmadır. İlk aşamada TPB taklaşımını kullanarak makineleri yüklerine bağlı olarak artan sırada sıralamaktadır. Daha sonra görevleri en erken tamamlanma süresine sahip makinelere atamaktadır. İkinci aşamada atama işlemi tamamlandıktan sonra makineler kontrol edilerek en fazla yüklenen makinenin başka bir makine ile görev değişikliği gerçekleştirilmektedir (Al-Qadhi *et al.* 2018).

3. MATERYAL ve YÖNTEM

Tezin bu bölümünde tez kapsamında geliştirilen benzetilmiş tavlama tabanlı metasezgisel yöntem ve bu yöntemin aşamaları (başlangıç sıcaklığının, soğutma oranının, donma noktasının, iç döngü iterasyon sayısının, başlangıç çözümünün ve veri yapısının, rastgele sayı üretme yönteminin, kabul kriterinin, pertürbasyon yönteminin, maliyet fonksiyonunun, iç döngü erken sonlandırma kriterinin, dış döngü erken sonlandırma kriterinin ve derleyici etkisinin araştırılması), veri seti modeli ve testlerin gerçekleştirildiği donanımsal ve yazılımsal bileşenler incelenecektir.

3.1. Yöntemin Tanıtımı

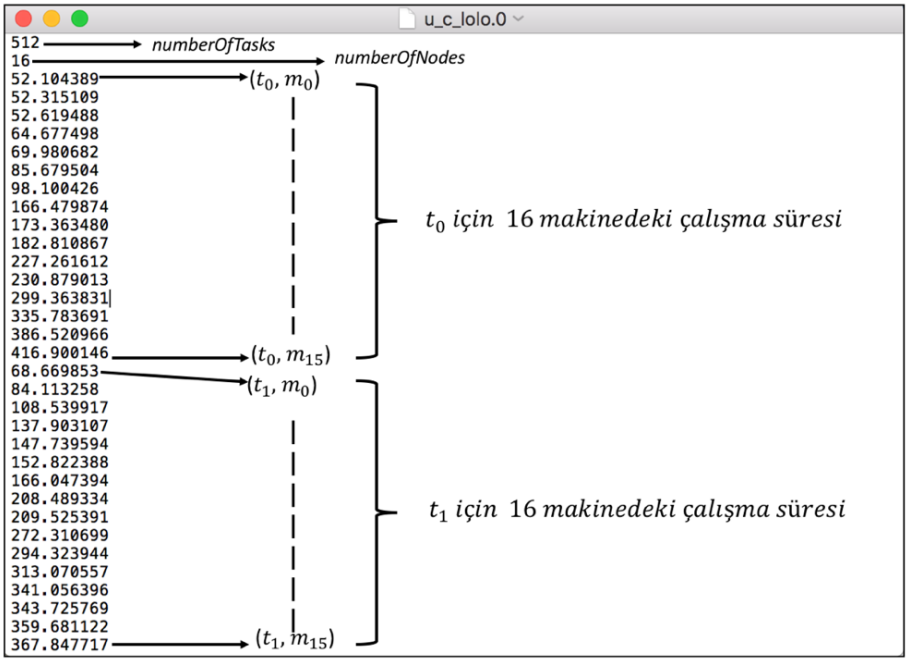
Bu çalışmada görev çizelgeleme için benzetilmiş tavlama tabanlı bir metasezgisel yaklaşımın seri ve paralel versiyonları C++ programlama dili kullanılarak bir bilgisayar programına dönüştürülmüştür. Paralel versiyon için aynı zamanda OpenMP kütüphanesinden faydalanılmıştır. Ayrıca geliştirilen yaklaşımın yetkinliği, görev çizelgeleme algoritmalarının performanslarını karşılaştırmak için kullanılan ve Braun modeliyle oluşturulmuş on iki meşhur benchmark ile test edilmiştir.

Benzetilmiş tavlama isimli metasezgisel, esin kaynağını metalurji biliminden almaktadır. Yüksek sıcaklıklara kadar ısıtılan metaller rastgele sıvı hale dönüşmekte ve uygun bir şekilde yavaşça soğutulduklarında düzenli bir kristal yapıya kavuşmaktadır. Bu gözlemden esinlenen benzetilmiş tavlama yöntemi de eldeki problemi temsil eden rastgele bir ilk çözümü yüksek sıcaklıklardan başlayarak ve bu sıcaklığı her adımda yavaşça azaltarak arzulanan global çözüme yaklaştırmayı hedeflemektedir.

3.2. Veri Seti

Veri seti, birbiriyle ilişkili verilerin belirli bir düzen dahilinde çok amaçlı kullanıma uygun şekilde bir arada tutulduğu bilgi topluluğudur. Veri seti oluşturulmasının amacı mevcut bir problemi çözmek için kullanılacak olan analiz etme, hata ayıklama gibi çeşitli karmaşık işlemlere düzenli bir girdi sağlamaktır.

Bu çalışmada veri seti olarak (Anonymous 2012)'den alınan Braun modeliyle oluşturulmuş on iki meşhur benchmark kullanılmaktadır. Şekil 3.1'de görüldüğü gibi her bir veri seti içerisinde 512 adet görevin 16 farklı makinede çalışma süresi yer almaktadır.



Şekil 3.1. Veri seti modeli

Bu süreler her bir benchmark için tek bir sütundan ve 8192 (512×16) satırdan oluşan ETC (*expected time to compute*) matrisinde saklanmaktadır. ETC matrisinin her bir satırı bir görevin bir makinede çalışma süresini içermektedir. Şekil 3.2.'de ETC matrisinin yapısı yer almaktadır.

ETC (t1, m1)
ETC (t1, m2)
…
ETC (t1, mM)
ETC (t2, m1)
ETC (t2, m2)
…
ETC (tT, m1)
…

Şekil 3.2. ETC matris modeli (Anonymous 2012)

Veri setlerinin her biri $\boldsymbol{u\_x\_yyzz.k}$ şeklinde isimlendirilmektedir. Bu isimlendirmede;

- $\boldsymbol{u}$: matris içerisinde eşit bir dağılımın olduğunu,
- $\boldsymbol{x}$**:** tutarlılık türünü,
- $\boldsymbol{yy}$**:** görev heterojenliğini,
- $\boldsymbol{xx}$**:** makine heterojenliğini,
- $\boldsymbol{k}$**:** aynı tür örnek sayısını ifade etmektedir.

Bu veri setleri tutarlılık (*consistency*), görev heterojenliği (*task heterogeneity*) ve makine heterojenliği (*machine heterogeneity*) yönünden üç gruba ayrılmaktadır. Ayrıca tutarlılık bakımından veri setleri tutarlı (*consistent*), tutarsız (*inconsistent*) ve yarı tutarlı (*semi consistent*) olmak üzere üç alt grupta değerlendirilmektedir.

Bir M_j makinesi T_i görevini M_k makinesinden daha hızlı çalıştırıyorsa M_j makinesi tüm görevleri M_k makinesinden daha hızlı çalıştıracaktır (Braun *et al.* 2001). Bu veri setinin tutarlı olduğu durumdur ve Çizelge 3.1'de gösterilmektedir.

Çizelge 3.1. Tutarlı (*Consistent*) veri seti modeli

$Tutarlı$	M_j	M_k	M_x
T_i	52.104389	52.315109	52.619488
T_{i+1}	68.669853	84.113258	108.539917
T_{i+2}	59.289188	63.177467	85.315323

Veri setinin yarı tutarlı olduğu durum ise M_j makinesinin görevleri çalıştırması açısından M_k makinesine göre tutarlı M_x makinesine göre tutarsız olması durumudur ve Çizelge 3.2'de gösterilmektedir.

Çizelge 3.2. Yarı tutarlı (*Semi Consistent*) veri seti modeli

$Yarı\ Tutarlı$	M_j	M_k	M_x
T_i	30165.400391	2534201.250	273580.250
T_{i+1}	11210.348633	792464.062500	99401.898438
T_{i+2}	9378.622070	57338.406250	45316.957031

Eğer M_j makinesi M_k makinesine göre bazı görevleri daha hızlı bazılarını daha yavaş çalıştırıyorsa bu durum veri setinin tutarsız olduğunu göstermektedir (Braun *et al.* 2001). Çizelge 3.3'de tutarsız bir veri seti gösterilmektedir.

Çizelge 3.3. Tutarsız (*Inconsistent*) veri seti modeli

$Tutarsız$	M_j	M_k	M_x
T_i	25607.804688	7109.959473	18425.058594
T_{i+1}	885.655945	296.813293	205.983322
T_{i+2}	50.374516	312.408783	221.202606

Görevlere bağlı olarak makinelerin görevleri çalıştırma sürelerindeki değişiklik makine heterojenliği olarak geçmektedir. Eğer sistemde benzer özellikte görevler varsa makine heterojenliğinin düşük (*low machine heterogeneity*), sistemde farklı görevler ve süper bilgisayarların var olması gibi durumlarda ise makine heterojenliğinin yüksek (*high machine heterogeneity*) olduğu belirtilmektedir. Bir sistemin görev heterojenliği sistemdeki görevlerin hesaplanma gereksinimleri ile ilişkilendirilmektedir. Eğer sistemde görevlerin hesaplanma gereksinimleri büyük ölçüde değişiyorsa yüksek görev heterojenliği (*high task heterogeneity*), tipik olarak benzer karmaşıklığa sahip problemlerin çözümü aranıyorsa düşük görev heterojenliği (*low task heterogeneity*) söz konusudur (Ali *et al.* 2000). Çizelge 3.4'de görevlerin ve makinelerin yüksek ve düşük heterojenlik durumları gösterilmektedir.

Çizelge 3.4. Görev ve makine heterojenliği

$Heterojenlik$	M_j	M_k	$Makine\ Yüksek$	$Makine\ Düşük$	$Görev\ Yüksek$	$Görev\ Düşük$
T_x	91753.19531	263115.43750	✓	✗	✓	✗
T_y	12.585024	131.499863	✓	✗	✗	✓
T_z	1275.370117	2571.468750	✗	✓	✓	✗
T_w	52.104389	52.315109	✗	✓	✗	✓

3.3. Giriş Dosyasına Erişim

Tez kapsamında geliştirilen bilgisayar programında veri setlerindeki verilere C++ dilinde yer alan ve aşağıda belirtilen dosya işleme komutları kullanılarak ulaşılmaktadır.

- **fstream:** Dosya işlemlerini yapabilmek için gerekli olan kütüphanedir.
- **ifstream:** Dosyadan veri okumaya yarayan sınıftır.
- **open():** Dosyayı açmak için kullanılan fonksiyondur.
- **fail():** Dosyanın açılıp açılmadığını kontrol eden fonksiyondur.
- **exit():** Programı sonlandıran fonksiyondur.
- **ios::in:** Dosyadan veri okumayı sağlayan moddur.
- **c_str():** string sınıfından bir nesneyi C stil karakter dizisine dönüştüren fonksiyondur.
- **close():** Dosyayı kapatmak için kullanılan fonksiyondur.

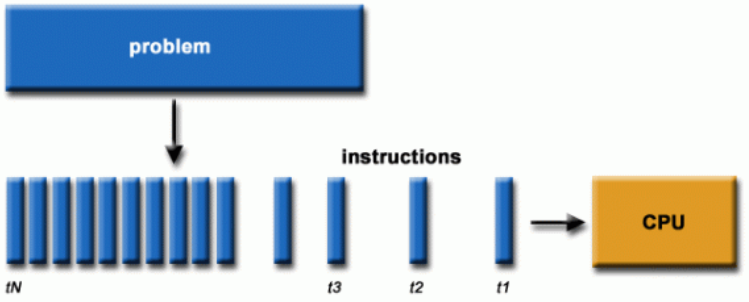
Şekil 3.3'de görüldüğü gibi dosyadan okunan ilk değer her bir veri setinde yer alan ortak görev sayısını (*numberOfTasks*), ikinci okunan değer ise ortak makine sayısını (*numberOfNodes*) ifade etmektedir. Daha sonra sırası ile okunan her bir değer görevlerin makinelerde çalışma sürelerini ifade etmekte ve iki boyutlu ETC matrisinde tutulmaktadır.

```
Begin
1       fileName ← "u_x_yyzz.k";
2       inputFile ← INIT-FILE;
3       inputFile.open(fileName);
4       if inputFile.fail()
5           exit(1);
6       inputFile >> numberOfTasks;
7       inputFile >> numberOfNodes;
8       for i ← 0 to numberOfTasks
9           for j ← 0 to numberOfNodes
10              inputFile >> ETC[i][j];
11      inputFile.close();
End
```

Şekil 3.3. Dosyadan veri okumanın sözde kodu

3.4. Seri Veri İşleme

Bir bilgisayar ve bir işlemci kullanılarak farklı komut serilerini çalıştırma işlemine seri veri işleme denilmektedir. Şekil 3.4'de görüldüğü gibi her bir komut bir diğerinden sonra işletilerek birim zamanda tek bir komut çalıştırılmaktadır.



Şekil 3.4. Seri veri işleme (Anonim 2013)

3.5. Metasezgisel Yaklaşım (Metaheuristic Approach)

Metasezgisel yaklaşım bir arama uzayında farklı yerel çözümlerden yola çıkarak her adımda üretilen yeni çözümler sonucunda global çözüme yaklaşılmasını sağlayan tekrarlı bir üretim sürecidir. Üst seviye sezgisel bir yaklaşım olan metasezgisel yaklaşımın amacı en etkili çözüme ulaşmak için arama alanını verimli bir şekilde keşfetmektir.

Metasezgisel yaklaşım çözüm arayışını yönlendirmek ve çeşitlendirmek için rastgele bileşenlere sahiptir. Çözüm için global değeri bulmayı garanti etmemekte ancak globale yakın çözümler bulabilmektedir. Çeşitliliği sağlamak için genellikle rastgele üretilen bir başlangıç çözümü ile aramaya başlamaktadır. Yerel optimuma takılmak yerine global optimumun bulunduğu arama alanı bölgelerini komşu çözümler üreterek keşfetmeyi denemektedir. Görev çizelgeleme problemi yüksek karmaşıklığa sahip olduğu için optimal çözümü belirlemek mümkün değildir, çünkü büyük arama alanı makul sürede tam olarak keşfedilmediğinden yüksek hesaplama verimliliği ile kaliteli çözümler bulmak için metasezgisel yaklaşım kullanılmaktadır (Flórez *et al.* 2015).

3.6. Benzetilmiş Tavlama (Simulated Annealing)

Benzetilmiş tavlamanın kökeni 1953 yılında Nicholas Metropolis ve arkadaşları tarafından geliştirilen Metropolis algoritmasına dayanmaktadır. Şekil 3.5'de görüldüğü gibi Metropolis algoritması başlangıçta belirlediği bir çözümden yola çıkarak rastgele yeni bir çözüm oluşturmaktadır. Daha sonra bu çözümlerin her ikisini de kullanarak bir enerji ΔC değeri hesaplamakta ve ilk aşamada bu değere bağlı olarak yeni çözümün kabulüne karar vermektedir. İkinci aşamada bir sıcaklık T değerini, sabit bir değeri (*Boltzmann constant*) K_B ve enerjiyi ΔC kullanarak karar durumu oluşturmaktadır (Merendino and Celebi 2013).

```
Begin
1       T ← INIT-TEMP;
2       scheduling ← INIT-SCHEDULING;
3       RANDOM(scheduling);
4       schedulingCost←COST(scheduling);
5       newScheduling ← PERTURB(scheduling);
6       ΔC ← newSchedulingCost-schedulingCost;
7       if ΔC < 0 then
8           scheduling ← newScheduling;
9       else if ΔC > 0 then
10          if exp(−K_B.ΔC/T) then
11              scheduling ← newScheduling;
End
```

Şekil 3.5. Metropolis algoritmasının sözde kodu (Merendino and Celebi 2013)

Minimumum arayışında olan problemlerde global minimumu bulan Metropolis algoritmasının birkaç eksikliği mevcuttur. Sıcaklığı temsil eden *T* değerinin sabit kalması ve iterasyon sayısının belirsizliği algoritmanın temel eksiklikleridir. Benzetilmiş tavlamanın 1982 yılının sonlarına doğru Scott Kirkpatrick ve arkadaşlarının Metropolis algoritmasının eksikliklerini gidermesi sonucu optimizasyon problemlerine uygulanabilir olduğu ve Şekil 3.6'da gösterilen günümüz şeklini aldığı bilinmektedir.

```
Begin
1       T ← INIT-TEMP;
2       scheduling ← INIT-SCHEDULING;
3       bestScheduling ← INIT-SCHEDULING;
4       RANDOM(scheduling);
5       schedulingCost ← COST(scheduling);
6       bestSchedulingCost ← schedulingCost;
7       while T > ε do
8           while inner_loop_criteria ← FALSE do
9               newScheduling ← PERTURB(scheduling);
10              newSchedulingCost ← COST(newScheduling);
11              ΔC ← newSchedulingCost - schedulingCost;
12              if ΔC < 0 then
13                  scheduling ← newScheduling;
14                  schedulingCost ← newSchedulingCost;
15                  if newSchedulingCost < bestSchedulingCost then
16                      bestSchedulingCost ← newSchedulingCost;
17                      bestScheduling ← newScheduling;
18              else if RANDOM(0,1) < e^(−(ΔC/T)) then
19                  scheduling ← newScheduling;
20                  schedulingCost ← newSchedulingCost;
21              T ← SCHEDULE(T);
22          end while
23      end while
End
```

Şekil 3.6. Benzetilmiş tavlama yaklaşımının sözde kodu

Benzetilmiş tavlama, demirin daha kullanışlı hale getirilmesi için yapılan ve ısısal bir işlem olan tavlamadan ilham alınarak geliştirilmiştir. Tavlama, demirin maksimum bir seviyeye kadar hızlıca ısıtılıp daha sonra yavaşça soğutulması mantığına dayanmaktadır. Isıtılma sırasında demir içerisindeki parçacıklar rastgele sıvı hale dönüşmekte ve

maksimum sıcaklığa ulaşınca uygun şekilde soğutulma yapılması halinde düzenli yapıda kristal parçacıklar oluşmaktadır. Benzetilmiş tavlama yöntemi de, yüksek bir sıcaklıkta, eldeki problemi temsil edecek rastgele bir ilk çözümle (mevcut) yola koyulmaktadır. Takip eden her bir adımda da bu mevcut çözümü küçük değişiklikler yaparak yeni bir çözüme (sonraki) dönüştürmekte ve sonrasında sıcaklığı azaltmaktadır. Maliyeti mevcut çözümden küçük olan sonraki çözümler koşulsuz, öte yandan maliyeti mevcut çözümden büyük olan sonraki çözümler ise olasılıksal olarak kabul edilmektedir. Bu sayede yaklaşım yerel optimuma takılmaktan kurtulmaktadır. Benzetilmiş tavlama yaklaşımı aşağıda yer alan dört farklı parametreyi kullanmaktadır.

- **Sıcaklık (T):** Sıcaklık değerini ifade etmektedir.
- **Alfa (α):** Soğutma oranını ifade etmektedir.
- **Epsilon (ε):** Donma noktasını ifade etmektedir.
- **İterasyon sayısı (S):** Her bir sıcaklık için gereken döngü sayısını ifade etmektedir.

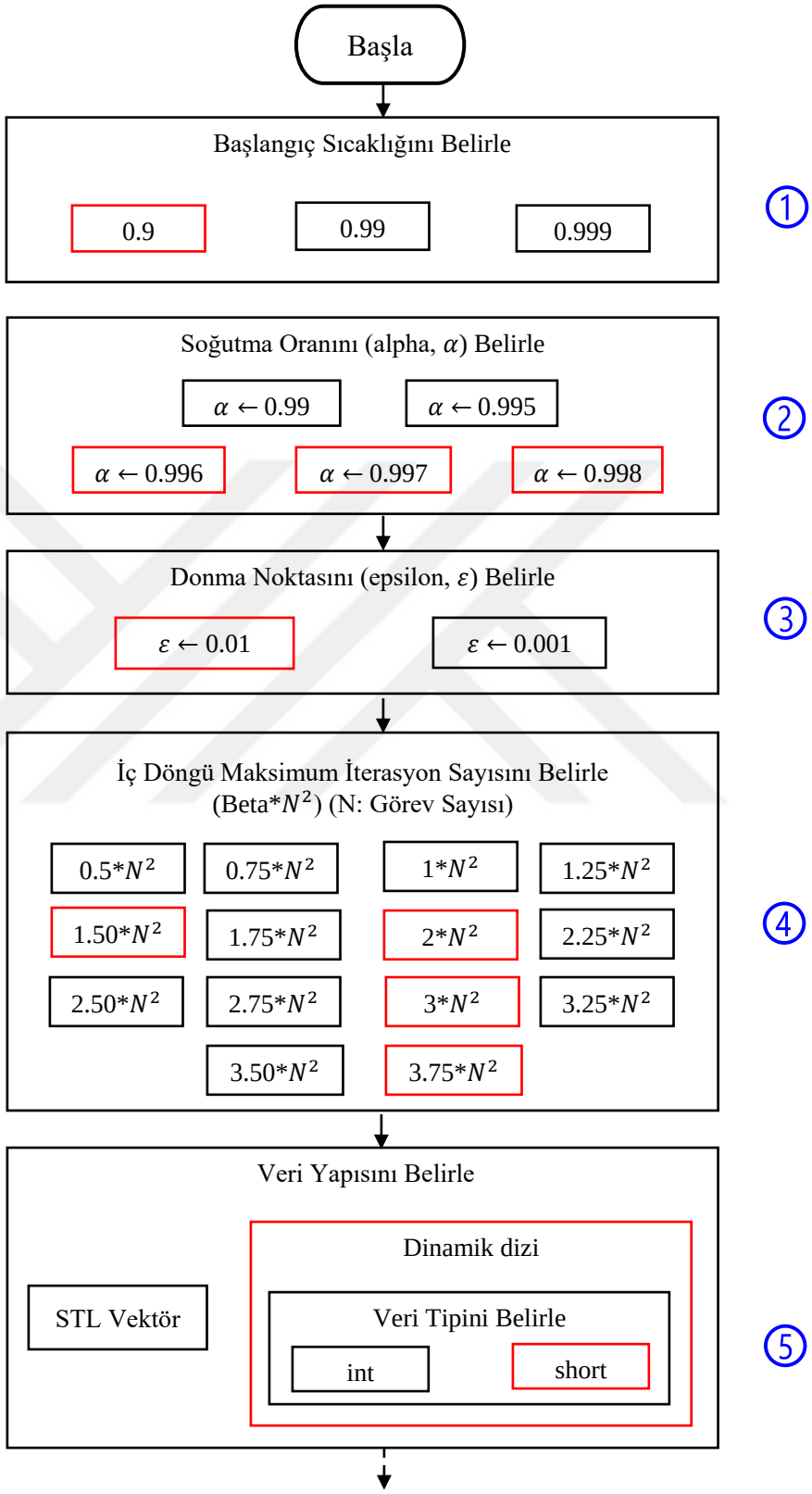
Ayrıca programın çalışması esnasında aşağıda yer alan üç farklı tanımdan faydalanılmaktadır.

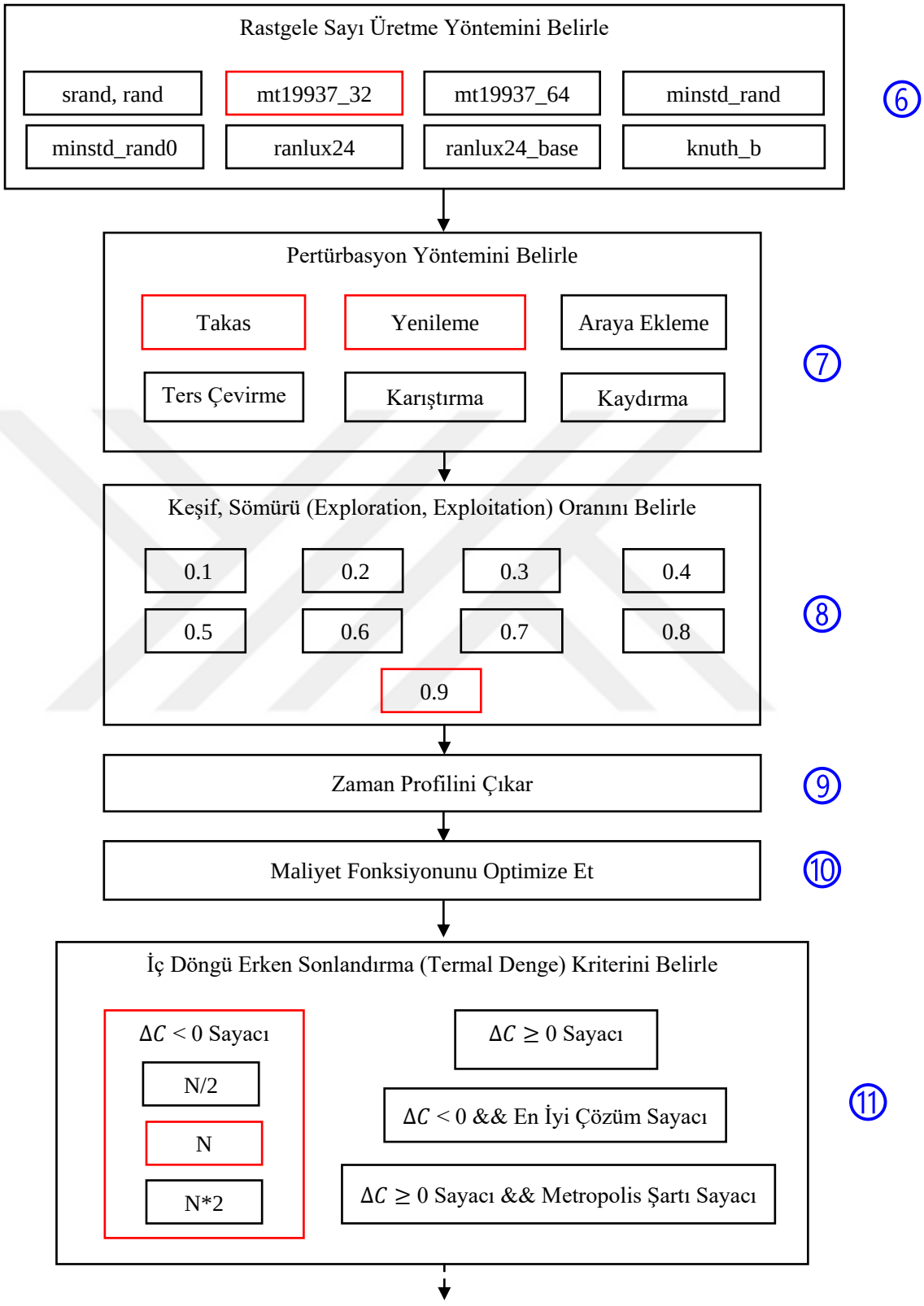
- **Delta (ΔC):** Mevcut çözümün ve yeni çözümün maliyetleri arasındaki farkı belirtmektedir.
- **Boltzmann dağılımı (P):**Yeni çözümün kabulü için kıyaslamada kullanılan değerdir.
- **Maliyet fonksiyonu (C_x):** x çözümü için maliyet değerini hesaplayan fonksiyondur.

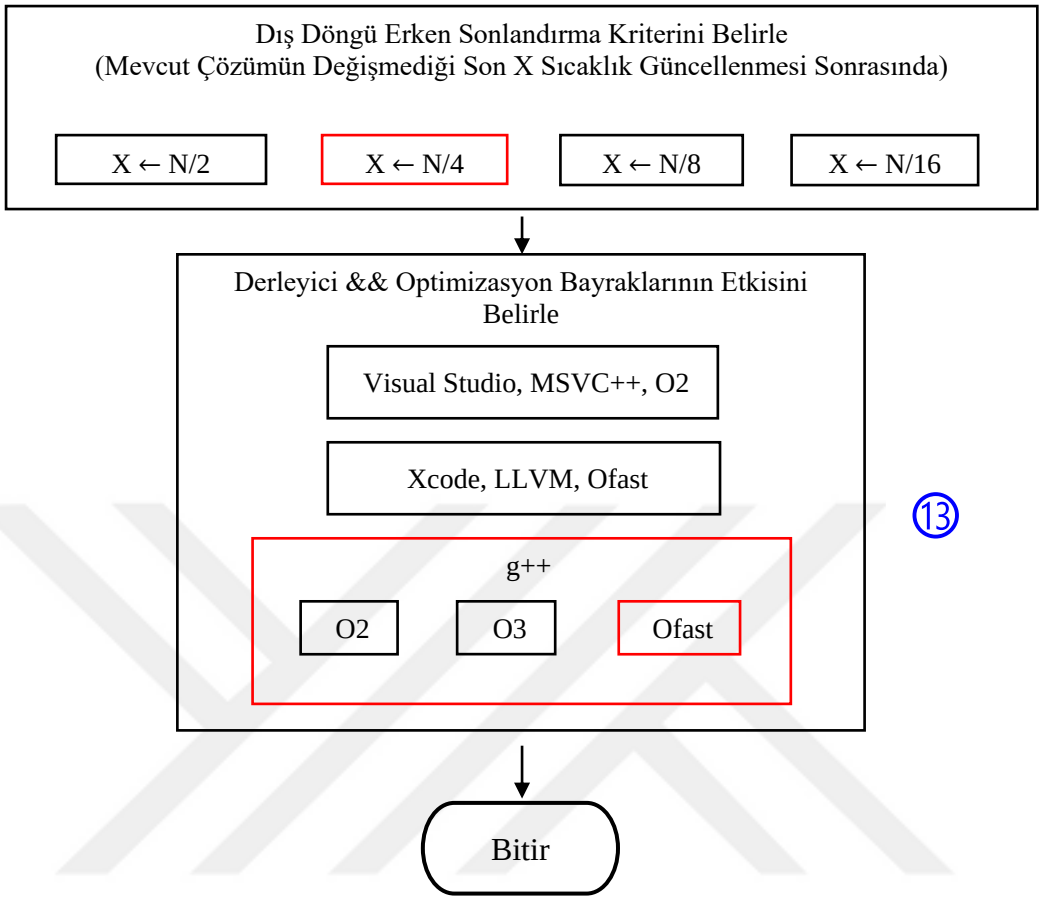
Benzetilmiş tavlamanın başlıca avantajları arasında, global en iyi çözümü bulma yeteneğine ve kolaylıkla düzenlenebilir bir yapıya sahip olmasının yanı sıra çok yönlülüğü sayesinde birden fazla probleme uyarlanabilirliği yer almaktadır. Çözüme ulaşmak için zamana ihtiyaç duyması ve uygun parametre seçiminin zaman alması ise başlıca dezavantajları arasında bulunmaktadır.

Benzetilmiş tavlamanın uygulama alanları arasında görev çizelgeleme, paketleme problemleri, seyahat problemleri, elektronik devre tasarımı, yol bulma problemleri, görüntü işleme ve malzeme fiziği benzetimi yer almaktadır.

Bu tez kapsamında bulutta görev çizelgeleme probleminin çözümüne yönelik geliştirilen benzetilmiş tavlama tabanlı optimizasyon yaklaşımına ait akış diyagramı Şekil 3.7 ile verilmiştir ve bu diyagramdaki her bir adım takip eden alt bölümlerde detaylandırılmıştır.


Başla
Başlangıç Sıcaklığını Belirle
0.9
0.99
0.999
①
Soğutma Oranını (alpha, α) Belirle
α ← 0.99
α ← 0.995
α ← 0.996
α ← 0.997
α ← 0.998
②
Donma Noktasını (epsilon, ε) Belirle
ε ← 0.01
ε ← 0.001
③
İç Döngü Maksimum İterasyon Sayısını Belirle
(Beta*N²) (N: Görev Sayısı)
0.5*N²
0.75*N²
1*N²
1.25*N²
1.50*N²
1.75*N²
2*N²
2.25*N²
2.50*N²
2.75*N²
3*N²
3.25*N²
3.50*N²
3.75*N²
④
Veri Yapısını Belirle
STL Vektör
Dinamik dizi
Veri Tipini Belirle
int
short
⑤


Rastgele Sayı Üretme Yöntemini Belirle
srand, rand
mt19937_32
mt19937_64
minstd_rand
minstd_rand0
ranlux24
ranlux24_base
knuth_b
⑥
Pertürbasyon Yöntemini Belirle
Takas
Yenileme
Araya Ekleme
Ters Çevirme
Karıştırma
Kaydırma
⑦
Keşif, Sömürü (Exploration, Exploitation) Oranını Belirle
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
⑧
Zaman Profilini Çıkar
⑨
Maliyet Fonksiyonunu Optimize Et
⑩
İç Döngü Erken Sonlandırma (Termal Denge) Kriterini Belirle
ΔC < 0 Sayacı
N/2
N
N*2
ΔC ≥ 0 Sayacı
ΔC < 0 && En İyi Çözüm Sayacı
ΔC ≥ 0 Sayacı && Metropolis Şartı Sayacı
⑪



Şekil 3.7. Benzetilmiş tavlama tabanlı optimizasyon yaklaşımının akış diyagramı

3.6.1. Başlangıç sıcaklığı (Initial temperature)

Metropolis algoritmasından benzetilmiş tavlamaya geçilmesine sebep olan eksiklik sıcaklıktan kaynaklanmaktadır. Metropolis algoritmasında arama boyunca T sabit kalmaktadır. Sabit olan bu T değerinin küçük seçilmesi yerel minimum değerlerden kurtulmayı önlerken, büyük seçilmesi global minimumu veren çözüme ulaşma hassasiyetini düşürmektedir.

Geliştirilen bilgisayar programında Şekil 3.7'de 1 numaralı mavi kutu ile belirtilen aşamada bu soruna çözüm olarak arama alanını iyice keşfetmeye yetecek yüksek bir T değeri (*hot enough*) aşağıda yer alan adımlar ile belirlenmiştir.

1. Benzetilmiş tavlamanın en önemli özelliği yerel optimumdan kaçabilmek amacıyla mevcut çözümden (*current*) daha kötü (maliyet fonksiyonunda artışa sebep olan) sonraki (*next*) çözümlerin de olasılıksal olarak kabul edilmesidir. Bu sebeple Metropolis kriterinde yer alan ΔC ve T değerleri ile üstel olarak hesaplanan Boltzmann dağılımından faydalanılmaktadır. Bu dağılım (P), [0,1) aralığında rastgele üretilen ondalıklı bir değer ile kıyaslanarak maliyet fonksiyonunda artışa sebep olan çözümlerin de kabul edilmesi sağlanmaktadır.

$$random \leftarrow [0,1)$$
$$P \leftarrow e^{\frac{-\Delta C}{T}}$$
$$random < P$$

2. Yukarıda gösterilen kıyaslamadan yola çıkılarak yerel optimuma takılmayı önlemek için Boltzmann dağılımının aşağıda belirtildiği şekilde yaklaşık olarak 1'e çok yakın bir değere eşit olması gerekmektedir.

$$P \cong 1$$
$$e^{\frac{-\Delta C}{T}} \cong 1$$

3. Dağılımda kullanılan parametrelerden birisi sıcaklıktır. Geliştirilen bilgisayar programında Şekil 3.7'de 1 numaralı mavi kutu ile belirtilen aşamada başlangıç sıcaklığını belirlemek için yukarıda belirtilen eşitlik kullanılmıştır ve aşağıda yer alan eşitlikte T'nin ΔC ve sabit bir değere ($\ln 1$) bağlı olduğu görülmüştür.

$$\ln e^{\frac{-\Delta C}{T}} \cong \ln 1$$
$$\frac{-\Delta C}{T} \ln e \cong \ln 1$$
$$\frac{-\Delta C}{T} \cong \ln 1 \quad (\ln e = 1)$$
$$T \cong \frac{-\Delta C}{\ln 1}$$

4. Her bir benchmark için ortalama bir başlangıç sıcaklığı öncelikle Şekil 3.8'de yer alan kod bloğu bir eşik değere (*threshold*) ulaşılıncaya kadar çalıştırılarak ΔC'nin pozitif olduğu durumlar için bir ortalama değer elde edilmiştir. Daha sonra bu ortalama ΔC değeri

$$T \cong \frac{-\Delta C}{\ln 1}$$

formülünde kullanılarak başlangıç sıcaklığı tespit edilmiştir. Bu denklemde $\ln 1$ yerine $\ln 0.9$, $\ln 0.99$ ve $\ln 0.999$ değerleri kullanılmıştır; en iyi sonuçların ise $\ln 0.9$ ile elde edildiği gözlemlenmiştir. Böylelikle veri setlerinin her biri için manuel sıcaklık girişi yapılması önlenmiştir.

```
Begin
1       scheduling ← INIT-SCHEDULING;
2       RANDOM(scheduling);
3       schedulingCost ← COST(scheduling);
4       while (true) do
5           nextScheduling ← scheduling;
6           PERTURB(nextScheduling);
7           nextSchedulingCost ← COST(nextScheduling);
8           ΔC ← nextSchedulingCost - schedulingCost;
9           if ΔC ≥ 0 then
10              scheduling ← nextScheduling;
11              schedulingCost ← nextSchedulingCost;
12              counter++;
13              totalCost + = ΔC;
14          if counter = threshold then
15              break;
16      return (- totalCost / counter) / log(0.9);
End
```

Şekil 3.8. Başlangıç sıcaklığını belirlemenin sözde kodu

3.6.2. Soğutma zamanlaması (Cooling schedule)

Başlangıç sıcaklığının ve sonrasında mevcut sıcaklığın iç döngü iterasyon sayısı üst sınırına ulaştığında soğutma oranı (α) kullanılarak düzenli şekilde düşürülmesine soğutma zamanlaması denilmektedir. Soğutma zamanlaması için literatürde farklı

yöntemler mevcuttur. Bu yöntemlerin en popülerlerinden birisi olan sıcaklığın belli bir α miktarı ile çarpılarak geometrik olarak azaltılmasını sağlayan geometrik soğutma yöntemi tercih edilmiştir. (Bkz. Şekil 3.8'deki 2 numaralı mavi kutucuk)

$$T = T * \alpha$$

Yine Şekil 3.7'de 2 numaralı mavi kutu ile belirtilen aşamada kırmızı kutu içerisinde yer alan ve $0 < \alpha < 1$ aralığındaki üç farklı değer soğutma oranı olarak kullanılmıştır. Sıcaklık değişimini dengede tutmak amacıyla başlangıçta belirlenen bu değerler arama boyunca da sabit tutulmuştur. Benzetilmiş tavlamada soğutma oranının geometrik soğutma yöntemi ile azaltılmasına ait sözde kod Şekil 3.9 ile verilmiştir.

```
Begin
1       T ← INIT-TEMP;
2       while T > ε do
3           while  inner_loop_criteria = TRUE do
¦                       ¦
¦                       ¦
n-2         end while
n-1         T ←T * α;
 n      end while
End
```

Şekil 3.9. Geometrik soğutmanın sözde kodu

3.6.3. Durdurma kriteri (Stopping criterion)

Şekil 3.7'de 3 numaralı mavi kutu ile belirtilen aşamada programın sonlandırılmasına karar vermek için aynı aşamada kırmızı kutu içerisinde yer alan ve başlangıçta belirlenen donma noktası (ε) kullanılmıştır. Bu donma noktası sıcaklığın soğutma oranı ile azaltılması sonucu dış döngüdeki sıcaklık ile kıyaslanmıştır. Sıcaklığın donma noktasının altına düşmesi durumunda program sonlandırılmıştır. Benzetilmiş tavlamada durdurma kriterinin uygulanışına ait sözde kod Şekil 3.10 ile verilmiştir.

```
Begin
1       T ← INIT-TEMP;
2       ε = 0.01
3       while T > ε do
        ⋮                    ⋮
n-1                 T ←T*α;
 n      end while
End
```

Şekil 3.10. Durdurma kriterinin sözde kodu

3.6.4. İterasyon sayısı

Benzetilmiş tavlamada her bir sıcaklık değerinde pertürbasyon işleminin kaç kez gerçekleştirileceğini belirlemek amacıyla iterasyon sayısı kullanılmaktadır. İterasyon sayısı da sıcaklık gibi arama alanının iyice keşfedilmesinde önemlidir. Bu nedenle Şekil 3.7'de 4 numaralı mavi kutu ile belirtilen aşamada aynı sıcaklıkta aramayı daha fazla yapmak adına iterasyon sayısını belirlemek için veri setlerinde yer alan ortak görev sayısına (*N*) ve *Beta* katsayısına bağlı bir sezgisel formül üretilmiştir.

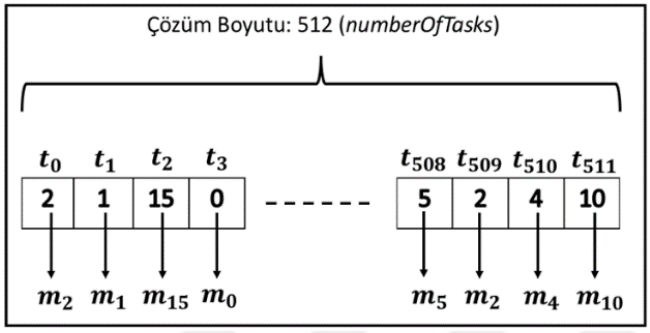
$$Beta * N^2$$

Yukarıda belirtilen formülden yola çıkılarak Şekil 3.7'de 4 numaralı mavi kutu ile belirtilen aşamada yer alan on dört farklı değer arasından kırmızı kutu içerisindekilerden faydalanılmıştır.

3.6.5. Başlangıç çözümü (Initial solution)

Benzetilmiş tavlama ile optimizasyon problemlerini çözmek için ilk olarak rastgele bir başlangıç çözümü oluşturulmaktadır. Bu çözümün boyutu problemin türüne göre farklılık göstermektedir. Başlangıç çözümü olası doğru çözümü temsil etmekle birlikte arama süresince daha iyi bir çözüm bulunması halinde değişmektedir. Programda Şekil 3.7'de 5 numaralı mavi kutu ile belirtilen aşamada belirtilen iki farklı veri yapısı arasından başlangıç çözümünü temsil etmek için kırmızı kutu içerisinde yer alan tek boyutlu dinamik dizi yapısından faydalanılmıştır. Ayrıca çözümü temsil etmek için veri setlerinde

yer alan ortak görev sayısı boyutunda Şekil 3.11'de gösterilen görev odaklı çözüm temsili kullanılmıştır.



Şekil 3.11. Başlangıç çözümü temsili

3.6.6. Rastgele sayı üretme teknikleri

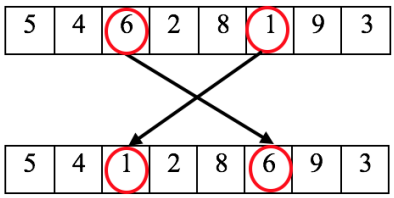
Benzetilmiş tavlama yaklaşımında başlangıç çözümü ve diğer fonksiyonlar rastgele sayılara ihtiyaç duymaktadır. Şekil 3.7'de 6 numaralı mavi kutu ile belirtilen aşamada programın rastgele sayı ihtiyacını karşılamak için sekiz farklı rastgele sayı üreteci test edilmiştir. Bu rastgele sayı üreteçlerinden *rand*, C'nin *cstdlib* kütüphanesinde tanımlıdır ve her seferinde farklı sayılar üretmek için *srand*'in zamana bağlı ürettiği tohumu (rastgele sayı üreteci algoritması tarafından kullanılacak bir tam sayı değerini) girdi olarak kullanmaktadır. rand'in çıktısı ise programın ihtiyaç duyduğu rastgele sayılardır.

Geri kalan yedi rastgele sayı üreteci C++'ın *random* kütüphanesinde tanımlıdır. Bu rastgele sayı üreteçlerinin her biri *random_device*'dan sağladığı tohumu (rasgele sayı üreteci algoritması tarafından kullanılacak bir tam sayı değerini) girdi olarak kullanmakta ve çıktı olarak program için gerekli rastgele sayıyı üretmektedir. Bu iki farklı kütüphaneye ait rastgele sayı üreteçleri arasından programda Şekil 3.7'de 6 numaralı mavi kutu ile belirtilen aşamada kırmızı kutu içerisinde yer alan *mt19337_32* üreteci tercih edilmiştir. Çözüm, görev odaklı olduğu için her bir görevi çalıştırmak üzere [0, N) aralığında bu üreteç ile üretilen makine indis değerleri çözüme yerleştirilmiştir.

3.6.7. Pertürbasyon yöntemi (Perturbation method)

Başlangıç çözümü ile başlayan ve global optimuma ulaşmak için arama boyunca üretilmesi gereken çok sayıda komşu çözüme ihtiyaç duyulmaktadır. Bu yüzden Şekil 3.7'de 7 numaralı mavi kutu ile belirtilen aşamada mevcut çözüm üzerinde küçük ama anlamlı birtakım değişiklikler yaparak komşu çözümler elde edilmiştir. Bu küçük değişiklikler için altı farklı pertürbasyon yönteminden faydalanılmıştır.

- **Takas (Swap, Exchange):** Bu yöntem ile pertürbasyonda mevcut çözüm üzerinde Şekil 3.12'de görüldüğü gibi rastgele iki görev indisi seçilmekte ve bu indislerde yer alan düğüm değerleri yer değiştirmektedir.



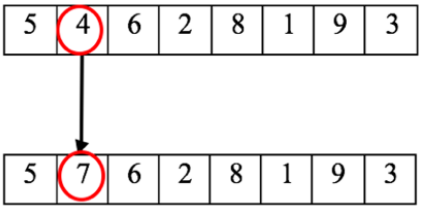
Şekil 3.12. Takas yöntemi

swap operatörü kullanılarak takas yöntemi ile pertürbasyon yapılmasına ait sözde kod Şekil 3.13'de verilmiştir.

```
Begin
1       randomIndex1 ← RANDOM(0, numberOfTasks-1);
2       randomIndex2 ← RANDOM(0, numberOfTasks-1);
3       swap(scheduling[randomIndex1], scheduling[randomIndex2]);
End
```

Şekil 3.13. Takas yönteminin sözde kodu

- **Yenileme:** Bu yöntem ile pertürbasyonda mevcut çözüm üzerinde Şekil 3.14'de görüldüğü gibi rastgele bir görev indisi seçilmekte ve bu indisde yer alan düğüm değeri yine rastgele belirlenen bir başka düğüm değeri ile güncellenmektedir.



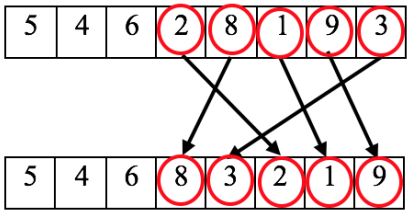
Şekil 3.14. Yenileme yöntemi

Yenileme yöntemi kullanılarak pertürbasyon yapılmasına ait sözde kod Şekil 3.15'de verilmiştir.

```
Begin
1       randomIndex1 ← RANDOM(0, numberOfTasks-1);
2       newNode ← RANDOM(0, numberOfNodes-1);
3       scheduling[randomIndex1] ← newNode;
End
```

Şekil 3.15. Yenileme yönteminin sözde kodu

- **Karıştırıcı (Random shuffle, Scramble):** Bu yöntem ile pertürbasyonda mevcut çözüm üzerinde Şekil 3.16'da görüldüğü gibi rastgele iki görev indisi seçilmekte ve bu indisler arasında kalan görevlerin atandığı düğümler rastgele karıştırılmaktadır.



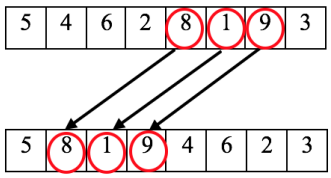
Şekil 3.16. Karıştırıcı yöntemi

Karıştırıcı yöntemi kullanılarak pertürbasyon yapılmasına ait sözde kod Şekil 3.17'de verilmiştir.

```
Begin
1       randomIndex1 ← RANDOM(0, numberOfTasks-1);
2       randomIndex2 ← RANDOM(0, numberOfTasks-1);
3       if  randomIndex1 > randomIndex2 then
4            swap(randomIndex1, randomIndex2);
5       random_shuffle(scheduling.begin()+randomIndex1, scheduling.begin()+randomIndex2 +1)
End
```

Şekil 3.17. Karıştırıcı yönteminin sözde kodu

- **Kaydırma (Displacement):** Bu yöntem ile pertürbasyonda mevcut çözüm üzerinde Şekil 3.18'de görüldüğü gibi rastgele iki görev indisi seçilmekte ve bu indisler de dahil olmak üzere arada kalan düğüm değerleri yine rastgele belirlenen bir konuma kaydırılmaktadır.



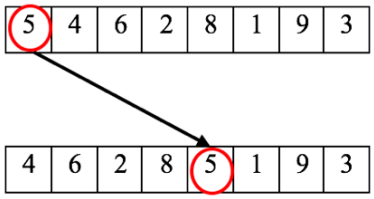
Şekil 3.18. Kaydırma yöntemi

Kaydırma yöntemi kullanılarak pertürbasyon yapılmasına ait sözde kod Şekil 3.19'da verilmiştir.

```
Begin
1       randomIndex1 ← RANDOM(0, numberOfTasks-1);
2       randomIndex2 ← RANDOM(0, numberOfTasks-1);
3       if  randomIndex1 > randomIndex2 then
4            swap(randomIndex1, randomIndex2);
5       displacementTourSize ← (randomIndex2 - randomIndex1)+1;
6       displacementTour.resize(displacementTourSize,0);
7       for  i ← 0  to  displacementTourSize
8            displacementTour[i] ← scheduling[index1+i];
9       scheduling.erase(scheduling.begin()+randomIndex1, scheduling.begin()+randomIndex2+1);
10       randomIndex3 ← RANDOM(0, scheduling.size()-1);
11      for i ← 0  to  displacementSize
12           scheduling.insert(scheduling.begin()+(randomIndex3 + i), displacementTour[i]);
End
```

Şekil 3.19. Kaydırma yönteminin sözde kodu

- **Araya ekleme (Insertion):** Bu yöntem ile pertürbasyonda mevcut çözüm üzerinde Şekil 3.20'de görüldüğü gibi rastgele bir görev indisi seçilmekte ve bu indisde yer alan düğüm değeri yine rastgele belirlenen bir konumda araya eklenmektedir. Araya ekleme kaydırma yönteminin tek bir düğümle gerçekleştirilen özel bir hali olarak da düşünülebilir.



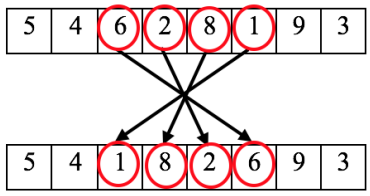
Şekil 3.20. Araya ekleme yöntemi

Araya ekleme yöntemi kullanılarak pertürbasyon yapılmasına ait sözde kod Şekil 3.21'de verilmiştir.

```
Begin
1       randomIndex1 ← RANDOM(0, numberOfTasks-1);
2       randomIndex2 ← RANDOM(0, numberOfTasks-1);
3       insertNode ← scheduling[randomIndex1];
4       scheduling.erase(scheduling.begin()+randomIndex1);
5       scheduling.insert(scheduling.begin()+randomIndex2,insertNode);
End
```

Şekil 3.21. Araya ekleme yönteminin sözde kodu

- **Ters çevirme (Inversion):** Bu yöntem ile pertürbasyonda mevcut çözüm üzerinde Şekil 3.22'de görüldüğü gibi rastgele iki görev indisi seçilmekte ve bu indisler dahil olmak üzere arada kalan düğüm değerlerinin tersine çevrilmesi sağlanmaktadır.



Şekil 3.22. Ters çevirme yöntemi

Ters çevirme yöntemi kullanılarak pertürbasyon yapılmasına ait sözde kod Şekil 3.23'de verilmiştir.

```
Begin
1       randomIndex1 ← RANDOM(0, numberOfTasks-1);
2       randomIndex2 ← RANDOM(0, numberOfTasks-1);
3       if  randomIndex1 > randomIndex2 then
4             swap(randomIndex1, randomIndex2);
5       reverse(scheduling.begin()+randomIndex1, scheduling.begin()+randomIndex1 +1);
End
```

Şekil 3.23. Ters çevirme yönteminin sözde kodu

Pertürbasyon yöntemlerinin her birinin çalıştırılması sonucu Şekil 3.7'de 7 numaralı mavi kutu ile belirtilen aşamada yer alan kırmızı kutular içerisinde gösterilen takas ve yenileme yönteminin en uygun yöntemler olduğu belirlenmiştir. Sonrasında geliştirilen programda küçük hareketlerle yeni çözümlere ulaşmak için bu iki yöntemin birlikte kullanılmasına karar verilmiştir. Takasın sömürme ve yenilemenin de keşif amacıyla kullanılabileceği değerlendirilmiştir.

Şekil 3.7'de 8 numaralı mavi kutu ile belirtilen aşamada takas ve yenileme yönteminin birlikte çalıştırılması için sömürü/keşif oranına (*exploitation/exploration ratio*) yer verilmiştir. Bu oran için yine aynı aşamada yer alan dokuz farklı değer arasından kırmızı kutu içerisinde gösterilen 0.9 değerinde karar kılınmıştır. Daha sonra bu oran [0,1) aralığında rastgele üretilen ondalıklı bir değer ile kıyaslanarak uygulanacak pertürbasyon yöntemine karar verilmiştir.

Sömürü/keşif oranı kullanılarak pertürbasyon yapılmasına ait sözde kod Şekil 3.24'de verilmiştir.

```
Begin
1       randomIndex1 ← RANDOM(0, numberOfTasks-1);
2       randomIndex2 ← RANDOM(0, numberOfTasks-1);
3       exploitationExplorationRatio ← 0.9;
4       if exploitationExplorationRatio < REAL_RANDOM(0, 1) then
5               swap(scheduling[randomIndex1], scheduling[randomIndex2]);
6         else
7               scheduling[randomIndex1] ← RANDOM(0, numberOfNodes-1);
End
```

Şekil 3.24. Sömürü/Keşif oranı ile kıyaslamanın sözde kodu

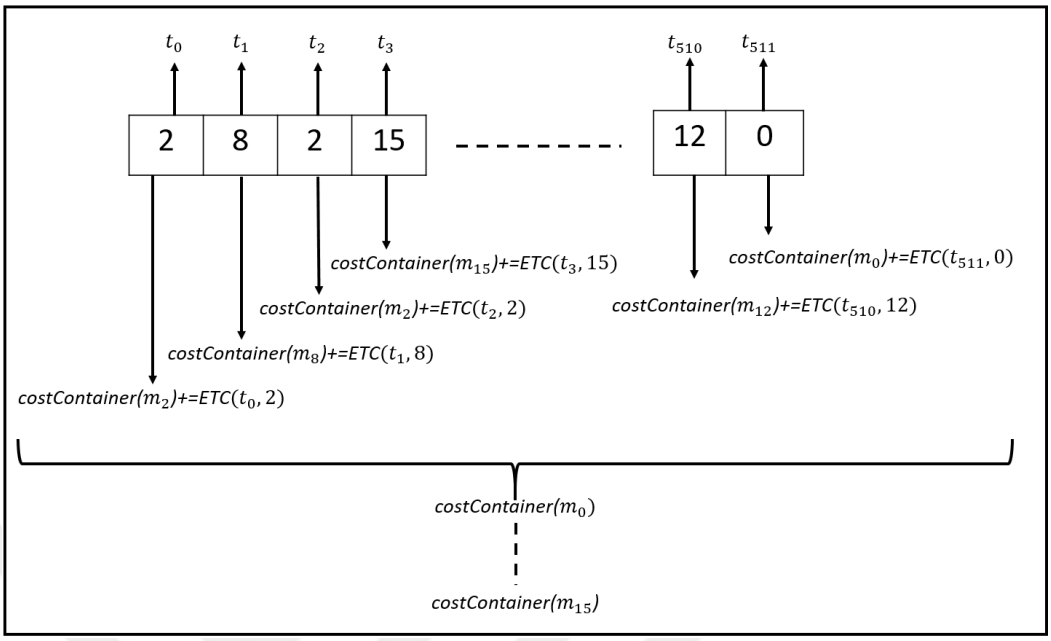
3.6.8. Maliyet fonksiyonu (Cost function)

Tavlama işlemi sonucu rastgele sıvı hale dönüşen parçacıkların dışarıya saldığı enerji miktarı maliyet değerini ifade etmektedir. Bu gözlemden esinlenen benzetilmiş tavlama yaklaşımında her iterasyonda üretilen çözümün maliyeti belirli matematiksel işlemler kullanılarak hesaplanmaktadır. Maliyet fonksiyonunun çalışma şekli aşağıda yer almaktadır.

1. Programda çözümün maliyetini hesaplamak için görevlerin düğümlerde çalışma sürelerinin yer aldığı ETC matrisinden faydalanılmaktadır.
2. Çözümde yer alan her bir düğümün kendisine atanan görevlerin çalıştırılma maliyetini saklamak için aşağıda belirtilen şekilde tek boyutlu bir dinamik dizi (*costContainer*) yapısı kullanılmaktadır.

$$costContainer[m_j] = \sum ETC[t_i][m_j]$$

3. Şekil 3.25'de görüldüğü gibi çözümden sırayla alınan her bir düğüm-görev eşleştirmesine ait maliyet ETC matrisinden temin edilerek maliyet dizisinde (*costContainer*) ilgili düğümün maliyetine eklenmektedir. Böylelikle çözümde yer alan her bir düğümün maliyeti çözümün tek seferde taranması ile hesaplanmaktadır.



Şekil 3.25. Maliyet hesaplama

Maliyet fonksiyonununa ait sözde kod Şekil 3.26'da verimiştir.

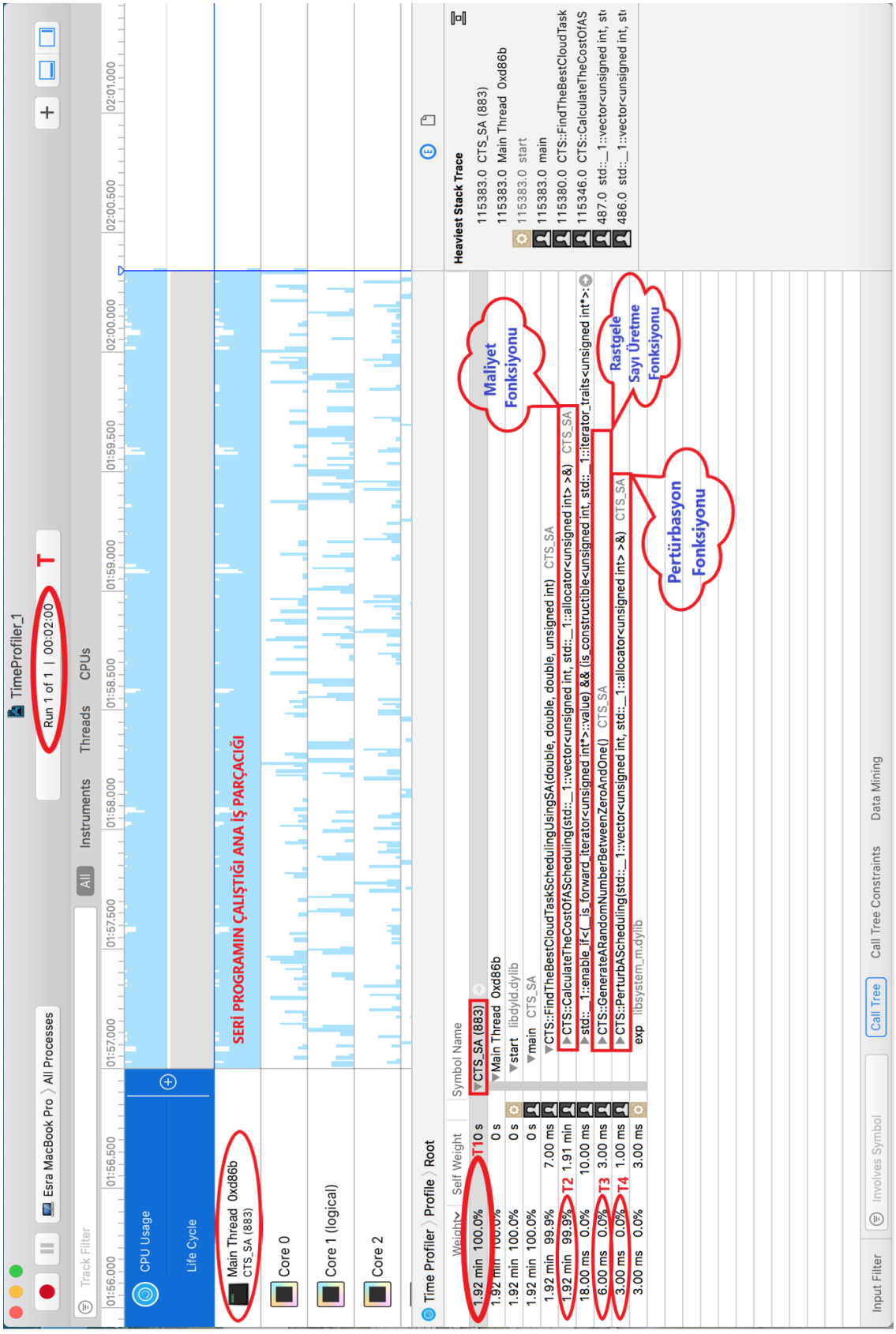
```
Begin
1       costContainer ← INIT-CONTAINER;
2       for i ← 0 to numberOfTasks - 1
3           costContainer [scheduling[i]] + ← ETC[i][scheduling[i]];
End
```

Şekil 3.26. Maliyet fonksiyonunun sözde kodu

Maliyet fonksiyonunun doğru çalışmasının yanı sıra dikkat edilmesi gereken en önemli hususlardan birisi de hesaplama yapmak için harcadığı süredir. Bu sürenin oldukça ideal bir değerde olması gerekmektedir. Şekil 3.7'de 9 numaralı mavi kutu ile belirtilen aşamada programda yer alan tüm fonksiyonların arama boyunca harcadıkları süreleri değerlendirmek, uygulamanın çalışma zamanı profilini çıkarmak amacıyla Xcode programının *Time Profiler* aracından faydalanılmıştır. *Time Profiler*, programın performansını ölçmek için her bir fonksiyonun çalışma süresini analiz etmektedir.

Program Şekil 3.26'da gösterilen maliyet fonksiyonu ile çalıştırıldığı zaman Şekil 3.27'de yer alan zaman analizine ulaşılmıştır. Bu analizin değerlendirilmesi aşağıda yer almaktadır.

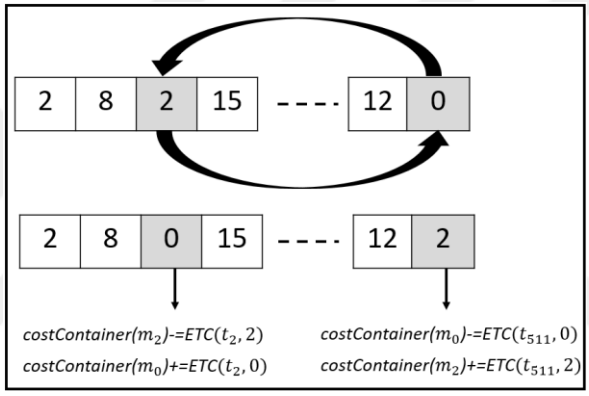
1. Şekil 3.28'da yer alan zaman profili programın T ile gösterilen 2 dakikalık sürede çalıştırılması sonucu elde edilmiştir.
2. Program seri olarak tek bir iş parçacığı (*thread*) ile çalıştırılmıştır.
3. T1 ile gösterilen zaman dilimi programda yer alan fonksiyonların toplam çalışma süresinin 1,92 dakika (1,92 *min*) olduğunu ifade etmektedir.
4. T2 ile gösterilen zaman dilimi maliyet fonksiyonunun programın %99,9'unda 1,92 dakikalık (1,92 *min*) bir çalışma süresi ile etkin olduğunu göstermektedir.
5. T3 ile gösterilen zaman dilimi rastgele sayı üretme fonksiyonunun çalışma süresinin 6,0 mili saniye (6,00 *ms*) olduğunu belirtmektedir.
6. T4 ile gösterilen zaman dilimi pertürbasyon fonksiyonunun çalışma süresinin 3,0 mili saniye (3.00 *ms*) olduğunu göstermektedir.
7. T1, T2, T3 ve T4 zaman dilimleri göz önüne alındığında programın T zaman diliminde çalışması sonucu en fazla çalışma süresine sahip olan fonksiyonun maliyet fonksiyonu olduğu anlaşılmaktadır.



Şekil 3.27. Optimizasyon öncesi zaman profili

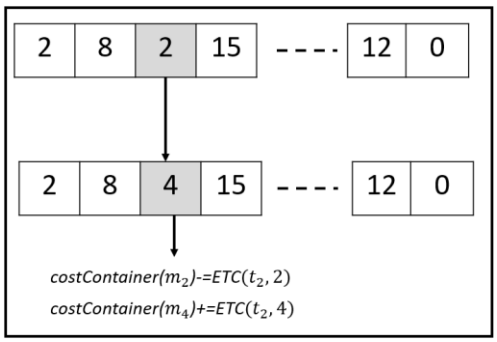
Şekil 3.27'de yer alan zaman profilinde ulaşılan sonuçlara bağlı olarak Şekil 3.7'de 10 numaralı mavi kutu ile belirtilen aşamada maliyet fonksiyonunda optimizasyon yapılmıştır. Bu optimizasyon sonrasında Şekil 3.26'da gösterilen maliyet fonksiyonu sadece başlangıç çözümünün maliyetini hesaplamak için kullanılmış ve her iterasyonda üretilen yeni çözümler için maliyet fonsiyonunun yeni çalışma şekli aşağıda verilmiştir.

1. Şekil 3.28'de görüldüğü gibi pertürbasyon fonksiyonunda takas yönteminin kullanılması durumunda sadece takas yapılan düğümlerin maliyetleri değişmektedir.



Şekil 3.28. Takas yöntemine bağlı maliyet hesaplama

2. Şekil 3.29'da görüldüğü gibi pertürbasyon fonksiyonunda yenileme yönteminin kullanılması durumunda ise sadece indisde mevcut olan düğümün ve yeni atanacak düğümün maliyeti değişmektedir.



Şekil 3.29. Yenileme yöntemine bağlı maliyet hesaplama

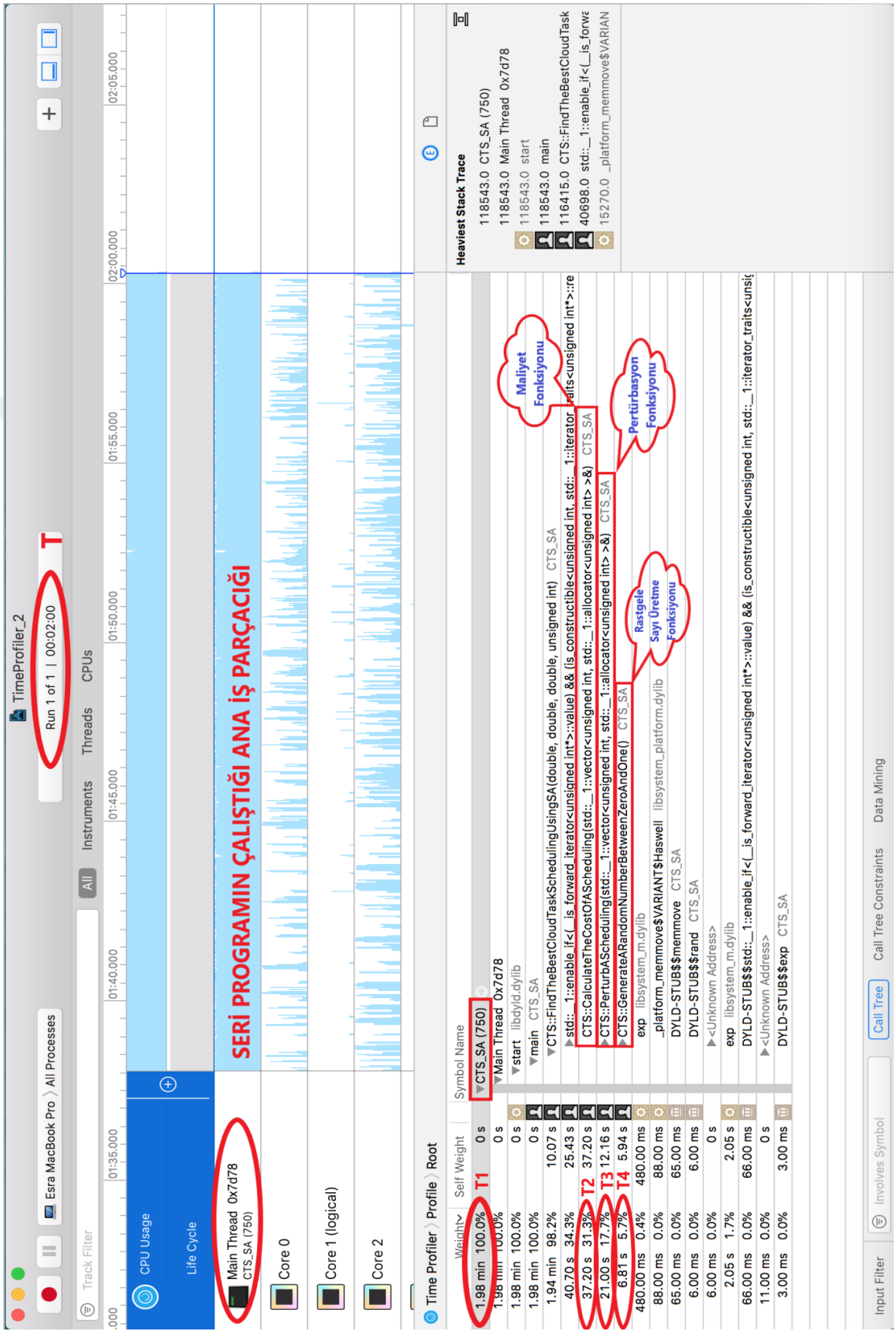
Maliyet fonksiyonunun pertürbasyon tekniğine bağlı çalışma şekline ait sözde kod Şekil 3.30'da verilmiştir.

```
Begin
1       if swap_method = true then
2            costContainer[scheduling[randomIndex1]] -= ETC[randomIndex1][scheduling[randomIndex1]];
3            costContainer[scheduling[randomIndex2]] -= ETC[randomIndex2][scheduling[randomIndex2]];
4            swap(scheduling[randomIndex1], scheduling[randomIndex2]);
5           costContainer [scheduling[randomIndex1]] += ETC[randomIndex1][scheduling[randomIndex1]];
6           costContainer [scheduling[randomIndex2]] += ETC[randomIndex2][scheduling[randomIndex2]];
7       else
8            costContainer[scheduling[randomIndex1]] -= ETC[randomIndex1][scheduling[randomIndex1]];
9            scheduling[randomIndex1] ← RANDOM(0, numberOfTasks-1);
10          costContainer[scheduling[randomIndex1]] += ETC[randomIndex1][scheduling[randomIndex1]];
End
```

Şekil 3.30. Pertürbasyon tekniğine bağlı maliyet fonksiyonunun sözde kodu

Program Şekil 3.30'da gösterilen maliyet fonksiyonu ile çalıştırıldığı zaman Şekil 3.31'de yer alan zaman analizine ulaşılmıştır. Bu analizin değerlendirilmesi aşağıda yapılmıştır.

1. Şekil 3.31'de yer alan zaman profili programın T ile gösterilen 2 dakika süresince çalıştırılması sonucu elde edilmiştir.
2. Program seri olarak tek bir iş parçacığı (*thread*) ile çalıştırılmıştır.
3. T1 ile gösterilen zaman dilimi programda yer alan fonksiyonların toplam çalışma süresinin 1,98 dakika (1,98 min) olduğunu ifade etmektedir.
4. T2 ile gösterilen zaman dilimi maliyet fonksiyonunun programın %31,3'ünde 37,2 saniyelik (37,2 sn) bir çalışma süresi ile etkin olduğunu göstermektedir.
5. T3 ile gösterilen zaman dilimi pertürbasyon fonksiyonunun programın %17,7'sinde 21 saniyelik (21 sn) bir çalışma süresi ile etkin olduğunu göstermektedir.
6. T4 ile gösterilen zaman dilimi rastgele sayı üretme fonksiyonunun programın %5,7'sinde 6.8 saniyelik (6.8 sn) bir çalışma süresi ile etkin olduğunu göstermektedir.
7. T1, T2, T3 ve T4 zaman dilimleri göz önüne alındığında maliyet fonksiyonunda pertürbasyon yönteminin kullanımına bağlı olarak yapılan optimizasyon sonucu fonksiyonlar arası çalışma zamanı dağılımının dengelendiği gözlenmiştir.



Şekil 3.31. Optimizasyon sonrası zaman profili

3.6.9. Kabul kriteri (Acceptance criterion)

Benzetilmiş tavlama yaklaşımında sonraki çözümün kabul edilmesi mevcut çözüm ile sonraki çözümün maliyetleri kullanılarak hesaplanan ΔC değerine bağlı olarak belirlenmektedir. Kabul kriteri üç farklı aşamadan oluşmaktadır.

1. İlk aşamada yeni çözüm ile mevcut çözümün maliyetleri arasındaki fark hesaplanmaktadır.

$$\Delta C = C_{nextScheduling} - C_{currentScheduling}$$

2. İkinci aşamada bu farkın 0'dan küçük olma durumuna bakılmakta ve 0'dan küçükse yeni çözüm ve bu çözüme ait maliyet değeri koşulsuz kabul edilmektedir. Yeni çözümün bu şekilde kabul edilmesi benzetilmiş tavlamada yüksek tırmanış (*high climbing*) olarak adlandırılmaktadır.

$$\Delta C < 0$$
$$currentScheduling \leftarrow nextScheduling$$
$$C_{currentScheduling} \leftarrow C_{nextScheduling}$$

3. Üçüncü aşamada ise benzetilmiş tavlamanın en temel özelliklerinden biri olan minimum değere yaklaştıracak çözüm aranırken maliyet fonksiyonundaki artışa sebep olan yeni çözümlerin de kabul edilmesidir. Bu durumda yeni çözümün kabulü için Metropolis kriterine bağlı olarak üstel dağılım (*Boltzmann*) rastgele üretilen değerle kıyaslanmakta ve *P*'nin rastgele değerden büyük olması halinde yeni çözüm olasılıksal olarak kabul edilmektedir.

$$P \leftarrow e^{\frac{-\Delta}{T}}$$
$$\Delta C \geq 0 \;\&\&\; real\_random\,(0,1) < P$$
$$currentScheduling \leftarrow nextScheduling$$
$$C_{currentScheduling} \leftarrow C_{nextScheduling}$$

Kabul kriterinin çalışma şekline ait sözde kod Şekil 3.32'de verilmiştir.

```
Begin
1       currentSchedulingCost ← COST(currentScheduling);
2       nextScheduling ← PERTURB(currentScheduling)
3       nextSchedulingCost ← COST(nextScheduling);
4       ΔC ← nextSchedulingCost - currentSchedulingCost;
5       if ΔC < 0 then
6             currentScheduling ← nextScheduling;
7             currentSchedulingCost ← nextSchedulingCost;
8       else if REAL_RANDOM(0, 1) < e^-(ΔC/T) then
9             currentScheduling ← nextScheduling;
10            currentSchedulingCost ← nextSchedulingCost;
End
```

Şekil 3.32. Kabul kriterinin sözde kodu

3.6.10. Erken sonlandırma (Early termination)

Benzetilmiş tavlama yaklaşımında sıcaklık başlangıçta belirlenen iterasyon sayısı maksimum değere ulaştığı zaman soğutma oranı kullanılarak azaltılmaktadır. Ancak programın çalışma süresinin uzadığı durumlarda Şekil 3.33'de görüldüğü gibi iki farklı erken sonlandırma koşulu oluşturularak (termal denge) çalışma süresi ideal bir değere getirilmiştir. Aşağıda erken sonlandırma koşullarının çalışma şekli yer almaktadır.

1. Şekil 3.7'de 11 numaralı mavi kutu ile belirtilen aşamada iç döngüye bağlı sonlandırma işlemi gerçekleştirilmektedir. Her bir sıcaklıkta başlangıçta belirlenen iterasyonun sonuna kadar gitmek yerine, iç döngü erken sonlandırma koşulu sağlanınca yeni sıcaklığa geçiş yapılmaktadır. İç döngü erken sonlandırma koşulu Şekil 3.7'de 11 numaralı mavi kutu ile belirtilen aşamada kırmızı kutu ile gösterilmekte ve herhangi bir sıcaklıkta ΔC'nin sıfırdan küçük olduğu çözümlerin sayısının kullanıcı tarafından belirlenen bir eşik değerini (*inner_loop_threshold*) geçmesi durumunda kullanılmaktadır.
2. Şekil 3.7'de 12 numaralı mavi kutu ile belirtilen aşamada ise dış döngüye bağlı erken sonlandırma işlemi gerçekleştirilmektedir. Dış döngü erken sonlandırma Şekil 3.7'de 12 numaralı mavi kutu ile belirtilen aşamada kırmızı kutu ile gösterilen N/4 iterasyon sonrasında mevcut çözümün değişmediği durumlarda kullanılmaktadır.

```
Begin
1       T ← INIT-TEMP;
2       ε ← 0.01;
3       while T > ε do
4           while inner_loop_criteria ← FALSE do
5               if ΔC < 0 then
6                   if inner_loop_threshold = true then
7                       break;
                    .
                    .
                    .
n-4         end while
n-3         if outher_loop_threshold = true then
n-2             break;
n-1         T ← SCHEDULE(T);
n       end while
End
```

Şekil 3.33. Erken sonlandırmanın sözde kodu

3.6.11. Derleyici etkisi

Benzetilmiş tavlama ve C++ dili kullanılarak hazırlanan programın Şekil 3.7'de 13 numaralı mavi kutu ile belirtilen aşamada çözüme ulaşma süresini büyük oranda etkileyen faktörlerden birisinin de derleyici etkisi olduğu anlaşılmıştır. Program aşağıda özellikleri belirtilen g++, LLVM ve MSVC++ derleyicileri ile çalıştırılarak test edilmiş ve Şekil 3.7'de 13 numaralı mavi kutu ile belirtilen aşamada kırmızı kutu ile gösterilen g++ derleyicisinin kullanımına karar verilmiştir.

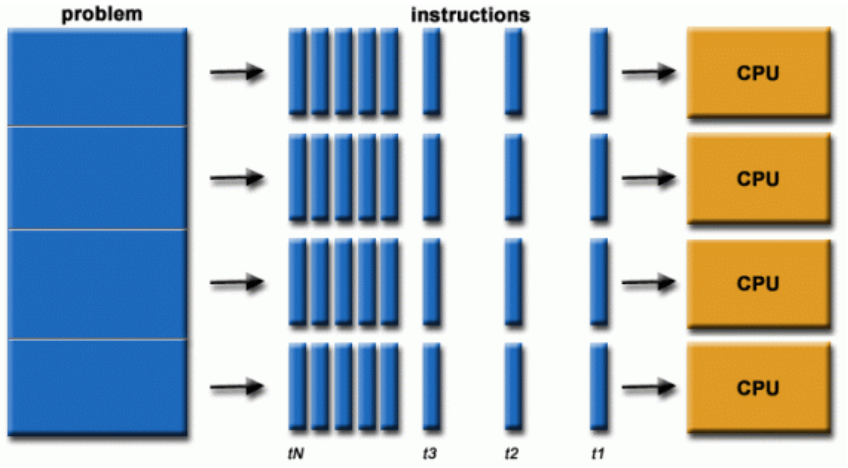
- **g++:** Ubuntu işletim sisteminde komut satırının kullanıldığı GCC'ye ait varsayılan C++ derleyicisidir ve *ccache* önbelleğini kullanmaktadır. Bu önbellek bir programın derlenmesi sonucu tümüyle aynı sonucu verecek kodların algılanmasını kriptografik karma algoritmasını kullanarak sağlamakta ve bunların tekrar derlenmesi yerine önceki sonuçları kullanarak derleme işlemini hızlandırmaktadır.
- **LLVM:** Xcode programının kullandığı varsayılan C++ derleyicisidir. Bu derleyicide g++ gibi *ccache* derleyici önbelleğini kullanmaktadır.

- **MSVC++:** Microsoft Visual Studio programının kullandığı varsayılan C++ derleyicisidir ve *clcache* önbelleğini kullanmaktadır. Bu derleyici önbelleği *ccache* önbelleğinden esinlenerek geliştirilmiştir. Ancak derleme işleminde *ccache* önbelleği kadar hızlı değildir.

3.7. Paralel Veri İşleme

Bir problemin çözümünü daha hızlı elde etmek için parçalara ayrılmış problemin çok sayıda işlemcide eşzamanlı olarak işletilmesi gerekmektedir. Birden fazla işlemci kullanarak farklı sayıdaki program parçalarını her parçayı kendi içerisinde farklı komut dizileri halinde çalıştırma işlemine paralel veri işleme denilmektedir.

Başlangıçta problem bir bütün halinden parçalara ayrılmaktadır. Her parça komutlar halinde işlemcilere dağıtılmaktadır. Böylelikle parçalar aynı anda farklı işlemciler üzerinde işlenmektedir. Şekil 3.34'de görüldüğü gibi her bir komut dizisi atandığı işlemcide çalışmaktadır. Paralel veri işleme büyük boyutlu problemlerde zaman açısından büyük kazanç sağlamaktadır.



Şekil 3.34. Paralel veri işleme (Anonim 2013)

3.8. Paralel Programlama Modeli

Paralel programlama modeli temelde uygulamalar, diller, derleyiciler, kütüphaneler, işletim sistemleri ve paralel giriş/çıkış alanlarını kapsamaktadır. Programcılar, kendileri ve uygulamaları için uygun bir model veya karma bir model seçerek, uygulamalarını geliştirmektedir. Sık kullanılan paralel programlama modellerinden birisi OpenMP'dir.

3.8.1. OpenMP Nedir

OpenMP birçok işletim sistemi (MAC OS X, Windows, Linux, HP-UX) üzerinde belirli programlama dillerini (C, C++, Fortran) kullanarak bellek paylaşımı yoluyla çoklu işlemeye olanak sağlayan uygulama geliştirme arayüzüdür. OpenMP bir görevi işlerken ana iş parçacığı ve yardımcı iş parçacıklarını kullanmaktadır. Ana iş parçacığı yardımcı iş parçacıklarına hem görev paylaşımı yapmakta hem de gerekli durumlarda yardımcı iş parçacıklarını durdurup yeni bir görev paylaşımı yapabilmektedir.

3.8.2. OpenMP ile Derleme

Visual Studio 2017 ortamında MSVC++ derleyicisini kullanarak OpenMP ile hazırlanan bir paralel programı derlemek için OpenMP kütüphanesi desteğine ihtiyaç duyulmaktadır. Bunun için;

- *Project Properties >> Configuration Properties >> C/C++ >> Language >> Open MP Support >> Yes (/openmp)* şeklinde bir ayarlama yapılmaktadır.

Visual Studio 2017'de OpenMP ile hazırlanan programı performans optimizasyonu bayrağı ile derlemek için ilk olarak;

- *Project Properties >> Configuration Properties >> C/C++ >> Optimization >> Optimization >> Maximum Optimization (Favor Speed) (/O2)* şeklinde bir ayarlama yapılmaktadır.

Daha sonra;

- *Project Properties >> Configuration Properties >> C/C++ >> Code Generation >> Basic Runtime Checks >> Default* şeklinde bir başka ayarlama yapılmaktadır.

Ayrıca g++ ve LLVM derleyicilerini kullanarak komut satırları üzerinden OpenMP ile hazırlanan programı derlemek için *fopenmp* bayrağı kullanılmaktadır.

3.9. Paralel Benzetilmiş Tavlama

Bir programın paralel olarak çalıştırılmasının amacı sistemde yer alan işlemcileri kullanarak daha verimli çözümleri daha ideal bir sürede elde etmektir. Bu tez kapsamında benzetilmiş tavlama yaklaşımını paralel olarak çalıştırmak için eş zamansız (*asenkron*) paralel uygulama, başka bir deyişle çoklu çalıştırma (*multi-run*) yönteminden faydalanılmıştır. Multi-Run, SA yaklaşımını paralelleştimek için kullanılan tekniğin adıdır ve SA yaklaşımının birbirinden bağımsız çoklu işlemciler ile (işlemciler arası bilgi alışverişi olmadan) aynı anda çalıştırılmasını sağlamaktadır. Böylelikle arama uzayını genişleterek aynı zaman diliminde birden fazla sonuca ulaşılması sağlanmaktadır.

Eş zamansız (*asenkron*) paralel benzetilmiş tavlama yaklaşımında Şekil 3.35'de görüldüğü gibi başlangıçta çalışacak iş parçacıklarının (*threads*) sayısı belirlenmektedir. Her iş parçacığının kendi etiketi (*threadID*) ile ulaşabileceği en iyi maliyetlerin tutulduğu tek boyutlu bir dinamik dizi (*bestSchedulingCost*) ve başlangıç sıcaklığı ortak paylaşım (*shared*) alanında oluşturulmaktadır. Daha sonra birbirlerinden bağımsız her bir iş parçacığı tarafından seri SA yaklaşımı koşturulmaktadır. Tüm iş parçacıklarının çalışmaları tamamlandığında ise her birinin ulaştığı en iyi sonuçları saklayan tek boyutlu dinamik dizi içerisindeki en iyi maliyet değeri belirlenmektedir.

```
Begin
1       numberOfThreads ← INIT-THREADS;
2       T ← INIT-TEMP;
3       bestScheduling ← INIT-SCHEDULING;
4       bestSchedulingCost ← INIT-COST;
5       #pragma omp parallel num_threads(numberOfThreads)
6               threadID ← omp_get_thread_num();
7               privateTemperature ← T;
8               scheduling ← INIT-SCHEDULING;
9               RANDOM(scheduling);
10              schedulingCost ← COST(scheduling);
11              bestSchedulingCost[threadID] ← schedulingCost;
12              while privateTemperature > ε do
13                      while inner_loop_criteria ← FALSE do
14                              newScheduling ← PERTURB(scheduling);
15                              newSchedulingCost ← COST(newScheduling);
16                              ΔC ← newSchedulingCost - schedulingCost;
17                              if ΔC < 0 then
18                                      scheduling ← newScheduling;
19                                      schedulingCost ← newSchedulingCost;
20                                      if newSchedulingCost < bestSchedulingCost[threadID] then
21                                              bestSchedulingCost[threadID] ← newSchedulingCost;
22                                              bestScheduling[threadID] ← newScheduling;
23                              else if RANDOM(0,1) < e^-(ΔC/T) then
24                                      scheduling ← newScheduling;
25                                      schedulingCost ← newSchedulingCost;
26                              privateTemperature ← SCHEDULE(T);
27                      end while
28              end while
29      end pragma
30      bestCost ← bestSchedulingCost[0];
31      for i ← 1 to numberOfThreads
32              bestCost ← bestSchedulingCost[i];
End
```

Şekil 3.35. Paralel benzetilmiş tavlama yaklaşımının sözde kodu

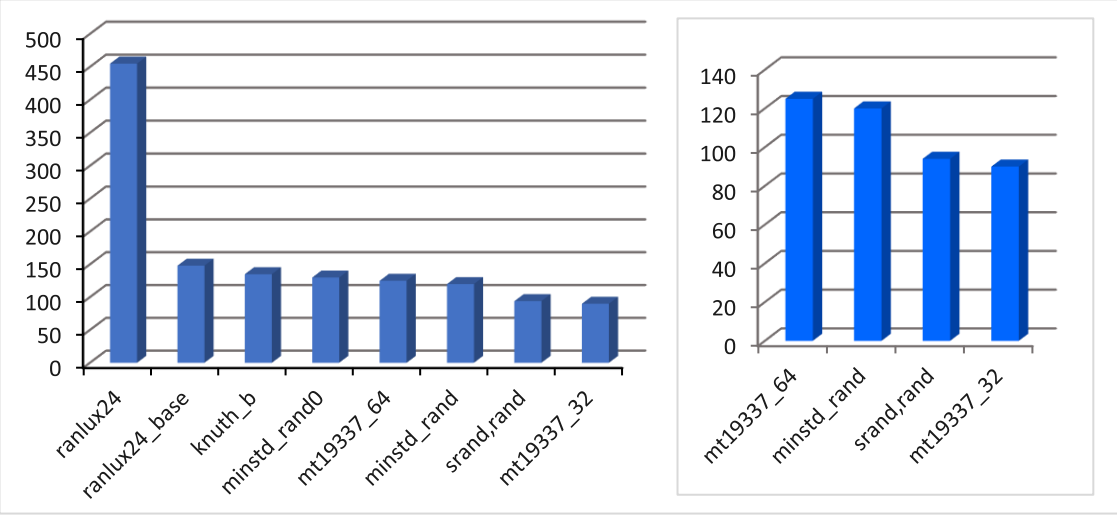
4. ARAŞTIRMA BULGULARI

Tezin bu bölümünde geliştirilen bilgisayar programının çalışma süresinin azaltılması için, benzetilmiş tavlamanın ihtiyaç duyduğu farklı rastgele sayı üretme ve pertürbasyon tekniklerine, veri yapılarına ve ver tiplerine, döngü sonlandırma koşullarına, keşif-sömürme oranlarına ve derleyici etkisine bağlı optimizasyon bulguları incelenecektir.

4.1. Rastgele Sayı Üretme Tekniklerine Bağlı Optimizasyon

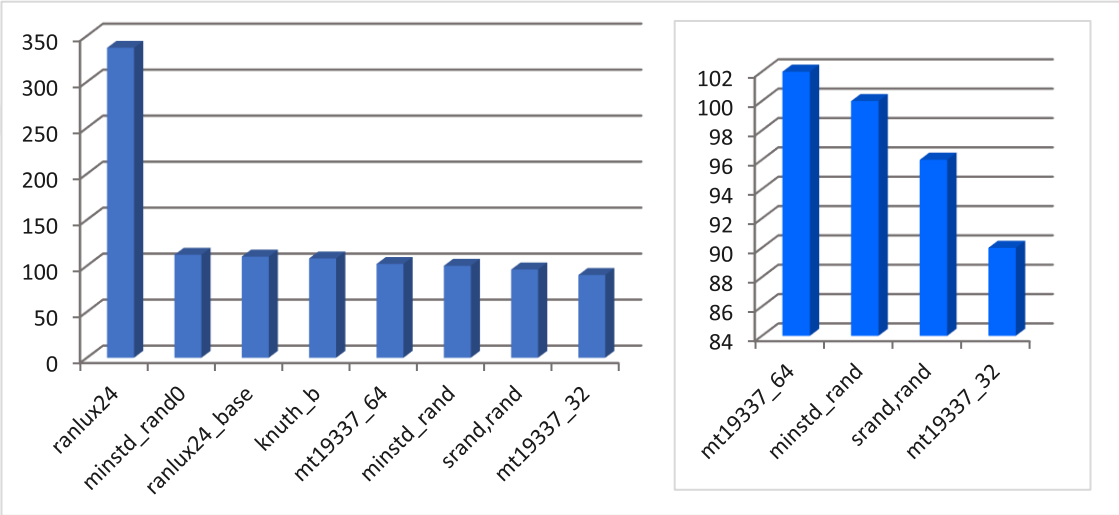
Benzetilmiş tavlamada başlangıç çözümüne değer atanmasından, pertürbasyon yöntemine kadar programın büyük çoğunluğunda rastgele üretilen sayılar kullanılmaktadır. Programda rastgele sayılar üretmek için sekiz farklı rastgele sayı üreteci Braun modelinde yer alan her bir veri seti ile çalıştırılarak test edilmiş ve grafiklerle detaylandırılmıştır.

Veri setindeki "u_c_hihi" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılmasına ait grafik Şekil 4.1'de yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin, belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %80.2, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %4.2 oranında kısa sürede çözüme ulaştığı görülmektedir.



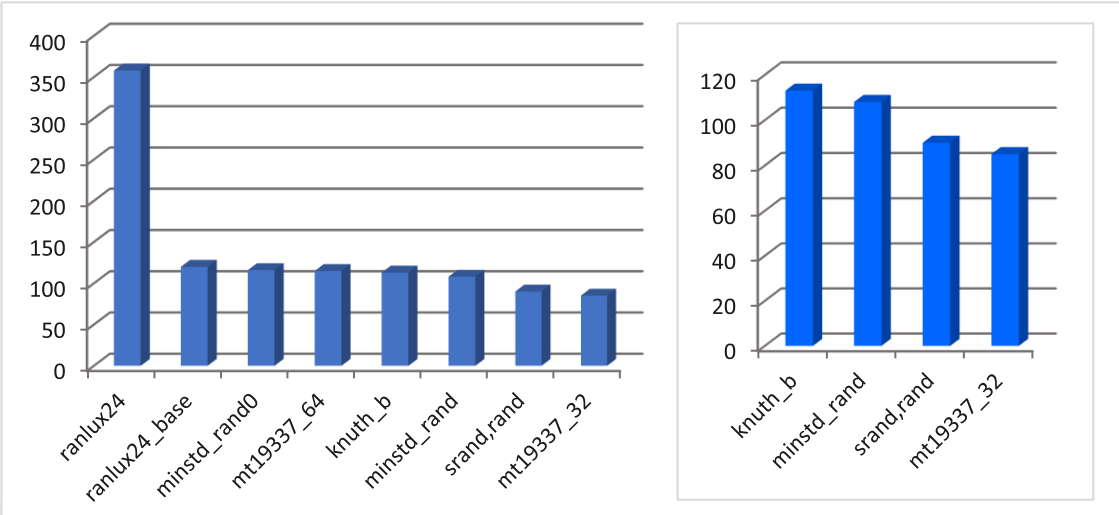
Şekil 4.1. u_c_hihi için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_c_hilo" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.2'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %73.2, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %6.2 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



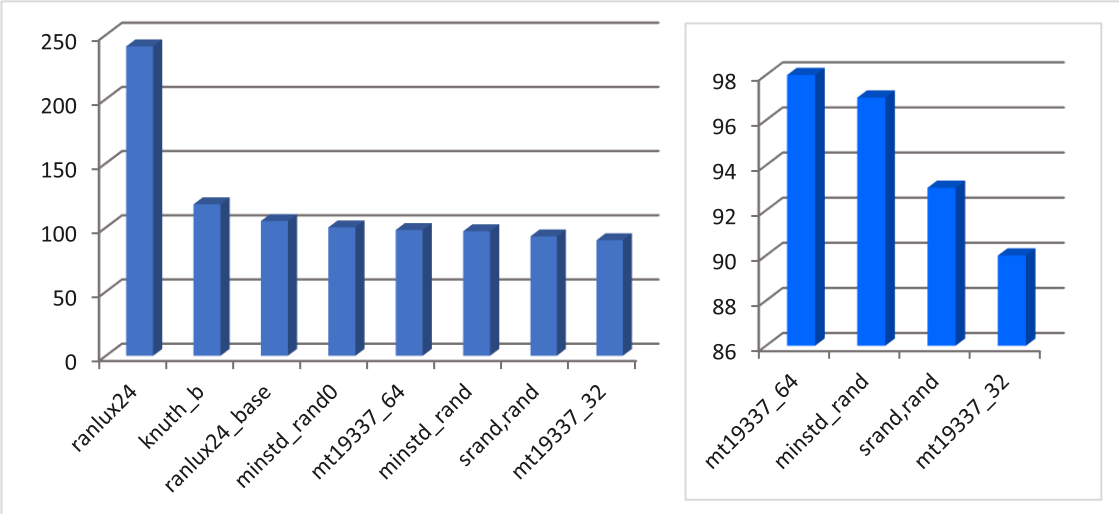
Şekil 4.2. u_c_hilo için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_c_lohi" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.3'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %76.2, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %5.5 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



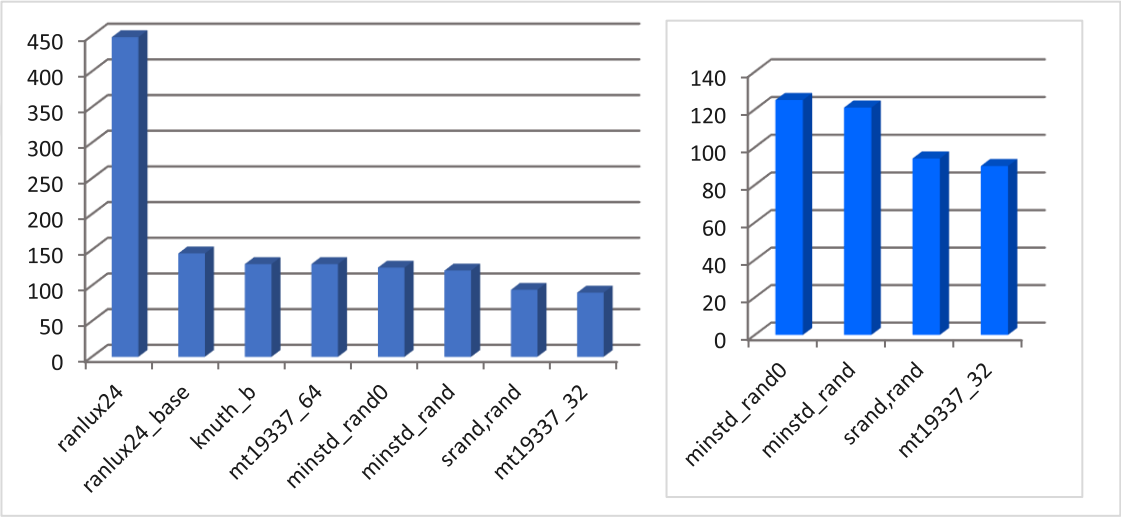
Şekil 4.3. u_c_lohi için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_c_lolo" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.4'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %62.6, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %3.2 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



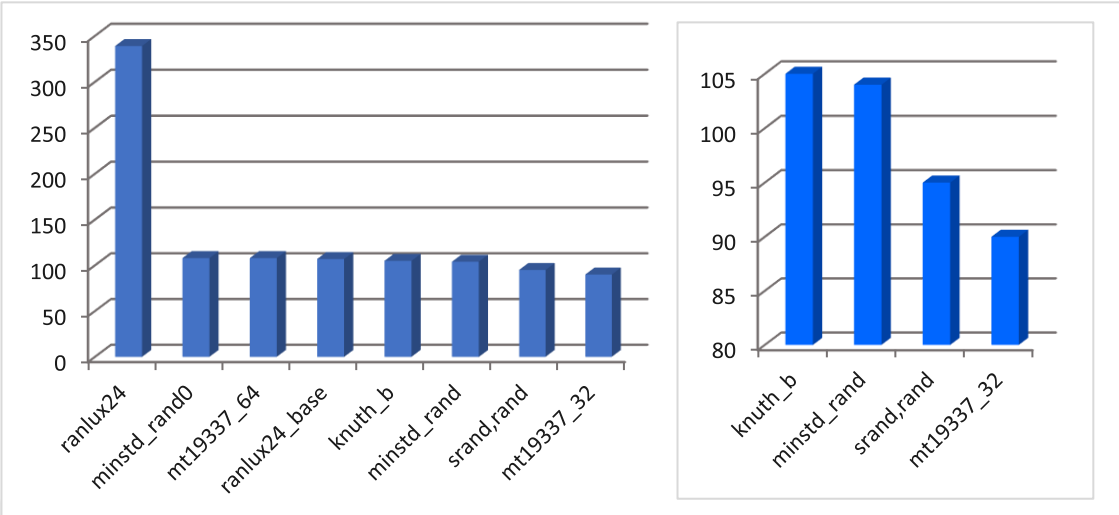
Şekil 4.4. u_c_lolo için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_i_hihi" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.5'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %79.9, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %4.2 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



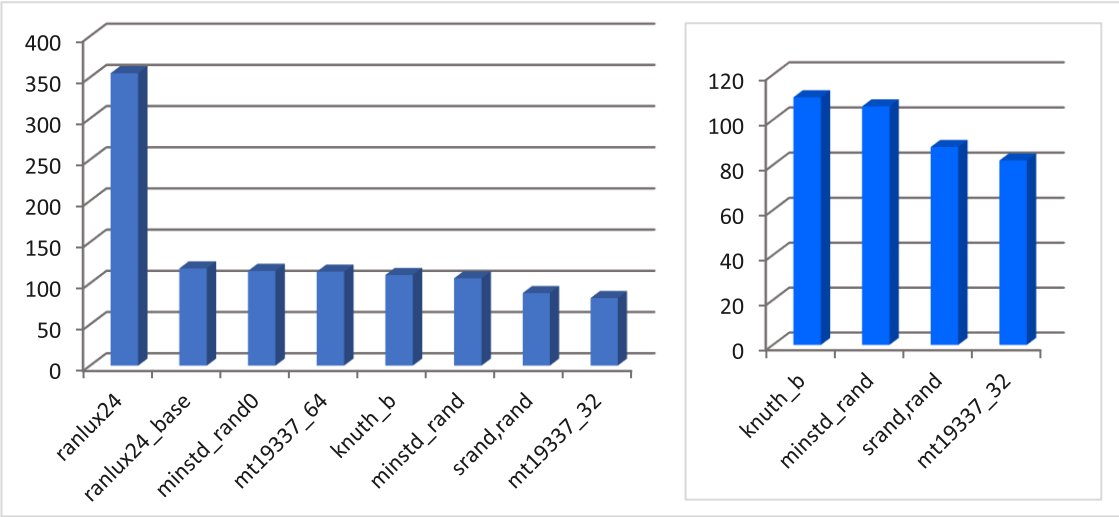
Şekil 4.5. u_i_hihi için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_i_hilo" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.6'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %73.4, kendisine en yakın çalışma süresine sahip *srand,rand* üretecine göre de %5.2 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



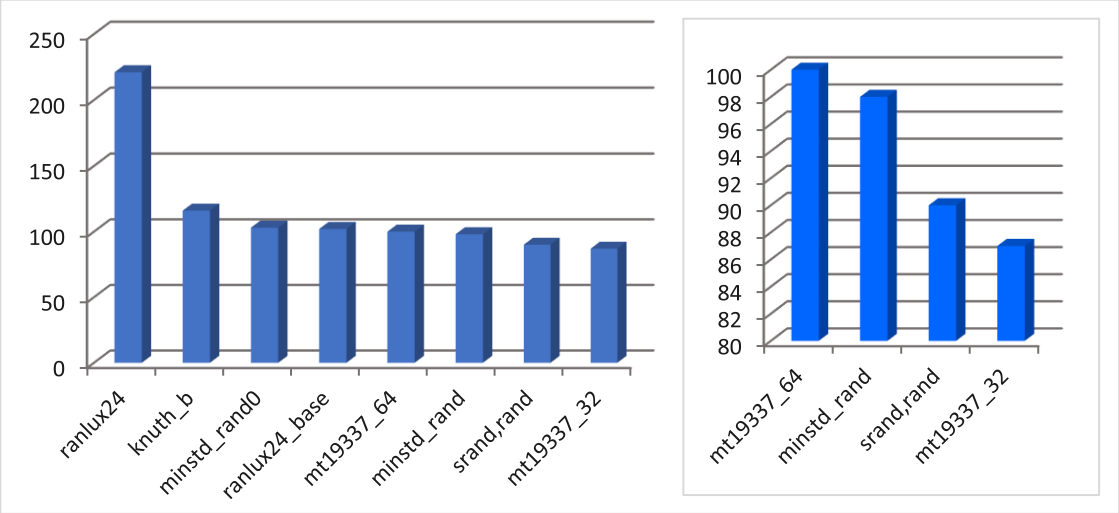
Şekil 4.6. u_i_hilo için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_i_lohi" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.7'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %76.9, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %6.8 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



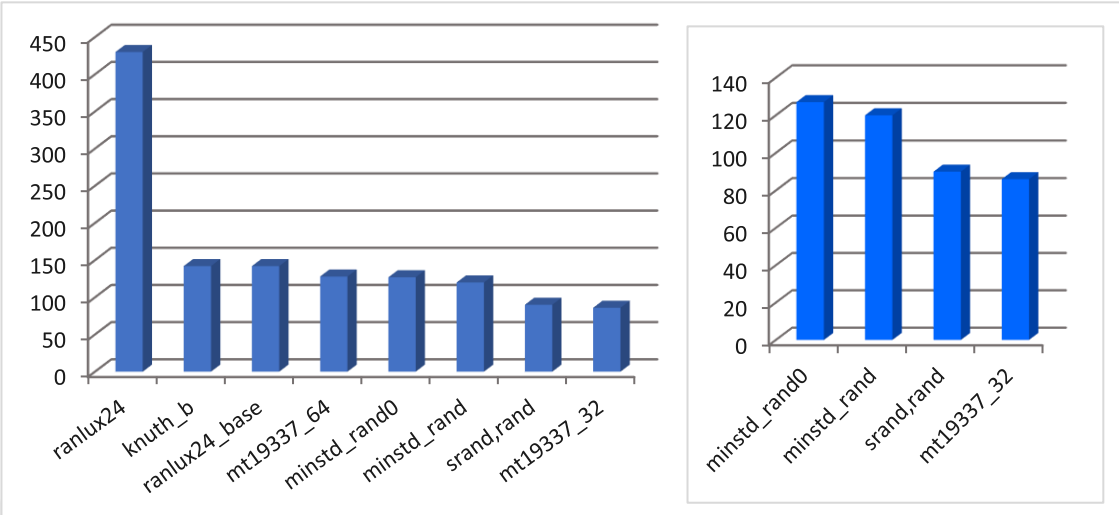
Şekil 4.7. u_i_lohi için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_i_lolo" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.8'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %60.6, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %3.3 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



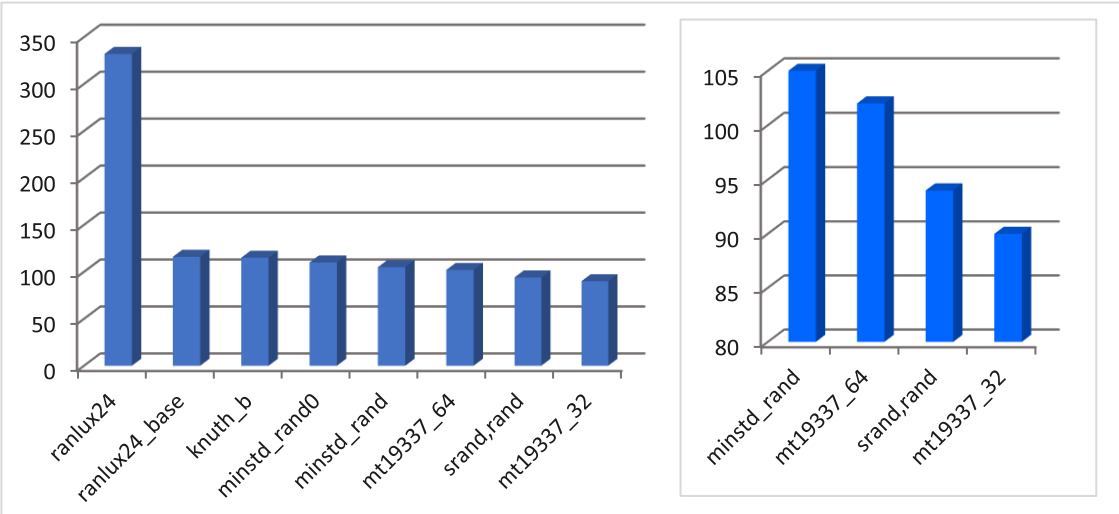
Şekil 4.8. u_i_lolo için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_s_hihi" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.9'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %80, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %4.4 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



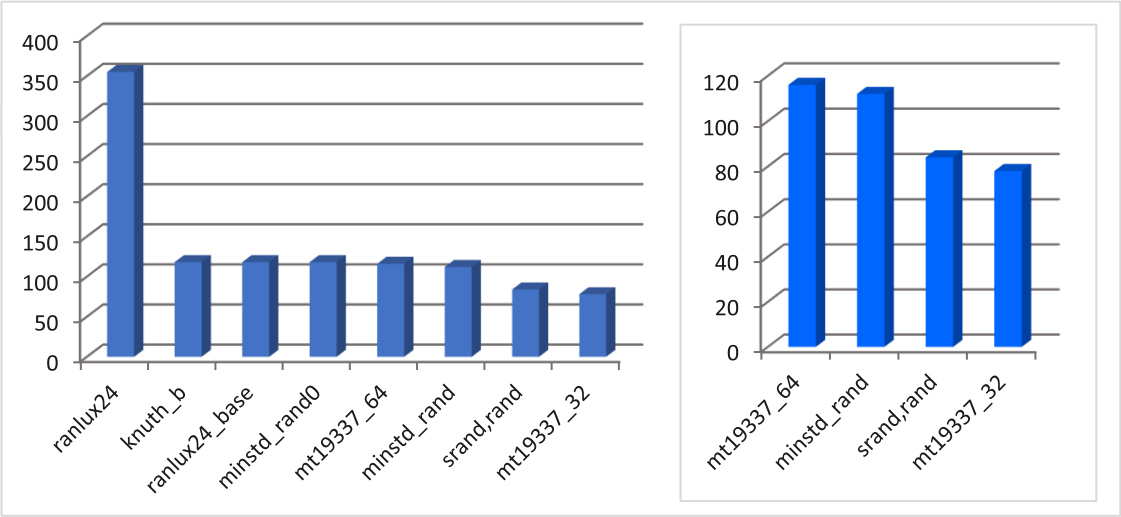
Şekil 4.9. u_s_hihi için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_s_hilo" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.10'da gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %72.8, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %4.2 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



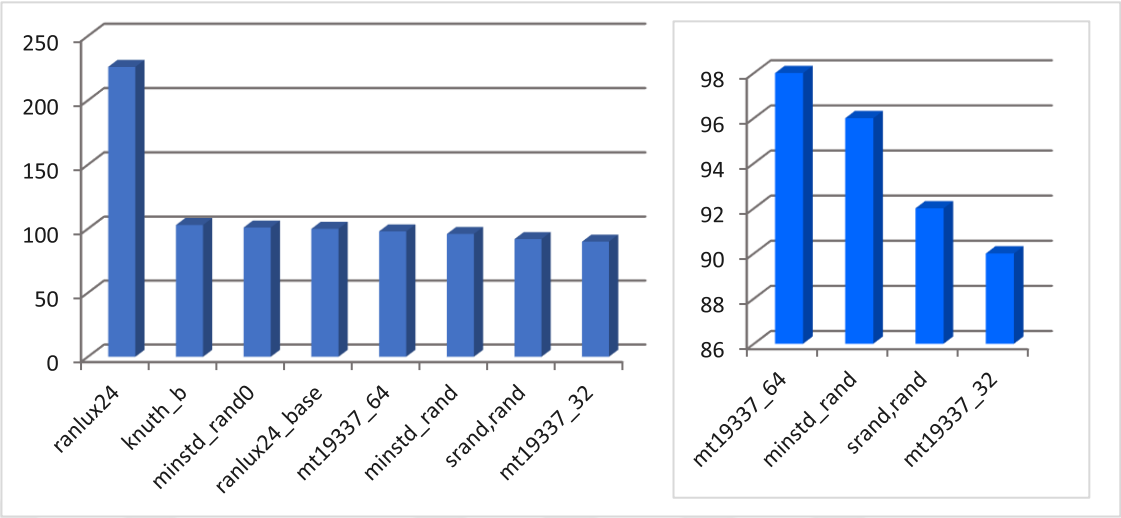
Şekil 4.10. u_s_hilo için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_s_lohi" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.11'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %78, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %7.1 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



Şekil 4.11. u_s_lohi için farklı rastgele sayı üreteçlerinin çalışma süreleri

Veri setindeki "u_s_lolo" isimli benchmarkın tüm rastgele sayı üreteçleri ile çalıştırılma süresi Şekil 4.12'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde *mt19337_32* üretecinin belirtilen veri seti için en yüksek çalışma süresine sahip *ranlux24* üretecine göre %60.1, kendisine en yakın çalışma süresine sahip *srand, rand* üretecine göre de %2.1 oranında daha iyi olduğu ve tüm rastgele sayı üreteçleri arasında en iyi çalışma süresine ulaştığı görülmektedir.



Şekil 4.12. u_s_lolo için farklı rastgele sayı üreteçlerinin çalışma süreleri

Grafikler program için en iyi çalışma sürelerine *rand* ve *mt19337_32* üreteçleri ile ulaşıldığını göstermektedir. Bu iki üretecin çalışma süreleri birbirine oldukça yakındır. Ancak *rand* üretecinin kullandığı algoritma ve tohum her defasında yeterince rastgelelik sağlayamadığı için farklı sayılar üretme kalitesi düşmektedir. Üretilen ilk rastgele sayı benzer tohum değerleri için fazla değişmemektedir. Bu nedenle programda kullanıldığı zaman düşük kaliteli çözümler (en iyi gecikme değerinden oldukça uzak sonuçlar) oluşmaktadır. Öte yandan *mt19337_32* üretecinin kullandığı algoritma ve tohum yeteri kadar rasgelelik sağladığı için minimum çalışma süresinde yüksek kaliteli çözümler (en iyi gecikme değerine oldukça yakın veya daha iyi) oluşmaktadır. Geliştirilen bilgisayar programında rastgele sayılar üretmek için *mt19337_32 (default_random_engine)* üreteci kullanılmıştır.

4.2. Pertürbasyon Tekniklerine Bağlı Optimizasyon

Benzetilmiş tavlamada mevcut çözümde küçük ve anlamlı değişiklikler yaparak yeni çözümler üretmek için kullanılabilir pertürbasyon teknikleri arasından en iyi olanını belirlemek amacıyla aşağıda yer alan iki farklı optimizasyon işleminden faydalanılmıştır.

1. İlk optimizasyon için program altı farklı pertürbasyon tekniği ile çalıştırılarak Çizelge 4.1'de yer alan sonuçlara ulaşılmıştır. Bu sonuçlardan yola çıkılarak en iyi pertürbasyon tekniklerinin takas ve yenileme olduğu belirlenmiştir.

Çizelge 4.1. Farklı pertürbasyon tekniklerine bağlı en iyi gecikme değerleri

BENCHMARK	Takas	Yenileme	Ters Çevirme	Araya Ekleme	Karıştırıcı	Kaydırma
u_c_hihi.0	9196900.0	7984322.4	19465230.1	20693512.1	18684789.3	22791432.4
u_c_hilo.0	164419.2	165699.3	198043.2	218758.2	225647.5	248240.4
u_c_lohi.0	328416.4	256501.1	590200.2	645753.3	748744.3	864549.3
u_c_lolo.0	5610.0	5521.1	6580.3	7440.2	7804.0	7951.2
u_i_hihi.0	2987125.2	2984011.4	7052734.5	10143321.2	10435246.2	19740245.3
u_i_hilo.0	74448.3	75376.2	100350.3	122917.1	108105.3	176150.4
u_i_lohi.0	103854.0	104470.4	210128.4	270311.0	238127.4	458157.5
u_i_lolo.0	2600.5	2630.5	3960.5	4302.5	4508.5	5820.5
u_s_hihi.0	5056322.4	4296461.1	10634654.3	12004734.2	10754893.5	18487342.1
u_s_hilo.0	99788.1	102101.2	121848.6	147834.3	162305.2	180058.2
u_s_lohi.0	174153.1	127150.0	338101.1	380418.4	455786.3	566335.5
u_s_lolo.0	3592.1	3643.5	4585.2	5462.3	5533.4	6052.2

2. İkinci optimizasyonda program sömürü/keşif oranını belirlemek için dokuz farklı ondalıklı değer ile çalıştırılarak Çizelge 4.2'de yer alan sonuçlara ulaşılmıştır. Bu sonuçlardan yola çıkılarak programda sömürü/keşif oranı için 0.9 değerinin kullanılmasına karar verilmiştir.

Pertürbasyon tekniklerine ve sömürü/keşif oranına bağlı olarak uygulanan iki optimizasyon sonrası en iyi sonuçlara %90 oranında takas, %10 oranında yenileme tekniğinin kullanılması ile ulaşıldığı anlaşılmıştır.

Çizelge 4.2. Farklı sömürü/keşif oranı için en iyi gecikme değerleri

BENCHMARK	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
u_c_hihi.0	7405795.1	7405182.1	7404336.2	7403655.8	7403662.6	7402252.4	7401126.1	7400084.0	*7377378.2*
u_c_hilo.0	154048.3	153824.1	153712.5	153748.3	153662.2	153504.0	153448.2	153302.1	*153088.7*
u_c_lohi.0	242610.1	241924.1	241611.0	240850.4	241100.0	240312.5	240004.1	239720.5	*239069.9*
u_c_lolo.0	5160.1	5165.3	5159.4	5157.4	5158.4	5155.4	5153.4	5152.4	*5145.5*
u_i_hihi.0	2959348.6	2957510.6	29535012.5	2944624.0	2943231.0	2942001.0	2940282.1	2939901.0	*2931630.9*
u_i_hilo.0	74110.3	74065.4	73884.2	73768.2	73660.5	73665.9	73518.6	73440.0	*73294.3*
u_i_lohi.0	103578.2	103224.0	103036.9	102882.9	102597.3	102310.5	102196.4	102090.1	*101737.7*
u_i_lolo.0	2560.5	2556.5	2558.2	2555.8	2552.3	2550.2	2548.8	2545.5	*2539.1*
u_s_hihi.0	4141244.6	4130108.5	4125432.0	4118850.5	4115002.0	4110050.2	4108994.4	4105250.5	*4098580.9*
u_s_hilo.0	96512.9	96442.5	96300.0	96254.8	96212.5	96125.0	96090.8	96034.0	*95764.9*
u_s_lohi.0	123365.4	123089.0	123076.7	122500.5	122646.8	122208.0	121120.0	122085.5	*121484.8*
u_s_hihi.0	3456.5	3450.2	3448.7	3446.2	3443.5	3440.2	3439.3	3435.2	*3425.9*

4.3. Veri Yapısına ve Veri Tipine Bağlı Optimizasyon

Geliştirilen bilgisayar programında veri yapısı, veri tipi ve hazır fonksiyon kullanımına bağlı olarak üç farklı optimizasyon gerçekleştirilmiştir.

1. Programda çözümü temsil etmek için tek boyutlu (görev odaklı) bir temsil kullanılmış ve başlangıçta veri yapısı olarak STL vektörden faydalanılmıştır. Daha sonra veri yapısı, programın çalışma süresini iyileştirmek amacıyla STL vektörden farklı olarak değişken adreslerini tutan ve bu nedenle belleğe doğrudan erişim sağlayan tek boyutlu bir dinamik diziye dönüştürülmüştür. Program her iki veri yapısı ile ayrı ayrı çalıştırıldığında tek boyutlu dinamik dizi yapısının STL vektör kullanımına göre çalışma süresinde oldukça fazla bir iyileşme sağladığı görülmüştür. Bu sebeple çözümü temsil etmek için tek boyutlu dinamik dizi yapısı tercih edilmiştir.

2. Çözümü temsil eden tek boyutlu dinamik dizinin her bir elemanı için başlangıçta bellekte 4 byte yer kaplayan bir veri tipi kullanılmıştır. Oysa [0,15] aralığındaki dizi elemanları için 1 byte yeterli olmaktadır. Bu nedenle programda veri tipi olarak bellekte 2 byte yer kaplayan *unsigned short* veri tipi kullanılmıştır.

3. Programda küçük değişiklikler yaparak yeni çözümler oluşturmak için kullanılan pertürbasyon teknikleri başlangıçta C++'ın *algorithm* kütüphanesinden temin edilmiştir. Ancak bu tekniklerde kullanılan operatörlerin (*swap*) çağrılması sırasında ortaya çıkan ek maliyeti önlemek için hazır tekniğin yaptığı işlem doğrudan elle yazılmıştır. Ayrıca programda çeşitli işlemleri yapmak için kullanılan çarpma ve bölme operatörlerinin getirdiği ek maliyeti önlemek amacıyla da çarpma işlemi yerine bit düzeyinde sola kaydırma, bölme işlemi yerine ise bit düzeyinde sağa kaydırma işleminden faydalanılmıştır.

Bu üç farklı optimizasyon sonucu programın çalışma süresinde ortalama 20 saniyelik bir iyileşme gözlemlenmiştir.

4.4. Maliyet Fonksiyonuna Bağlı Optimizasyon

Benzetilmiş tavlama yaklaşımında kullanılan fonksiyonların her birinin programın çalışma süresini etkileme oranını belirlemek için programın zaman profili oluşturulmuş ve en yüksek oranın maliyet fonksiyonuna ait olduğu tespit edilmiştir. Maliyet fonksiyonunda programda kullanılan pertürbasyon yöntemi dikkate alınarak optimizasyon gerçekleştirilmiştir. Bu optimizasyon neticesinde maliyet fonksiyonunun programın çalışma süresindeki % 99.9 etki oranı %31.3'e düşürülerek programın çalışma süresinde büyük oranda iyileşme sağlanmıştır. (Bkz. Şekil 3.27 ve Şekil 3.31)

4.5. Erken Sonlandırma Koşuluna Bağlı Optimizasyon

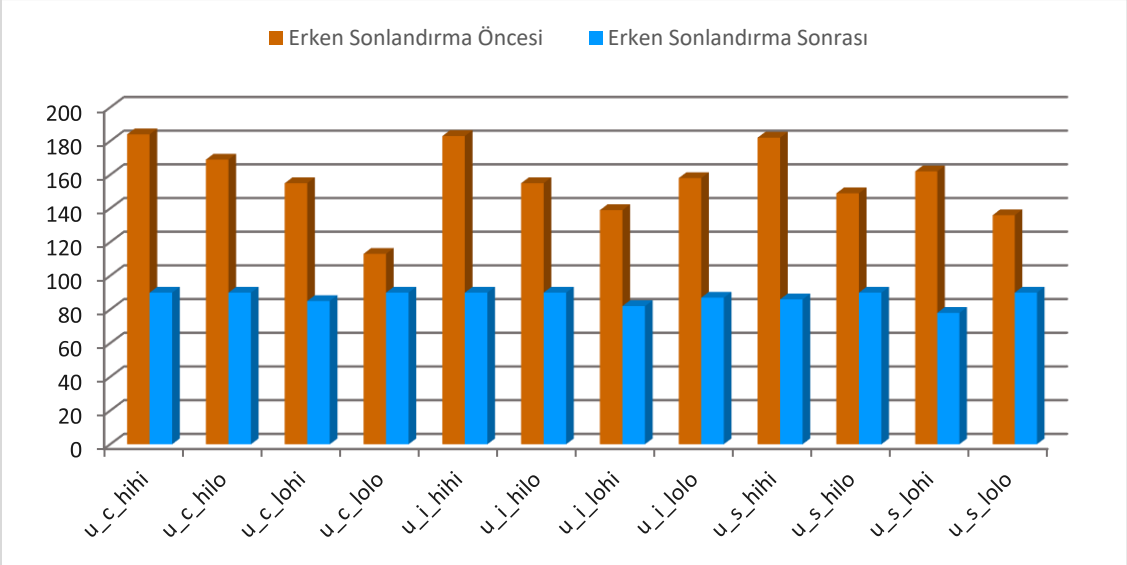
Geliştirilen bilgisayar programında farklı arama parametreleri kullanılarak daha iyi sonuçlara ulaşılmaya çalışılmıştır. Ancak parametrelerin değerlerindeki artışa paralel olarak programın çalışma süresinde de artış gözlenmiştir. Bu durumda programın çalışma süresini iyileştirmek için iç döngü ve dış döngü erken sonlandırma koşulları oluşturulmuştur. İç döngü erken sonlandırma işlemi için aşağıda belirtilen dört farklı koşul test edilmiştir.

1. Herhangi bir sıcaklıkta ΔC'nin sıfırdan küçük olduğu çözümlerin sayısının kullanıcı tarafından belirlenen bir eşik değerini (*threshold*) geçmesi durumunda sonlandırma yapılmıştır.
2. Herhangi bir sıcaklıkta ΔC'nin sıfırdan büyük olduğu çözümlerin sayısının kullanıcı tarafından belirlenen bir eşik değerini (*threshold*) geçmesi durumunda sonlandırma yapılmıştır.
3. Herhangi bir sıcaklıkta Metropolis kriterinin sağlanma sayısının kullanıcı tarafından belirlenen bir eşik değerini (*threshold*) geçmesi durumunda sonlandırma yapılmıştır.
4. Herhangi bir sıcaklıkta en iyi çözümün bulunma sayısının kullanıcı tarafından belirlenen bir eşik değerini (*threshold*) geçmesi ile durumunda sonlandırma yapılmıştır.

Dört farklı koşul arasından 2. veya 3. koşul ile iç döngü erken sonlandırma işlemi yapıldığında programın çalışma süresinin ortalama birkaç saniye sürdüğü ve oldukça düşük kalitede sonuçlar üretildiği görülmüştür. 4. koşula bağlı erken sonlandırma işlemi kullanıldığında ise programın çalışma süresinde artış oluştuğundan iç döngü erken sonlandırma işlemi için yüksek kalitede sonuçları ve çalışma süresini veren 1. koşul tercih edilmiştir.

Dış döngü erken sonlandırma işlemi ise görev sayısına (N) bağlı olarak gerçekleştirilmiştir. Program görev sayısının farklı katları (N, N/2, N/4, N/6, N/8) ile test edilmiş ve dış döngü sonlandırma işlemi için en uygun değerin N/4 olduğuna karar verilmiştir.

Şekil 4.13'de erken sonlandırma koşuluna bağlı optimizasyon öncesi ve sonrası her bir veri seti için programın çalışma süresi verilmiştir.

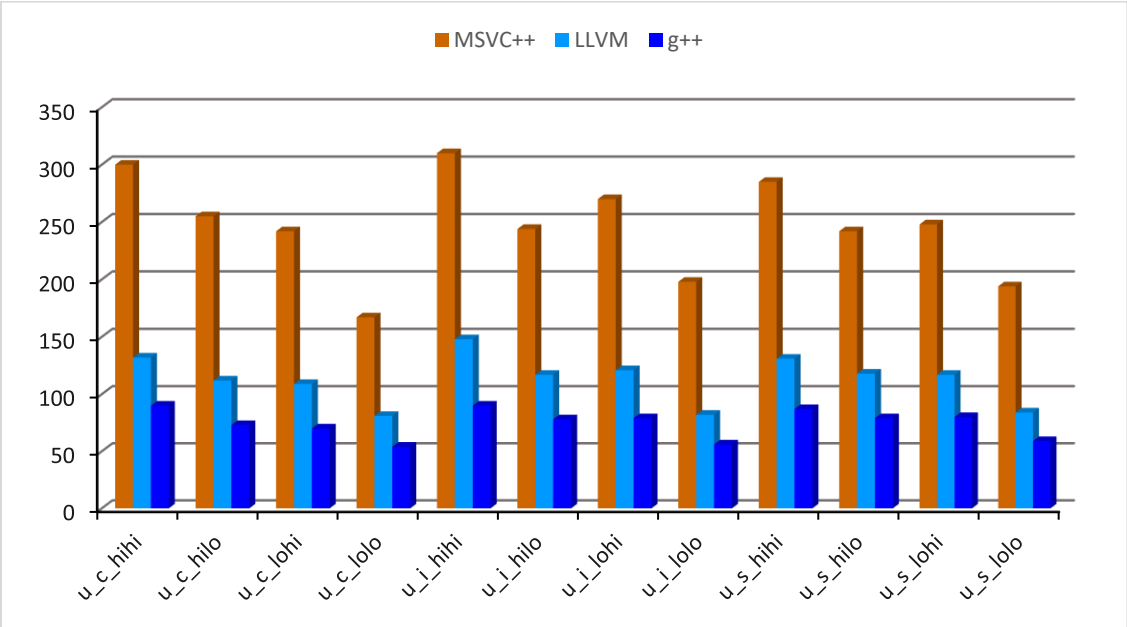


Şekil 4.13. Erken sonlandırma öncesi ve sonrası çalışma süreleri

4.6. Derleyici Etkisine Bağlı Optimizasyon

Doğru sonuçlar üreten bir bilgisayar programının çalışma süresini etkileyen en önemli faktörlerden birisi de kullanılan derleyicidir. Derleyicilerin bağlı olduğu uygulama, kullandıkları bellek türü ve ortamın işlemci hızı programın çalışma süresini doğrudan etkilemektedir. Program g++, LLVM ve MSVC++ olmak üzere üç farklı derleyici ile çalıştırıldığında Şekil 4.14'de yer alan grafikte de görüldüğü gibi her bir derleyicinin programı çalıştırma süresinde farklılık olduğu tespit edilmiştir. Grafikteki sonuçları dikkate alarak en ideal çalışma süresini veren g++ derleyicisi tercih edilmiştir.

Derleyicilerin programı derleme sürelerini azaltmak ve/veya kodun performansını ön plana çıkarmak için her bir derleyicide optimizasyon bayrakları kullanılmaktadır. Program g++ derleyicisine ait üç farklı optimizasyon bayrağı (*O2, O3, Ofast*) ile derlenmiştir. Her bir optimizasyon bayrağının programı derleme süresine etkisinin birbirinden farklı olduğu ve en iyi derleme süresine *Ofast* bayrağı ile ulaşıldığı görülmüştür.



Şekil 4.14. Farklı derleyiciler için çalışma süreleri

5. SONUÇLAR ve ÖNERİLER

Bu tez kapsamında öncelikle büyük veri ve bulut bilişim kavramları, bulut bilişim hizmeti sağlayan altyapılarda kullanılan sanallaştırma teknolojisi, Hadoop yapısı ve çizelgeleme hakkında detaylı bilgiler verilmiştir. Sonraki aşamada literatür araştırılarak görev çizelgeleme üzerine çalışan farklı algoritmalardan bahsedilmiştir. Daha sonra sezgisel ve metasezgisel yaklaşımlar hakkında bilgi verilmiş ve bulut görev çizelgelemesi için benzetilmiş tavlama tabanlı bir metasezgisel yaklaşım geliştirilmiştir. Bu yaklaşımın seri ve paralel versiyonları C++ programlama dili kullanılarak bir bilgisayar programına dönüştürülmüştür. Paralel versiyon için aynı zamanda OpenMP kütüphanesinden faydalanılmıştır. Geliştirilen bilgisayar programının çalışma süresinin azaltılması için, benzetilmiş tavlamanın ihtiyaç duyduğu farklı rastgele sayı üretme ve pertürbasyon teknikleri, veri yapıları, döngü sonlandırma koşulları, keşif-sömürme oranları ve derleyici etkisi yine bu tez kapsamında detaylı olarak analiz edilmiştir.

Son bölümde ise geliştirilen yaklaşımın yetkinliği görev çizelgeleme algoritmalarının performanslarını karşılaştırmak için kullanılan ve Braun modeliyle oluşturulmuş on iki meşhur benchmark ile test edilmiştir. Önerilen yaklaşım, literatürde şu ana kadar rapor edilen en iyi ve ortalama gecikme değerleri ile kıyaslanmış ve sonuçlar grafiklerle alt başlıklarda detaylı analiz edilmiştir.

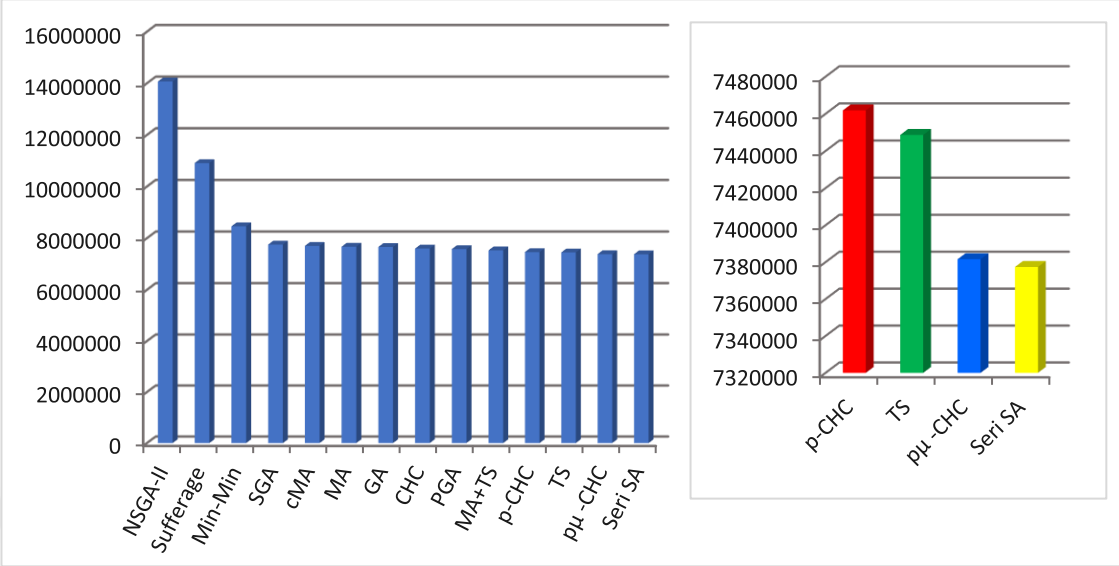
Önerilen yaklaşımın hem seri hem de paralel versiyonu, tüm benchmarklar için, literatürde şu ana kadar bir sezgisel veya metasezgisel kullanılarak rapor edilen en iyi gecikme değerlerinden daha iyi sonuçlar üretmiştir. Öte yandan hem seri hem de paralel versiyonda birçok benchmark için yine literatürde rapor edilen en iyi ortalama değerlerden daha iyi değerlere ulaşılmıştır. Programda çalışma süresi için 90 saniye kısıtlaması kullanılmıştır. Sürenin 90 saniye ile kısıtlanması literatürde yer alan sezgisel veya metasezgisel yaklaşımlara ait en iyi gecikme değerlerine 90 saniyede (cMA maksimum 90 saniye, MA maksimum 90 saniye, TS maksimum 100 saniye, MA+TS maksimum 90 saniye, SGA maksimum 90 saniye, GA maksimum 90 saniye, CHC maksimum 90 saniye, pCHC maksimum 90 saniye, pμCHC maksimum 90 saniye) ulaşılmasından kaynaklanmaktadır.

Paralel versiyonda farklı sayıda iş parçacıkları (2, 4 ve 8) için ulaşılan sonuçlardaki kalite farklılıkları iş parçacıklarının ek yönetim maliyetinden (*overhead),* başka bir deyişle iş parçacıklarının oluşturulmaları, çalıştırılmaları ve yıkılmaları sırasında ortaya çıkan ek yükden (ek süre) kaynaklanmaktadır. Her bir iş parçacığının getirdiği bu ek yük programın çalışma süresini arttırmaktadır. Çalışma süresinde meydana gelen artışı önlemek için benzetilmiş tavlama yaklaşımının temel parametreleri olan alfa (α) ve iterasyon sayısı (S) azaltılmıştır. Bu nedenle iş parçacığı sayısının arttırılmasına rağmen bazı veri setlerinde ulaşılan sonuçların kalitesi artmamış, aksine azalmıştır.

5.1. Seri benzetilmiş tavlama yaklaşımı için analiz sonuçları

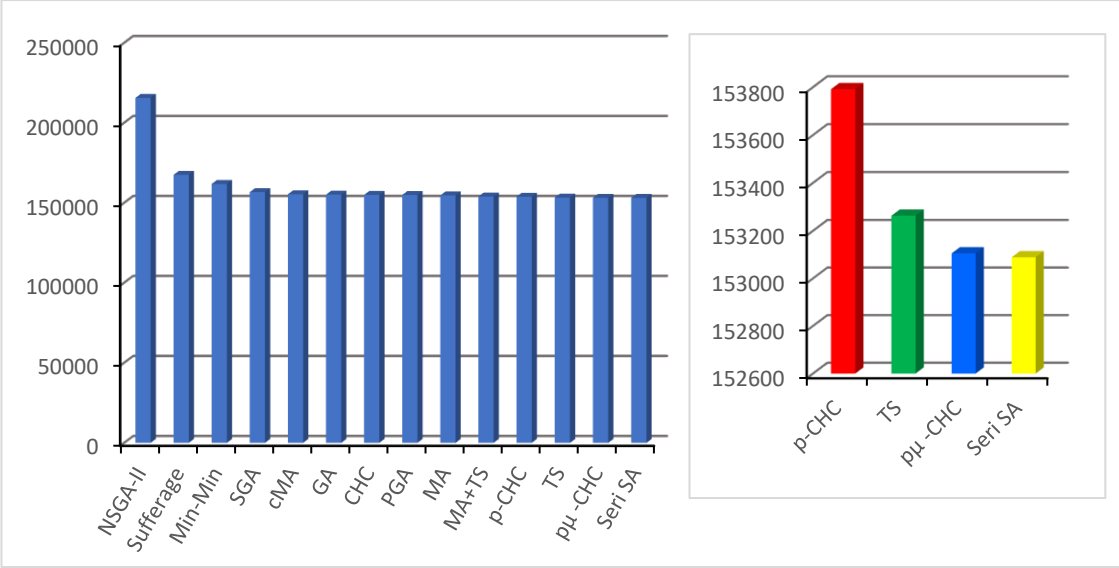
Benzetilmiş tavlama yaklaşımını kullanarak hazırladığımız seri program ile literatürdeki on üç farklı sezgisel ve metasezgisel yaklaşımın Braun modelinde yer alan her bir veri seti için ulaştıkları en iyi ve ortalama gecikme değerlerinin karşılaştırılması grafiklerde gösterilmiştir.

Modeldeki "u_c_hihi" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.1'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %47.5, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.05 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi gecikme değerine ulaştığı görülmektedir.



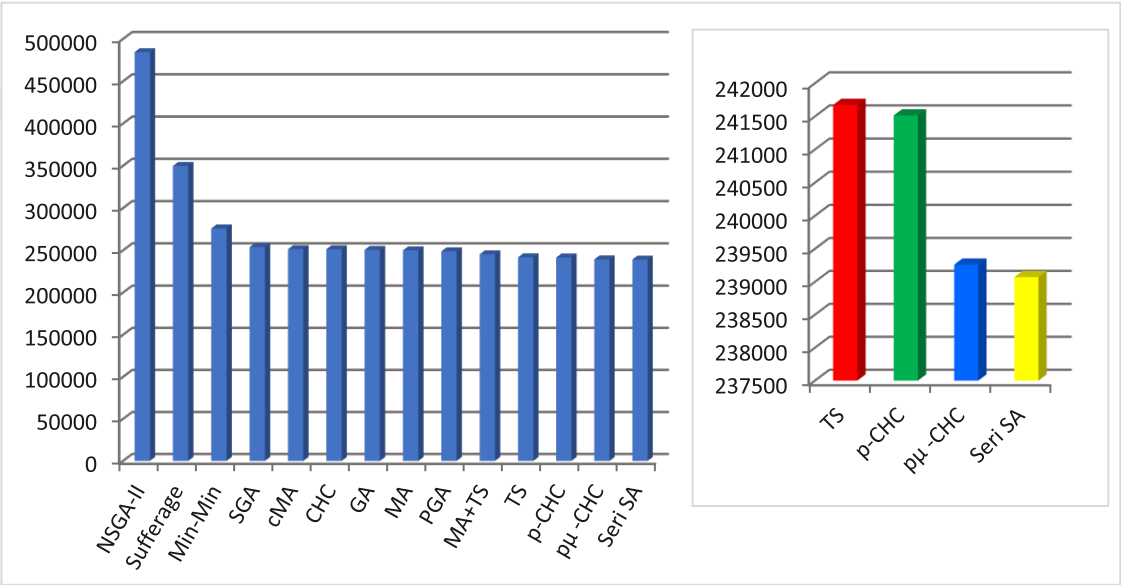
Şekil 5.1. u_c_hihi için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_c_hilo" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.2'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %29.0 daha iyi, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.01 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi gecikme değerine ulaştığı görülmektedir.



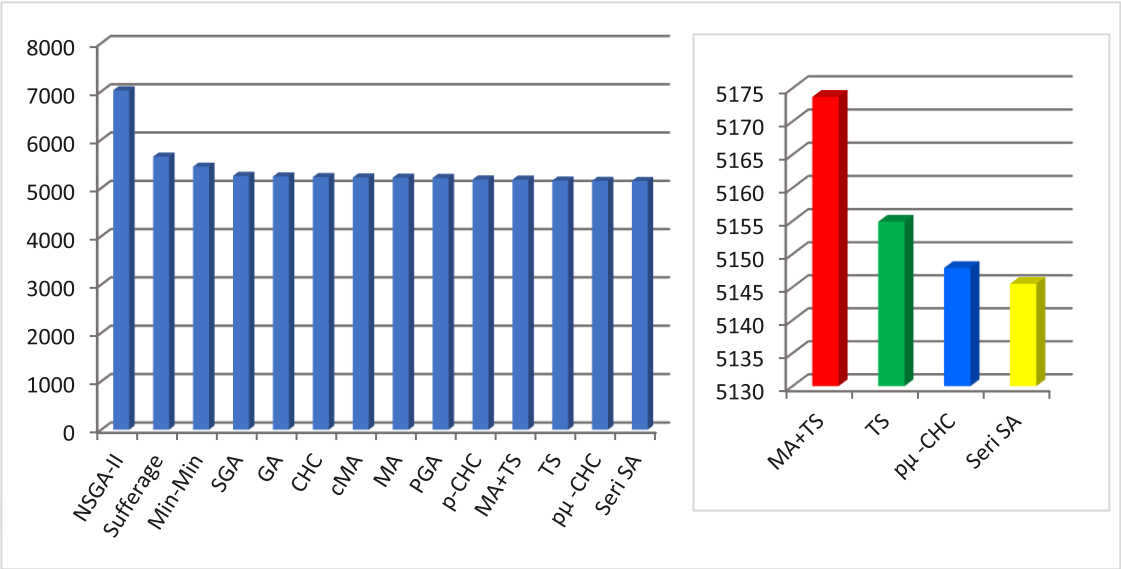
Şekil 5.2. u_c_hilo için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_c_lohi" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.3'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %50.6, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.07 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi gecikme değerine ulaştığı görülmektedir.



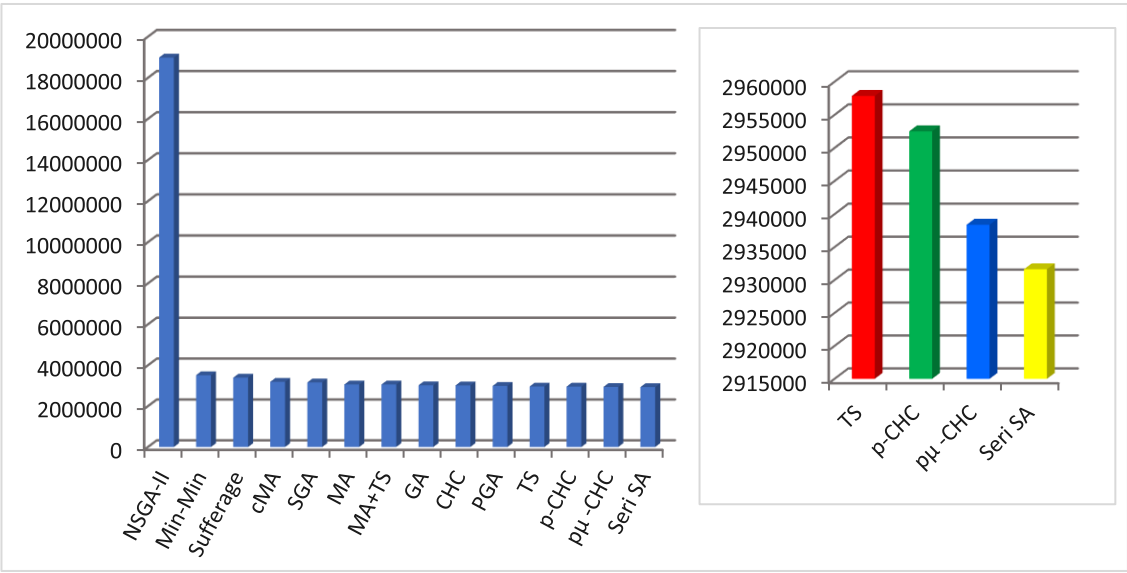
Şekil 5.3. u_c_lohi için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_c_lolo" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.4'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %26.7, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.04 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi geçikme değerine ulaştığı görülmektedir.



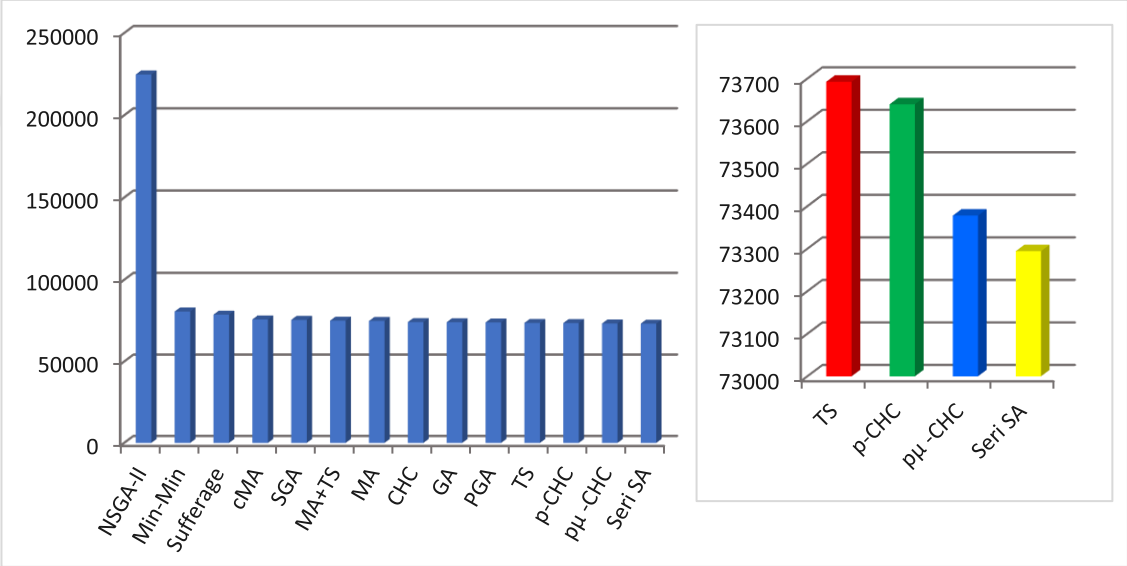
Şekil 5.4. u_c_lolo için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_i_hihi" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.5'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %84.5, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.22 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi geçikme değerine ulaştığı görülmektedir.



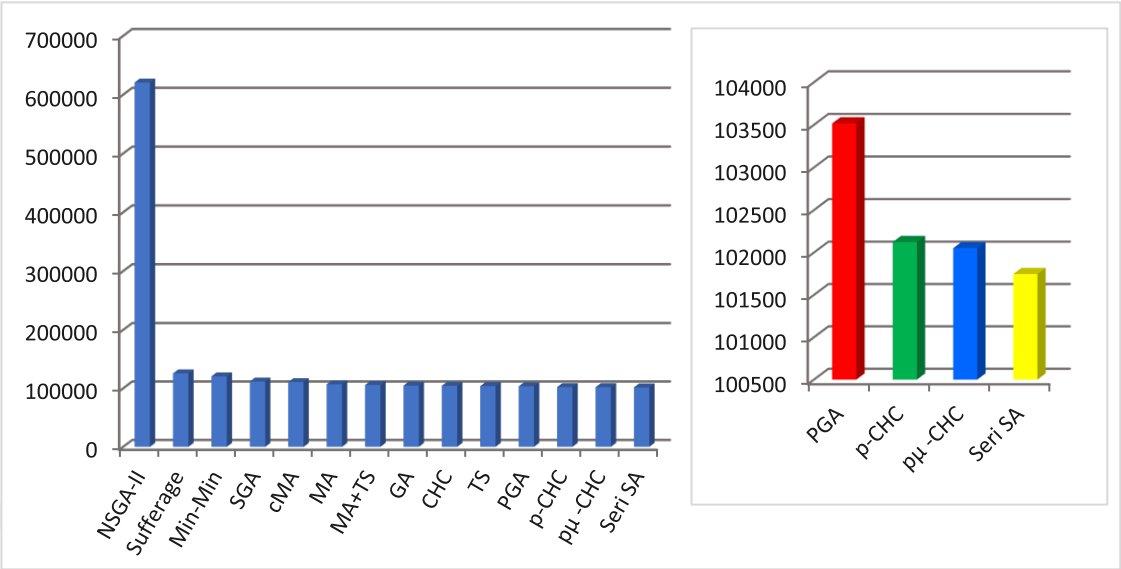
Şekil 5.5. u_i_hihi için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_i_hilo" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.6'da gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %67.3, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.11 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi geçikme değerine ulaştığı görülmektedir.



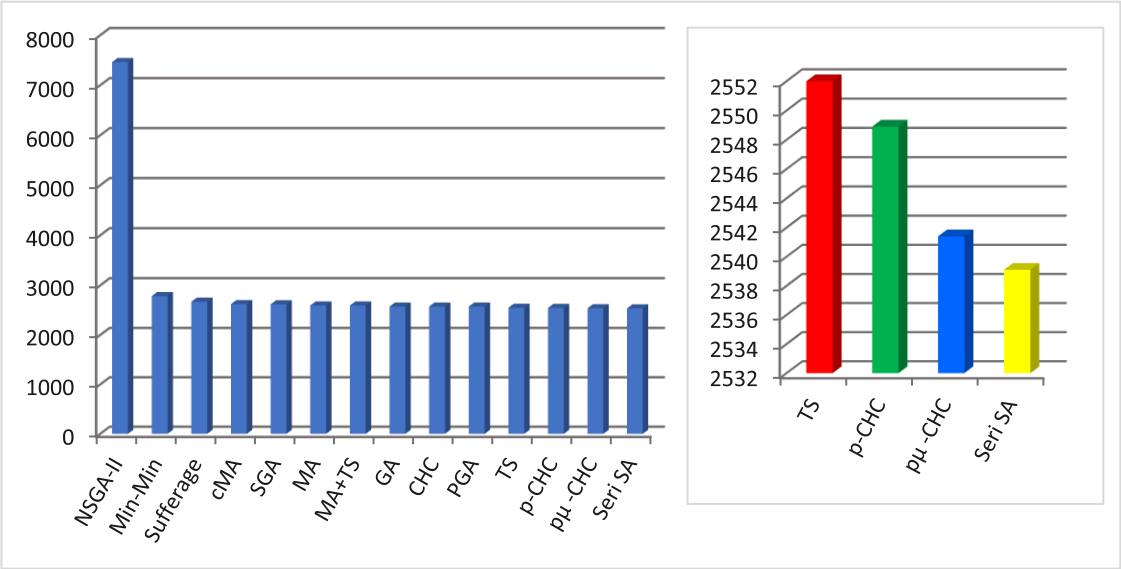
Şekil 5.6. u_i_hilo için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_i_lohi" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.7'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %83.6, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.29 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi geçikme değerine ulaştığı görülmektedir.



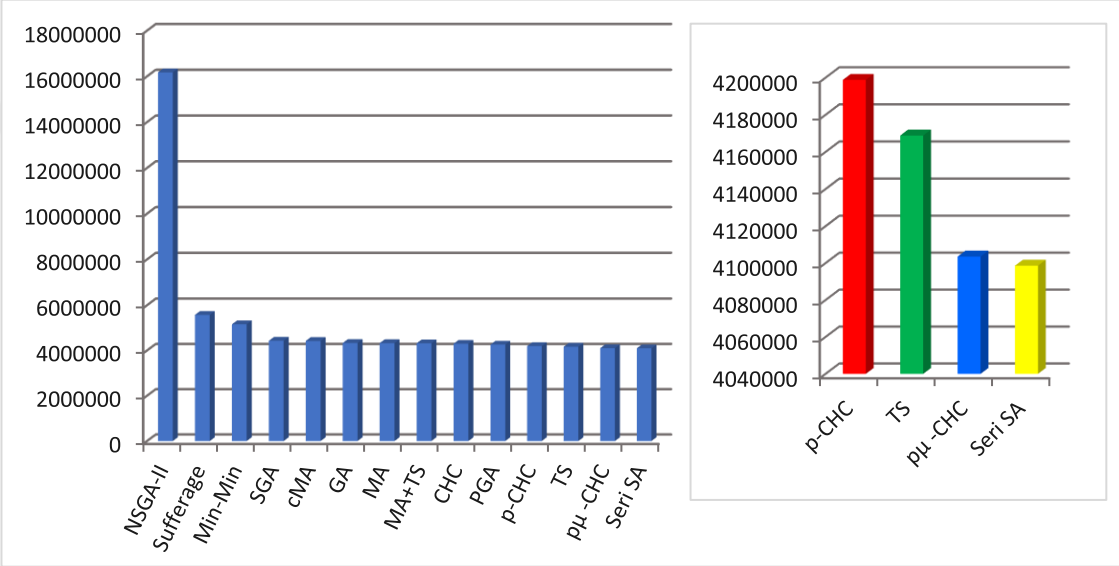
Şekil 5.7. u_i_lohi için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_i_lolo" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.8'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %65.9, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.09 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi geçikme değerine ulaştığı görülmektedir.



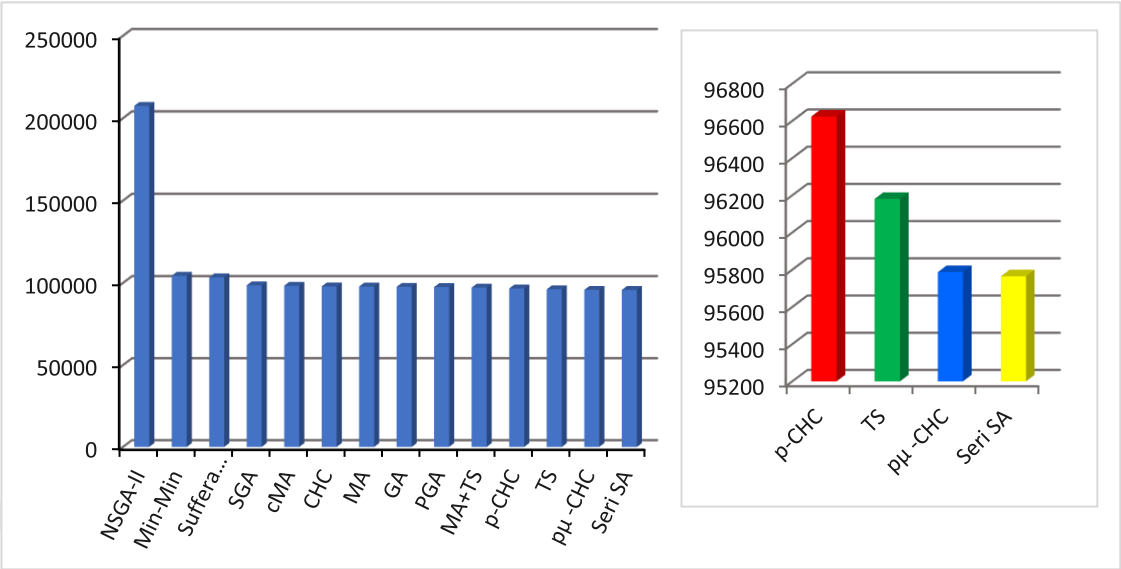
Şekil 5.8. u_i_lolo için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_s_hihi" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.9'da gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %74.6, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.11 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi geçikme değerine ulaştığı görülmektedir.



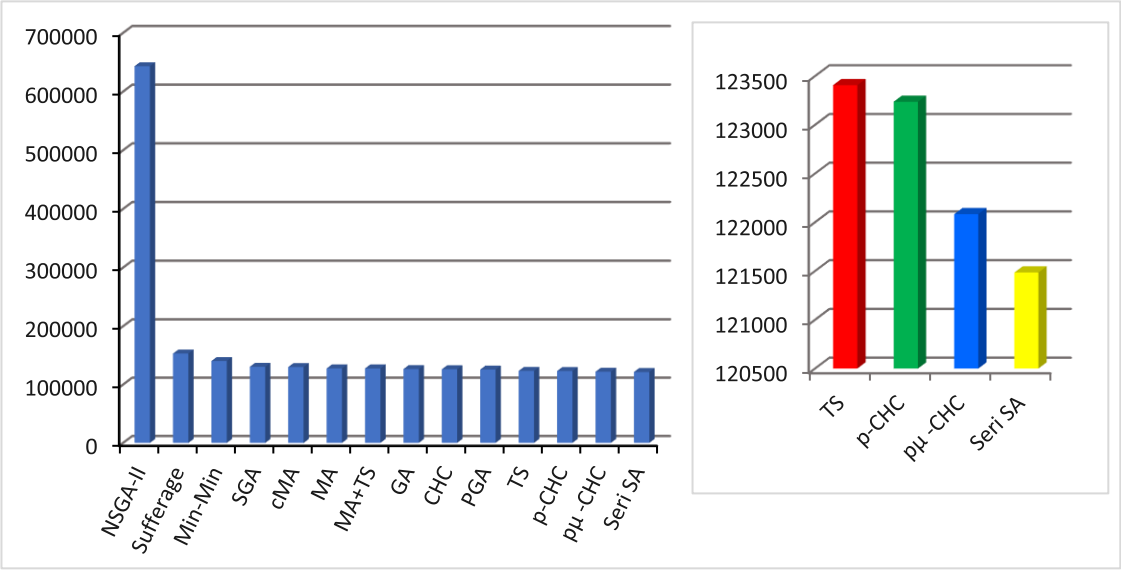
Şekil 5.9. u_s_hihi için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_s_hilo" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.10'da gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %53.8, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.02 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi geçikme değerine ulaştığı görülmektedir.



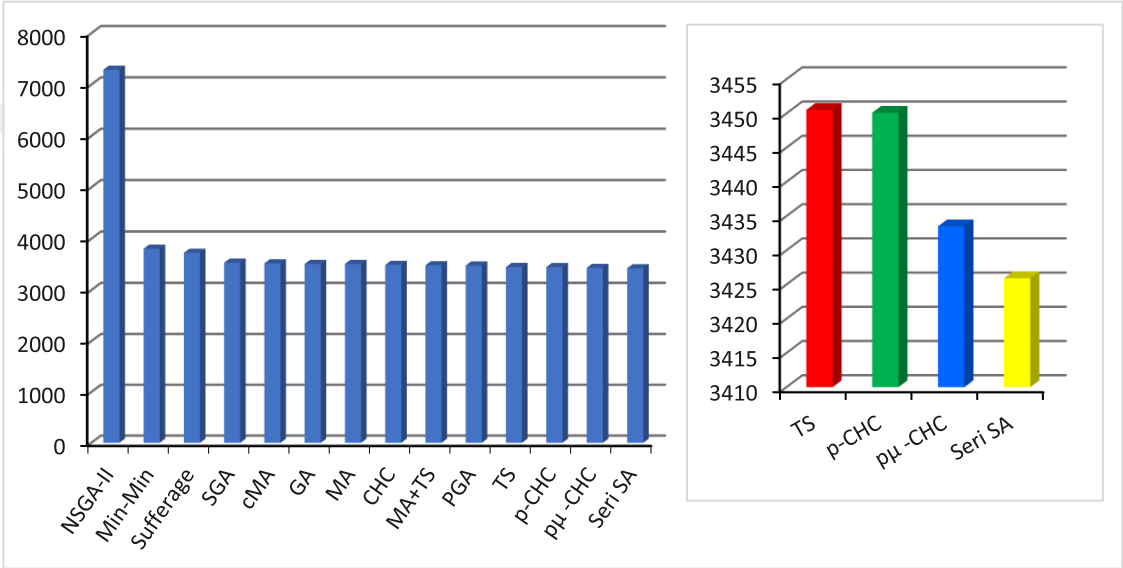
Şekil 5.10. u_s_hilo için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_s_lohi" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.11'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %81.1, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.49 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi geçikme değerine ulaştığı görülmektedir.



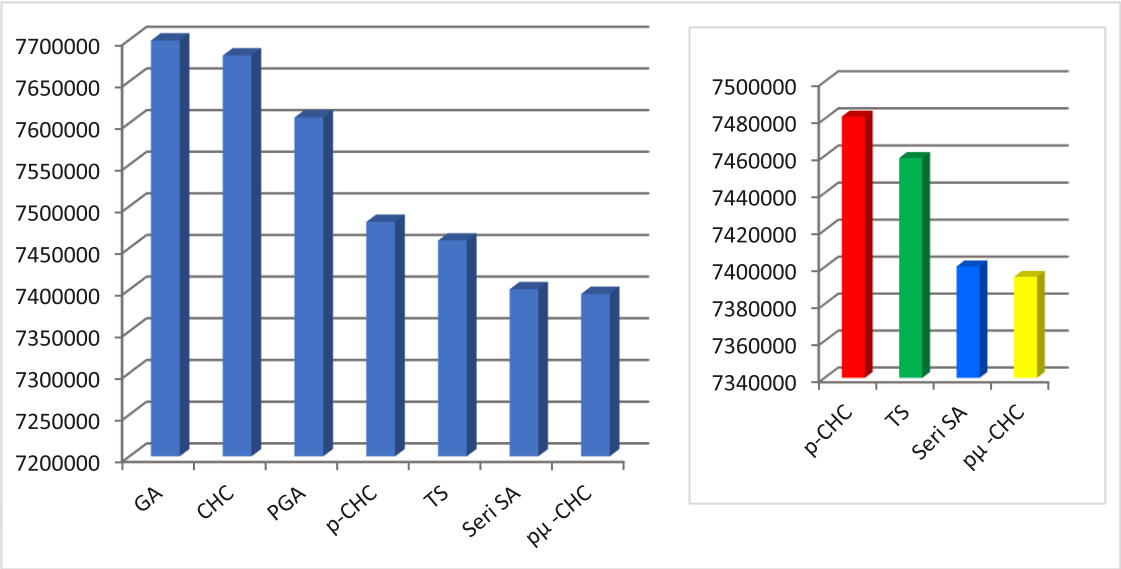
Şekil 5.11. u_s_lohi için farklı algoritmaların en iyi gecikme değerleri

Modeldeki "u_s_lolo" veri seti için tüm yaklaşımların ulaştıkları en iyi gecikme değerleri Şekil 5.12'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek gecikme değerine sahip NSGA-II yaklaşımına göre %52.9, kendisine en yakın gecikme değerine sahip pμCHC yaklaşımına göre %0.22 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi gecikme değerine ulaştığı görülmektedir.



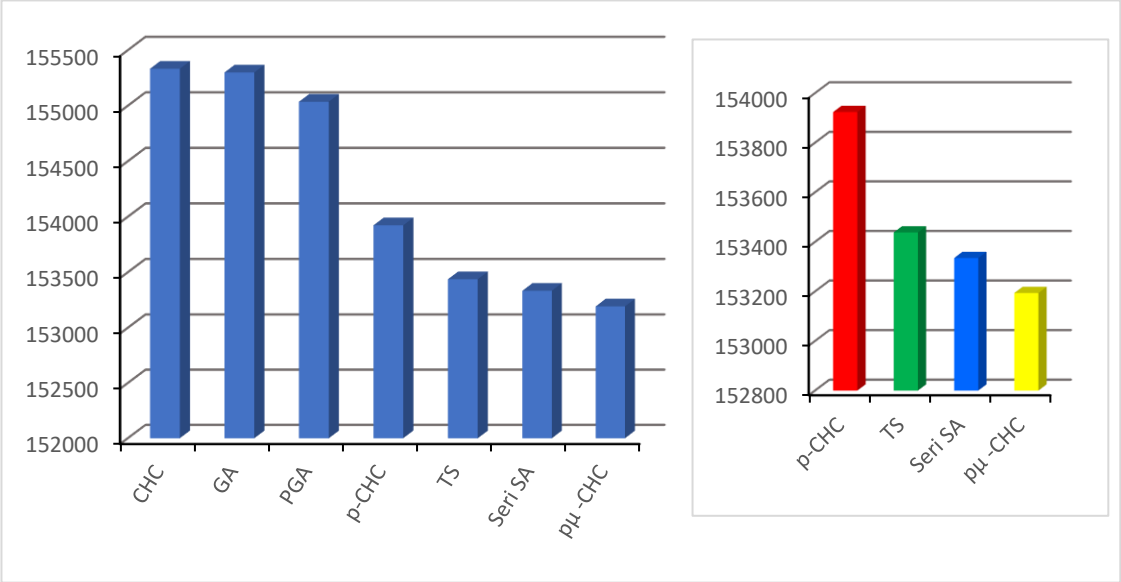
Şekil 5.12. u_s_lolo için farklı algoritmaların en iyi gecikme değerleri

Benzer şekilde programın literatürdeki altı farklı sezgisel ve metasezgisel yaklaşımın Braun modelinde yer alan "u_c_hihi" veri seti için ulaştıkları ortalama gecikme değerleri Şekil 5.13'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama değerine sahip GA yaklaşımına göre %3.9 daha iyi, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.07 oranında daha düşük kalitede olduğu görülmektedir.



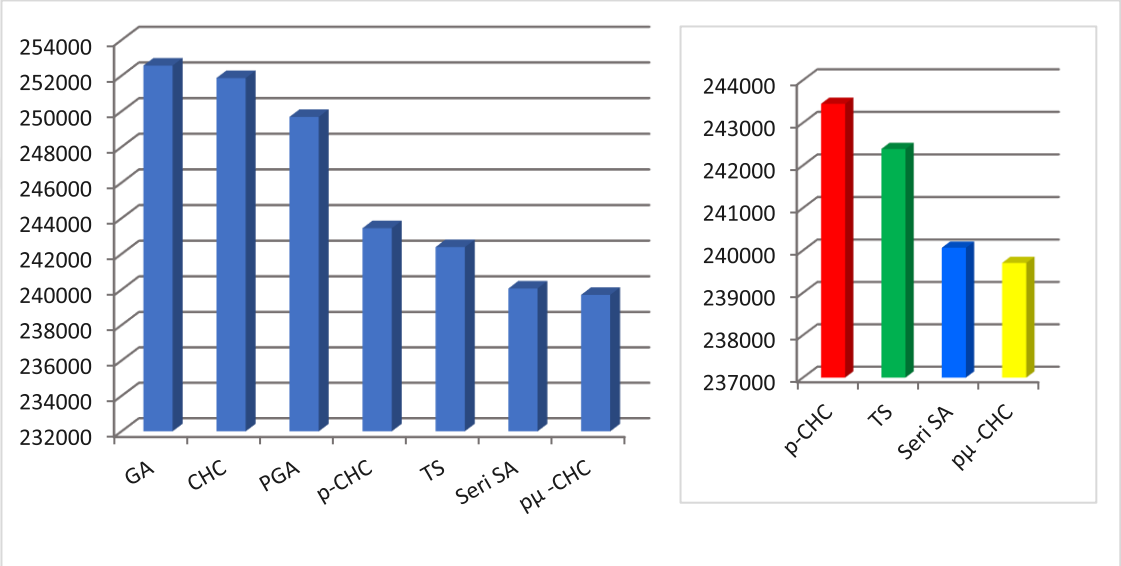
Şekil 5.13. u_c_hihi için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_c_hilo" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.14'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip CHC yaklaşımına göre %1.2 daha iyi, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.09 oranında daha düşük kalitede olduğu görülmektedir.



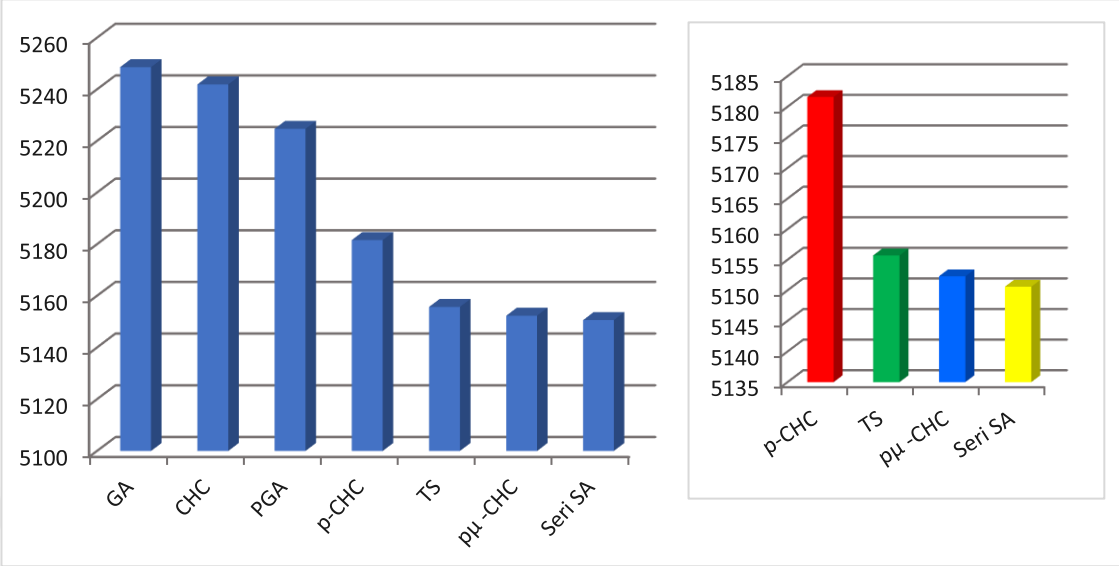
Şekil 5.14. u_c_hilo için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_c_lohi" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.15'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip GA yaklaşımına göre %4.9 daha iyi, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.14 oranında daha düşük kalitede olduğu görülmektedir.



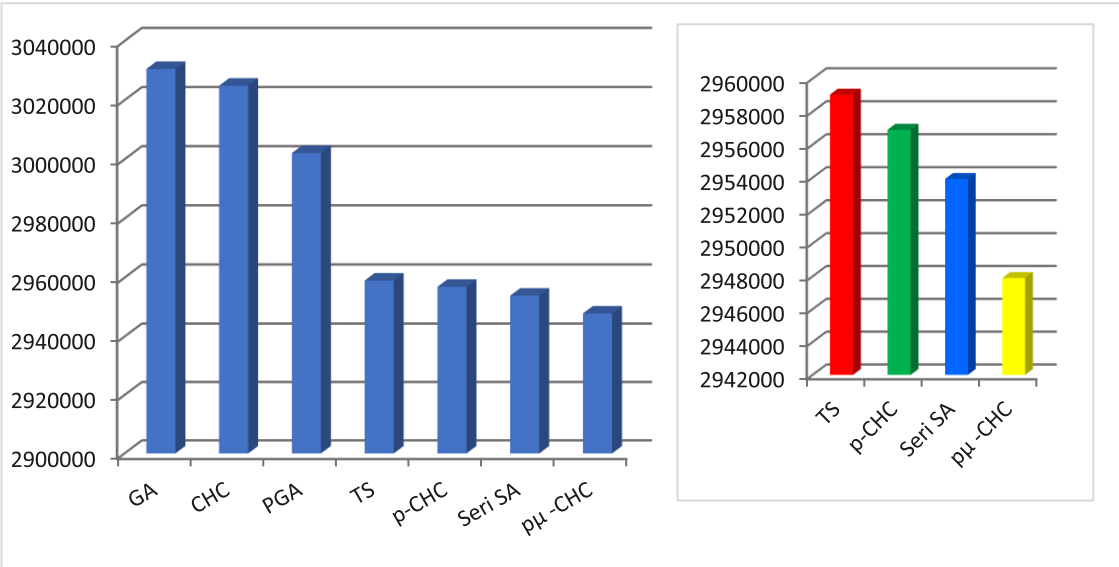
Şekil 5.15. u_c_lohi için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_c_lolo" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.16'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip GA yaklaşımına göre %1.8, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.03 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi ortalama değerine ulaştığı görülmektedir.



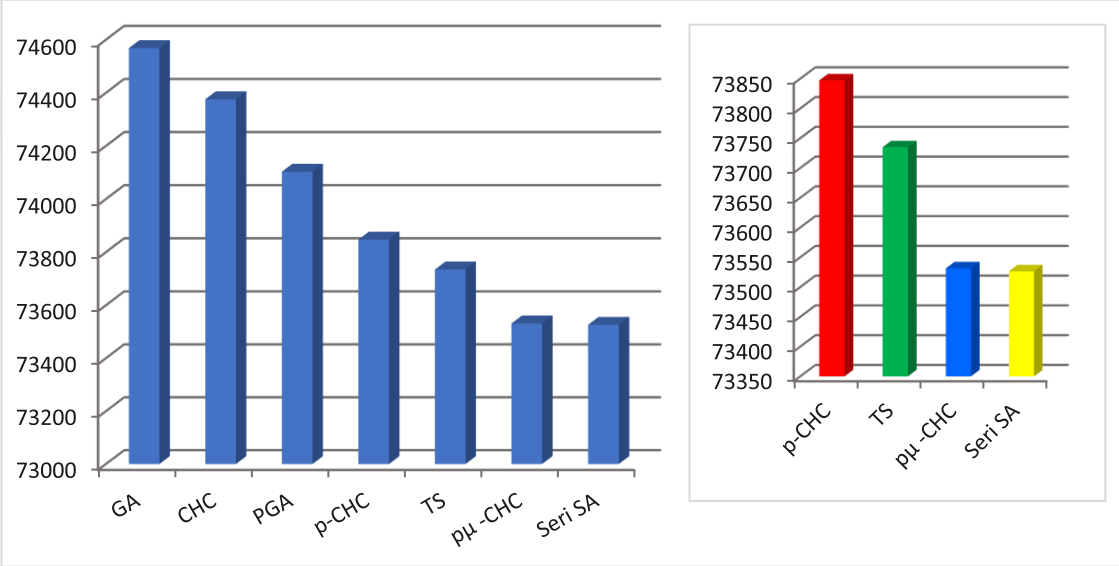
Şekil 5.16. u_c_lolo için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_i_hihi" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.17'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip GA yaklaşımına göre %2.5 daha iyi, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.20 oranında daha düşük kalitede olduğu görülmektedir.



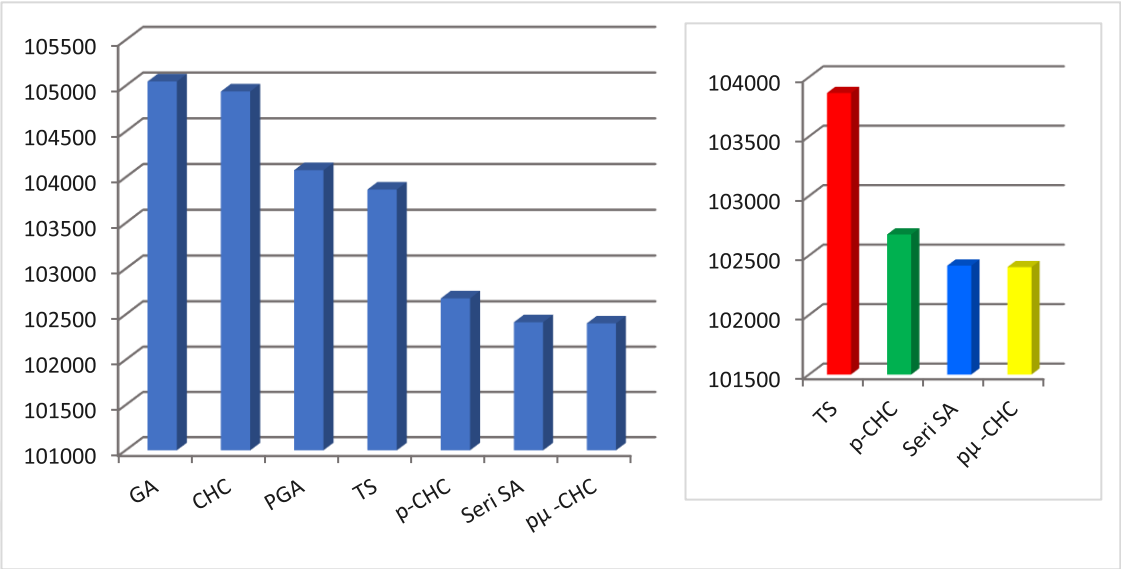
Şekil 5.17. u_i_hihi için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_i_hilo" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.18'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip GA yaklaşımına göre %1.3, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.006 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi ortalama değerine ulaştığı görülmektedir.



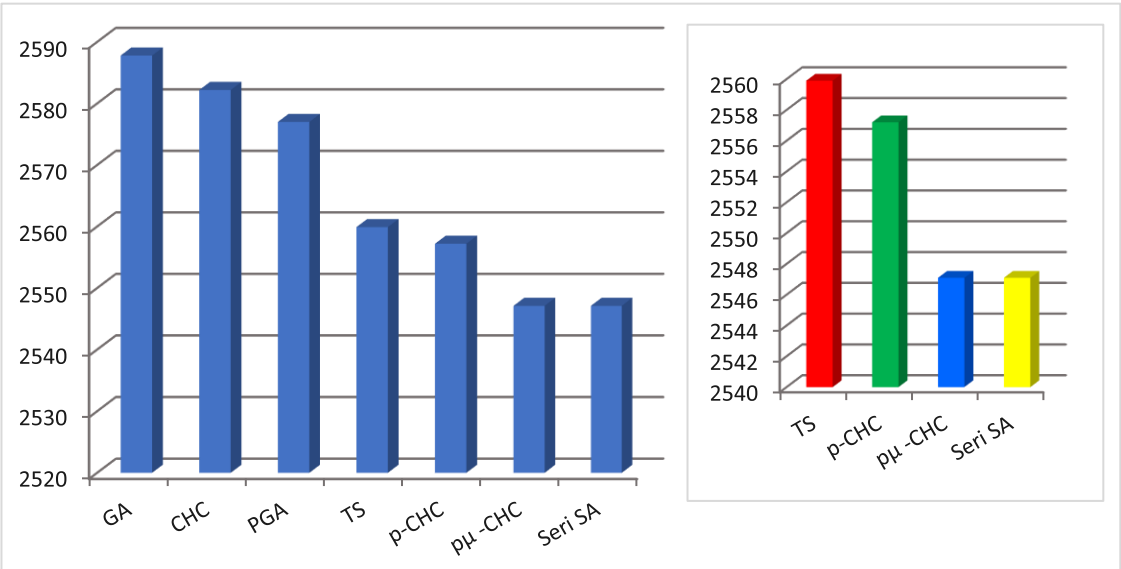
Şekil 5.18. u_i_hilo için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_i_lohi" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.19'da gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip GA yaklaşımına göre %2.5 daha iyi, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.01 oranında daha düşük kalitede olduğu görülmektedir.



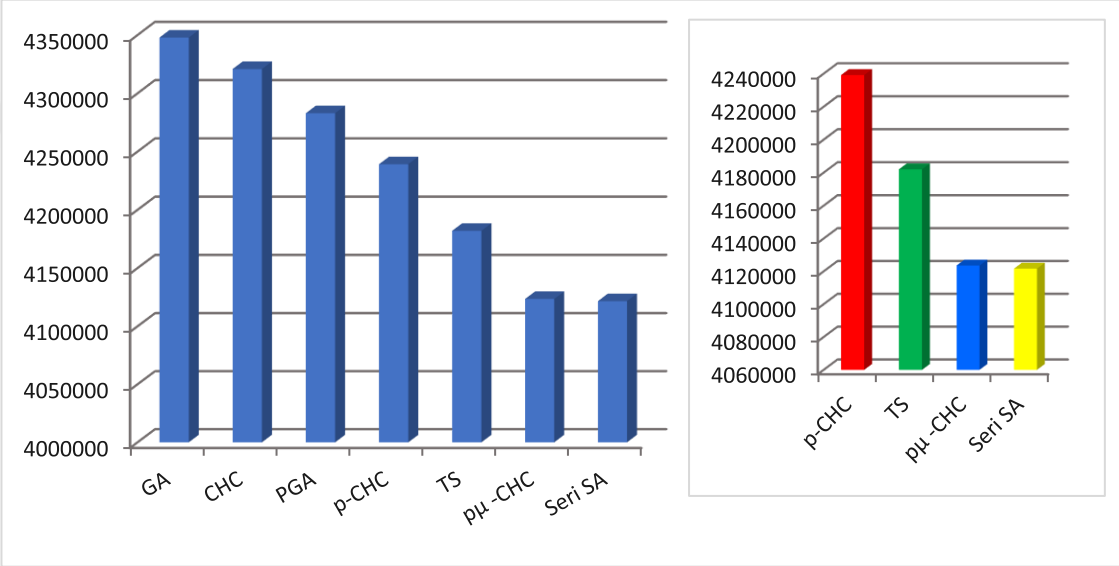
Şekil 5.19. u_i_lohi için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_i_lolo" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.20'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip GA yaklaşımına göre %1.5, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımı ile aynı olduğu ve diğer tüm yaklaşımlar arasında en iyi ortalama değerine ulaştığı görülmektedir.



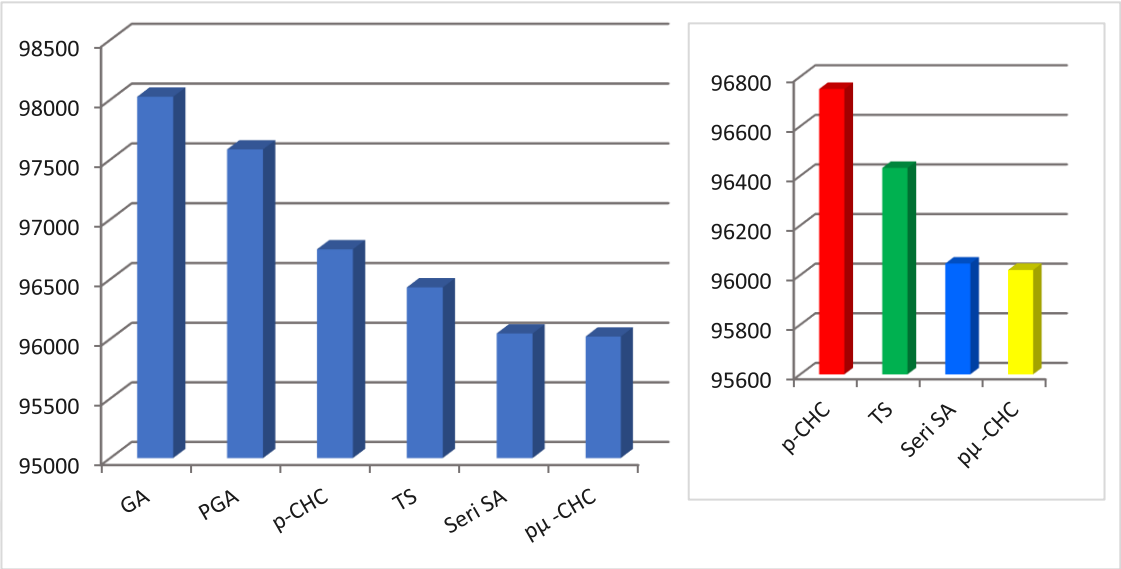
Şekil 5.20. u_i_lolo için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_s_hihi" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.21'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip GA yaklaşımına göre %5.2, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.04 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi ortalama değerine ulaştığı görülmektedir.



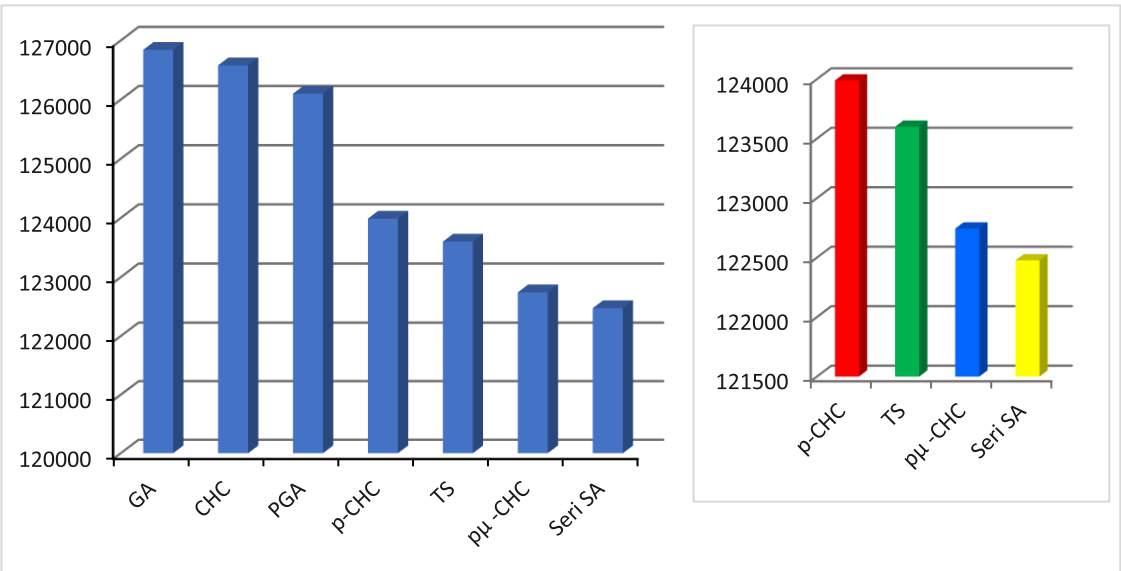
Şekil 5.21. u_s_hihi için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_s_hilo" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.22'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip GA yaklaşımına göre %2.2 daha iyi, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.02 oranında daha düşük kalitede olduğu görülmektedir.



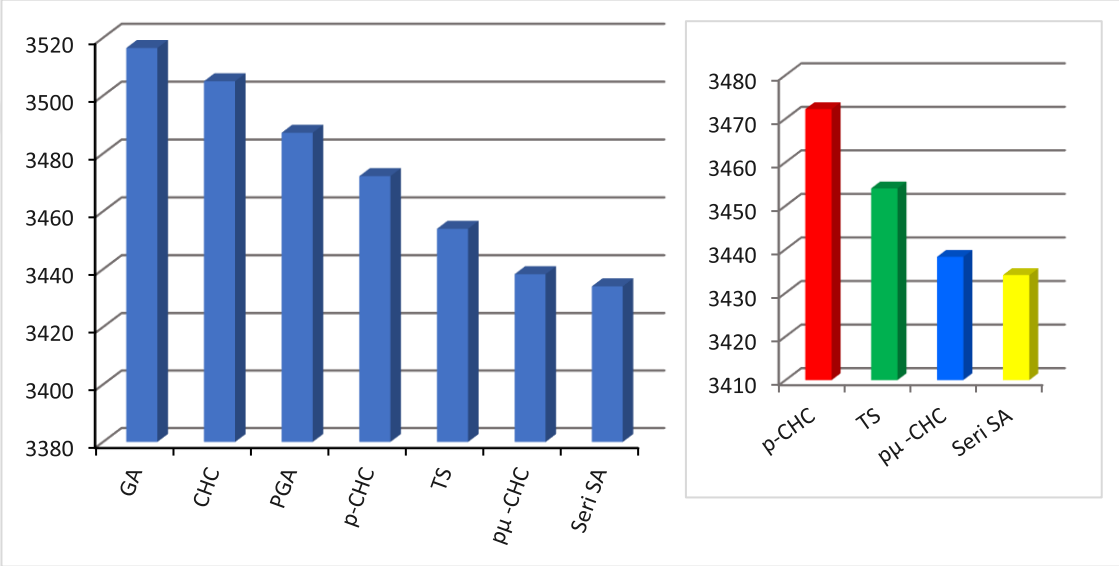
Şekil 5.22. u_s_hilo için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_s_lohi" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.23'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip GA yaklaşımına göre %3.4, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.21 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi ortalama değerine ulaştığı görülmektedir.



Şekil 5.23. u_s_lohi için farklı algoritmaların ortalama gecikme değerleri

Modeldeki "u_s_lolo" veri seti için tüm yaklaşımların ulaştıkları ortalama gecikme değerleri Şekil 5.24'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde SA'nın belirtilen veri seti için en yüksek ortalama gecikme değerine sahip GA yaklaşımına göre %2.3, kendisine en yakın ortalama gecikme değerine sahip pμCHC yaklaşımına göre %0.12 oranında daha iyi olduğu ve tüm yaklaşımlar arasında en iyi ortalama değerine ulaştığı görülmektedir.



Şekil 5.24. u_s_lolo için farklı algoritmaların ortalama gecikme değerleri

Braun modeline ait her bir veri seti için seri benzetilmiş tavlama yaklaşımı ile ulaşılan en iyi gecikme değerleri ve literatürde aynı model için rapor edilen diğer yaklaşımlara ait en iyi gecikme değerleri Çizelge 5.1, Çizelge 5.2 ve Çizelge 5.3'de yer almaktadır. Çizelgelerde seri SA yaklaşımı ile ulaşılan ve literatürde şu ana kadar rapor edilen en iyi değerlerden daha iyi sonuçlar kırmızı renkle gösterilmiştir. Öte yandan Çizelge 5.4'de seri benzetilmiş tavlama yaklaşımı ile ulaşılan ortalama gecikme değerleri ile literatürde aynı model için rapor edilen diğer yaklaşımlara ait ortalama gecikme değerleri yer almaktadır. Çizelgede seri SA yaklaşımı ile ulaşılan ve literatürde şu ana kadar rapor edilen ortalama değerlerden daha iyi sonuçlar mavi renkle gösterilmiştir.

Çizelge 5.1. Farklı algoritmaların en iyi gecikme değerleri 1

BENCHMARK	NSGA-II (Subashini *et al.* 2010)	Sufferage (Xhafa *et al.* 2007b)	Min-Min (Xhafa *et al.* 2007b)	SGA (Xhafa *et al.* 2008c)	cMA (Xhafa *et al.* 2008a)	Seri SA
u_c_hihi.0	14071196.1	10908697.8	8460675.0	7752689.0	7700929.7	*7377378.2*
u_c_hilo.0	215624.7	167483.2	161805.4	156680.5	155334.8	*153088.7*
u_c_lohi.0	484360.5	349746.0	275837.3	253926.0	251360.2	*239069.9*
u_c_lolo.0	7022.6	5649.8	5441.4	5251.1	5218.1	*5145.5*
u_i_hihi.0	18981587.8	3391758.3	3513919.2	3161104.9	3186664.7	*2931630.9*
u_i_hilo.0	224802.5	78828.2	80755.6	75598.4	75856.6	*73294.3*
u_i_lohi.0	620779.0	125688.6	120517.7	111792.1	110620.7	*101737.7*
u_i_lolo.0	7462.8	2673.8	2785.6	2620.7	2624.2	*2539.1*
u_s_hihi.0	16179189.2	5574357.7	5160342.8	4433792.2	4424540.8	*4098580.9*
u_s_hilo.0	207484.6	103400.8	104375.1	98560.0	98283.7	*95764.9*
u_s_lohi.0	643263.1	153094.0	140284.4	130425.8	130014.5	*121484.8*
u_s_lolo.0	7288.9	3727.9	3806.8	3534.3	3522.0	*3425.9*

Çizelge 5.2. Farklı algoritmaların en iyi gecikme değerleri 2

BENCHMARK	GA (Nesmachnow*et al.* 2010)	MA (Xhafa 2007a)	CHC (Nesmachnow*et al.* 2010)	MA+TS (Xhafa 2007a)	Seri SA
u_c_hihi.0	7659878.7	7669920.4	7599288.4	7530020.1	*7377378.2*
u_c_hilo.0	155092.0	154631.1	154947.0	153917.1	*153088.7*
u_c_lohi.0	250511.8	249950.8	251194.3	245288.9	*239069.9*
u_c_lolo.0	5239.1	5213.0	5225.9	5173.7	*5145.5*
u_i_hihi.0	3019844.3	3058785.6	3015048.5	3058474.9	*2931630.9*
u_i_hilo.0	74142.9	74939.8	74240.9	75108.4	*73294.3*
u_i_lohi.0	104688.0	107038.8	104546.0	105808.5	*101737.7*
u_i_lolo.0	2577.0	2598.4	2576.7	2596.5	*2539.1*
u_s_hihi.0	4332248.2	4327249.7	4299146.0	4321015.4	*4098580.9*
u_s_hilo.0	97630.1	97804.7	97888.2	97177.2	*95764.9*
u_s_lohi.0	126438.0	127648.9	126238.0	127633.0	*121484.8*
u_s_lolo.0	3510.4	3510.0	3492.1	3484.0	*3425.9*

Çizelge 5.3. Farklı algoritmaların en iyi gecikme değerleri 3

BENCHMARKS	PGA (Nesmachnow*et al.* 2010)	TS (Xhafa *et al.* 2008b)	p-CHC (Nesmachnow*et al.* 2010)	pμCHC (Nesmachnow*et al.* 2012)	Seri SA
u_c_hihi.0	7577921.9	7448640.4	7461819.1	7381570.0	*7377378.2*
u_c_hilo.0	154915.0	153263.3	153791.9	153105.4	*153088.7*
u_c_lohi.0	248772.4	241672.6	241513.2	239260.0	*239069.9*
u_c_lolo.0	5208.3	5154.9	5177.5	5147.9	*5145.5*
u_i_hihi.0	2990517.8	2957854.0	2952493.2	2938380.8	*2931630.9*
u_i_hilo.0	74030.3	73692.8	73639.8	73378.0	*73294.3*
u_i_lohi.0	103516.0	103865.6	102123.1	102050.6	*101737.7*
u_i_lolo.0	2575.4	2552.0	2548.9	2541.4	*2539.1*
u_s_hihi.0	4262337.5	4168795.8	4198779.5	4103500.3	*4098580.9*
u_s_hilo.0	97505.5	96180.8	96623.3	95787.4	*95764.9*
u_s_lohi.0	125717.0	123407.4	123236.9	122083.3	*121484.8*
u_s_lolo.0	3480.3	3450.5	3450.1	3433.5	*3425.9*

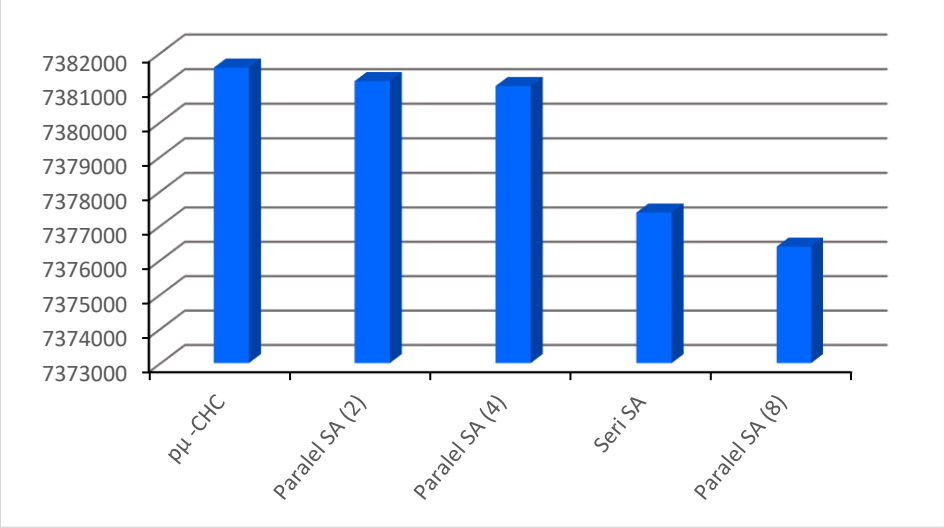
Çizelge 5.4. Farklı algoritmaların ortalama gecikme değerleri

BENCHMARKS	GA	CHC	PGA	TS	p-CHC	pμ-CHC	Seri SA
u_c_hihi.0	7699080.1	7681050.1	7606613.0	7458864.4	7481194.5	7394702.7	7400359.9
u_c_hilo.0	155300.1	155333.4	155036.5	153438.0	153924.0	153193.7	153334.7
u_c_lohi.0	252568.7	251868.3	249687.9	242385.3	243446.3	239706.2	240062.8
u_c_lolo.0	5248.6	5241.9	5224.7	5155.7	5181.6	5152.3	*5150.6*
u_i_hihi.0	3030564.2	3024904.9	3002119.3	2959029.3	2956905.7	2947896.4	2953919.5
u_i_hilo.0	74568.4	74375.9	74102.8	73734.8	73847.1	73531.4	*73526.3*
u_i_lohi.0	105048.1	104939.1	104078.6	103867.1	102677.3	102402.8	102415.3
u_i_lolo.0	2587.8	2582.2	2577.0	2559.9	2557.2	2547.1	*2547.1*
u_s_hihi.0	4347835.5	4320803.4	4282920.5	4181985.8	4239146.3	4123537.3	*4121642.4*
u_s_hilo.0	98026.1	98307.4	97585.5	96432.1	96750.3	96020.5	96046.4
u_s_lohi.0	126840.8	126238.0	126100.1	123600.5	123989.4	122744.4	*122332.4*
u_s_lolo.0	3516.5	3505.0	3487.2	3454.0	3472.2	3438.3	*3434.1*

5.2. Paralel benzetilmiş tavlama yaklaşımı için analiz sonuçları

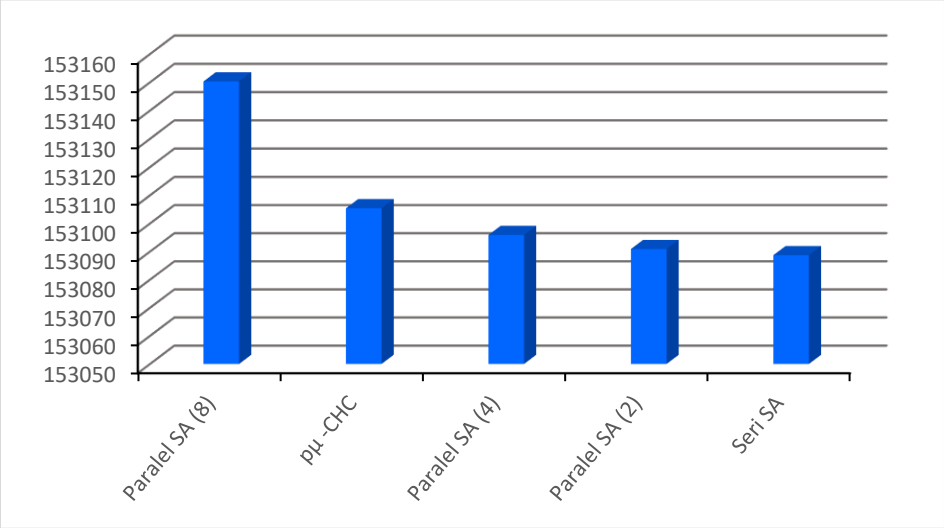
Benzetilmiş tavlama yaklaşımını kullanarak hazırladığımız seri ve paralel (üç farklı iş parçacığı (2, 4 ve 8) ile çalıştırdığımız) programın Braun modelinde yer alan her bir veri seti için ulaştığı sonuçlar ile literatürde bu model için en iyi gecikme değerlerinin rapor edildiği pμCHC yaklaşımının ulaştığı sonuçlar aşağıda yer alan grafiklerde gösterilmiştir.

Modeldeki "u_c_hihi" veri seti için en iyi gecikme değerleri Şekil 5.25'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde sekiz iş parçacığı ile çalıştırılan paralel SA'nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.07 oranında daha iyi olduğu görülmektedir.



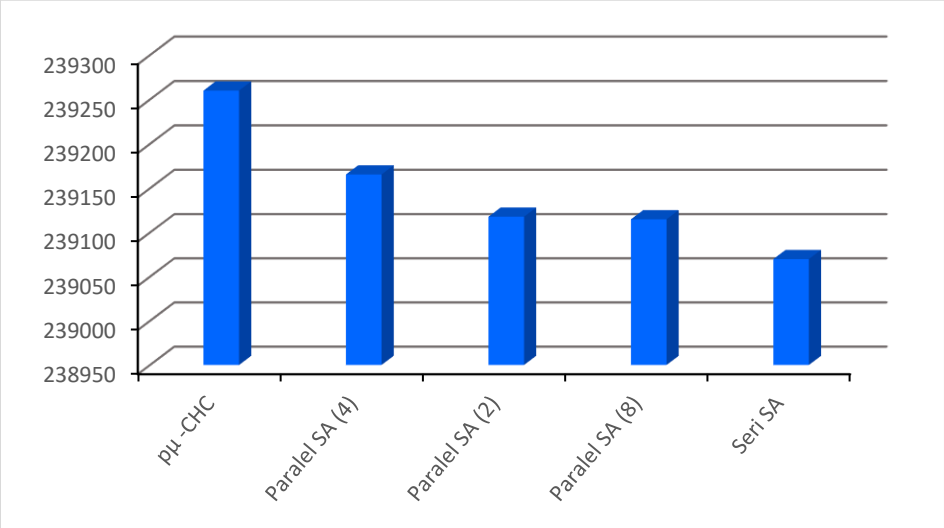
Şekil 5.25. u_c_hihi için en iyi gecikme değerleri

Modeldeki "u_c_hilo" veri seti için en iyi gecikme değerleri Şekil 5.26'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde iki iş parçacığı ile çalıştırılan paralel SA'nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0,01 oranında daha iyi olduğu görülmektedir.



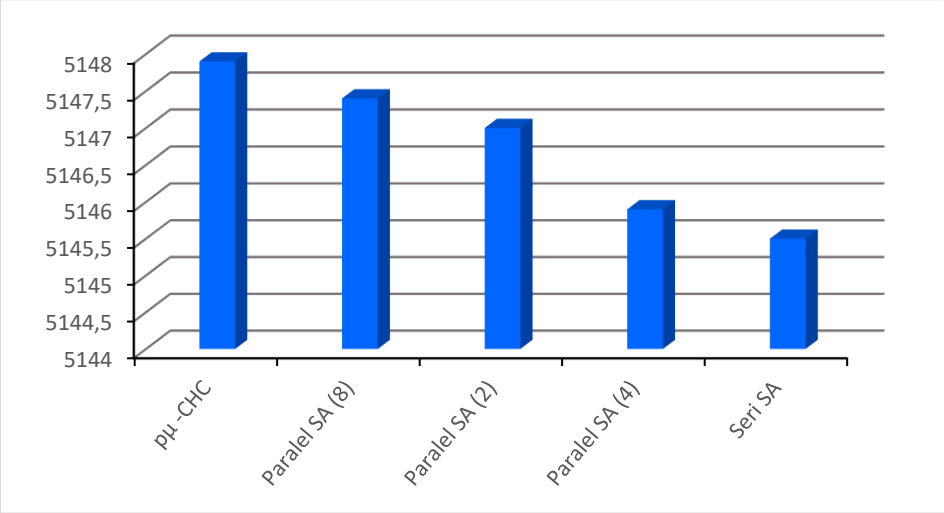
Şekil 5.26. u_c_hilo için en iyi gecikme değerleri

Modeldeki "u_c_lohi" veri seti için en iyi gecikme değerleri Şekil 5.27'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde seri SA'nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.07 oranında daha iyi olduğu görülmektedir.



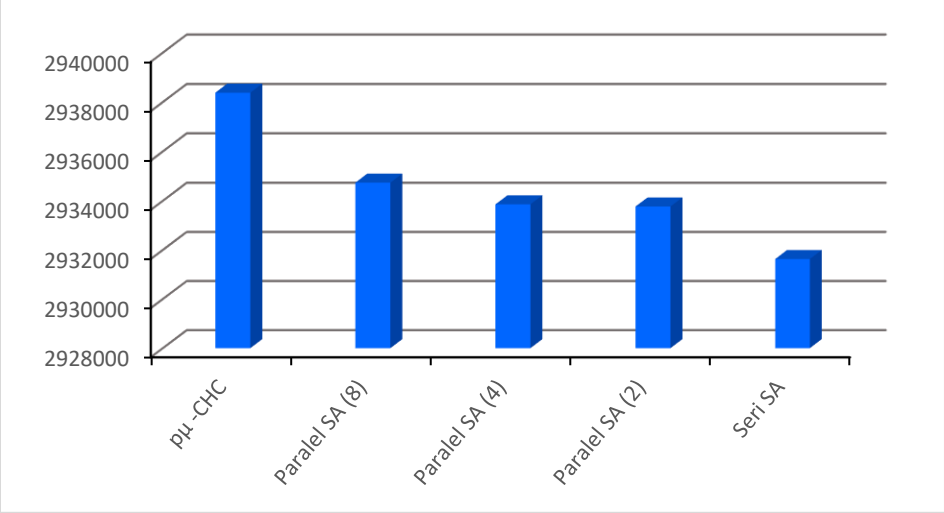
Şekil 5.27. u_c_lohi için en iyi gecikme değerleri

Modeldeki “u_c_lolo” veri seti için en iyi gecikme değerleri Şekil 5.28’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde seri SA’nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.04 oranında daha iyi olduğu görülmektedir.



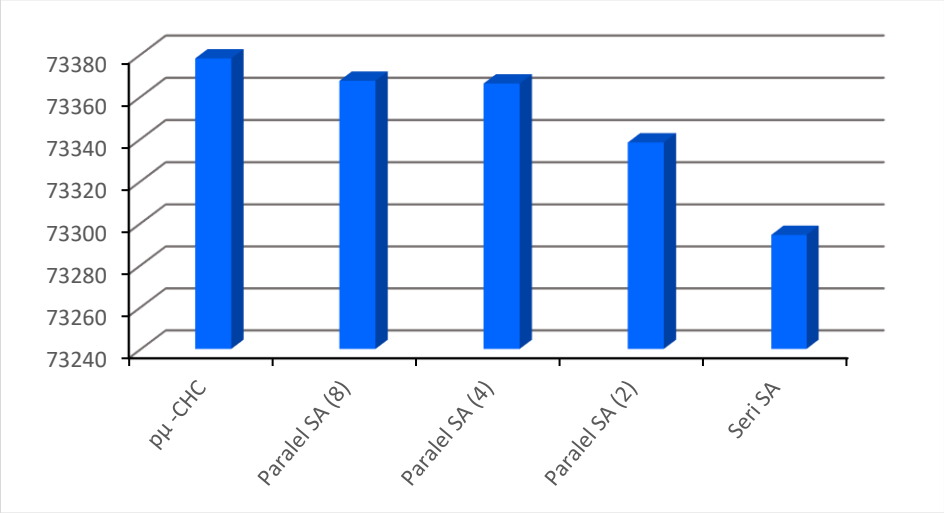
Şekil 5.28. u_c_lolo için en iyi gecikme değerleri

Modeldeki “u_i_hihi” veri seti için en iyi gecikme değerleri Şekil 5.29’da gösterilen grafikte yer almaktadır. Grafik incelendiğinde seri SA’nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.22 oranında daha iyi olduğu görülmektedir.



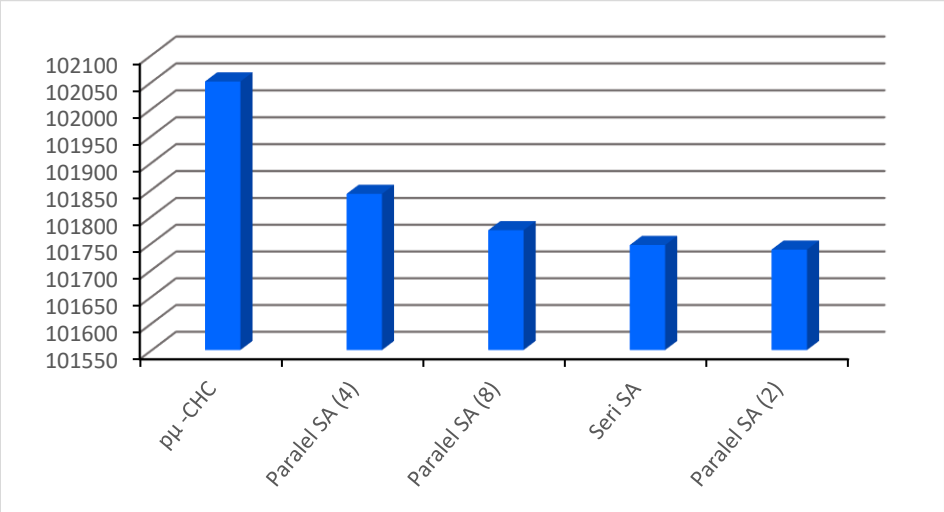
Şekil 5.29. u_i_hihi için en iyi gecikme değerleri

Modeldeki “u_i_hilo” veri seti için en iyi gecikme değerleri Şekil 5.30’da gösterilen grafikte yer almaktadır. Grafik incelendiğinde seri SA’nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.30 oranında daha iyi olduğu görülmektedir.



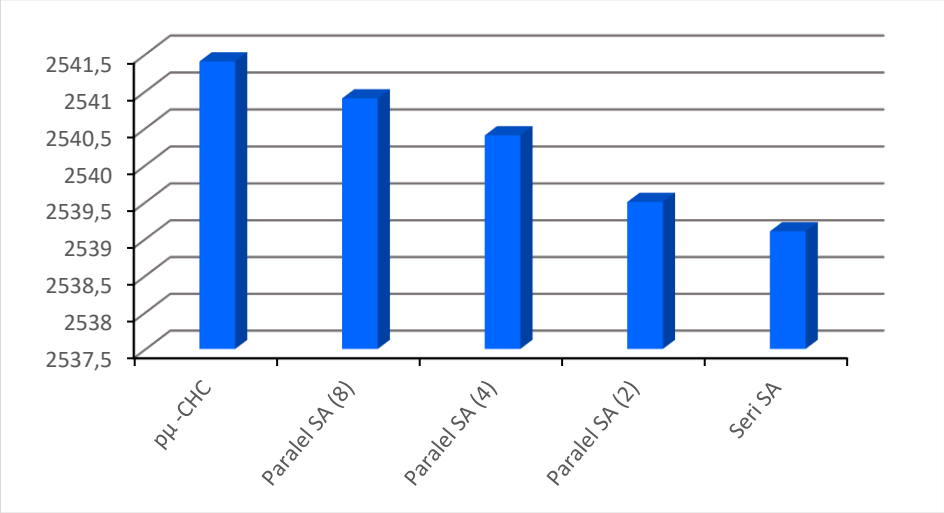
Şekil 5.30. u_i_hilo için en iyi gecikme değerleri

Modeldeki “u_i_lohi” veri seti için en iyi gecikme değerleri Şekil 5.31’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde iki iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.11 oranında daha iyi olduğu görülmektedir.



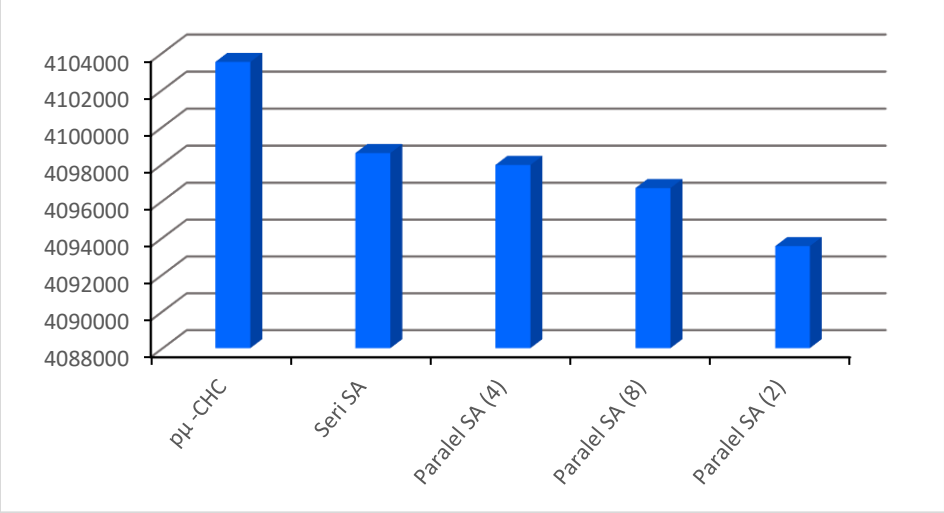
Şekil 5.31. u_i_lohi için en iyi gecikme değerleri

Modeldeki “u_i_lolo” veri seti için en iyi gecikme değerleri Şekil 5.32’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde seri SA’nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.09 oranında daha iyi olduğu görülmektedir.



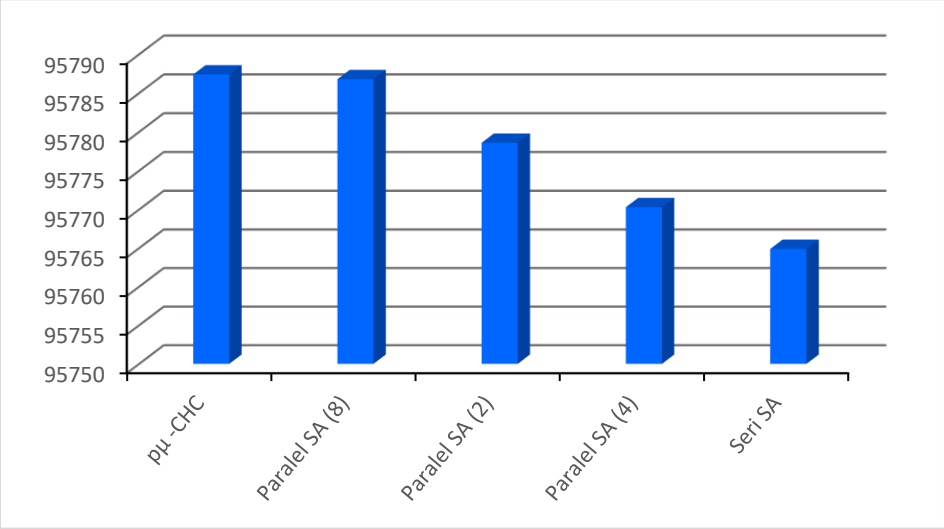
Şekil 5.32. u_i_lolo için en iyi gecikme değerleri

Modeldeki “u_s_hihi” veri seti için en iyi gecikme değerleri Şekil 5.35’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde iki iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.24 oranında daha iyi olduğu görülmektedir.



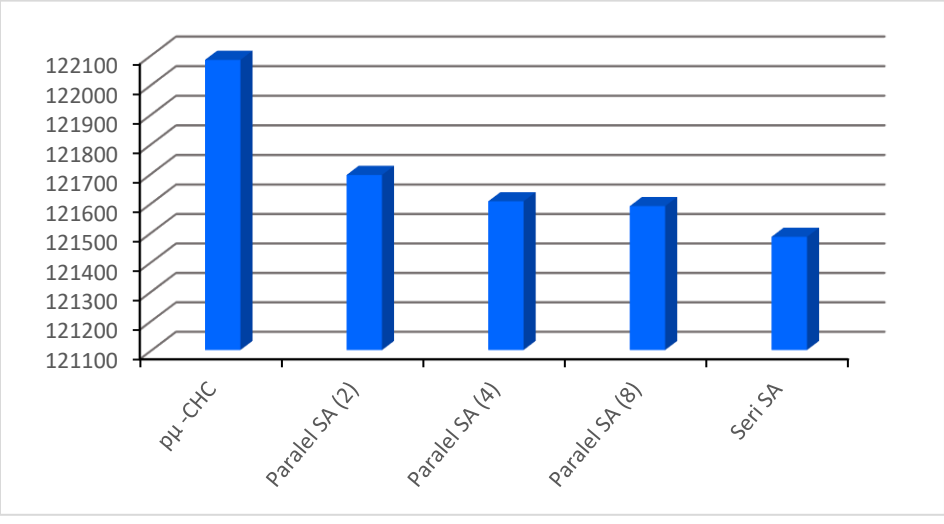
Şekil 5.33. u_s_hihi için en iyi gecikme değerleri

Modeldeki “u_s_hilo” veri seti için en iyi gecikme değerleri Şekil 5.34’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde seri SA’nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.02 oranında daha iyi olduğu görülmektedir.



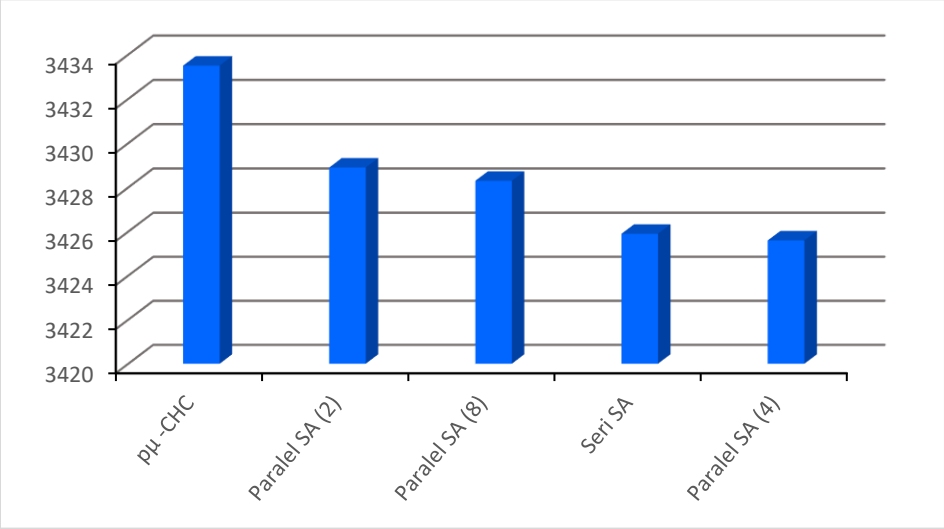
Şekil 5.35. u_s_hilo için en iyi gecikme değerleri

Modeldeki “u_s_lohi” veri seti için en iyi gecikme değerleri Şekil 5.35’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde seri SA’nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.49 oranında daha iyi olduğu görülmektedir.



Şekil 5.35. u_s_lohi için en iyi gecikme değerleri

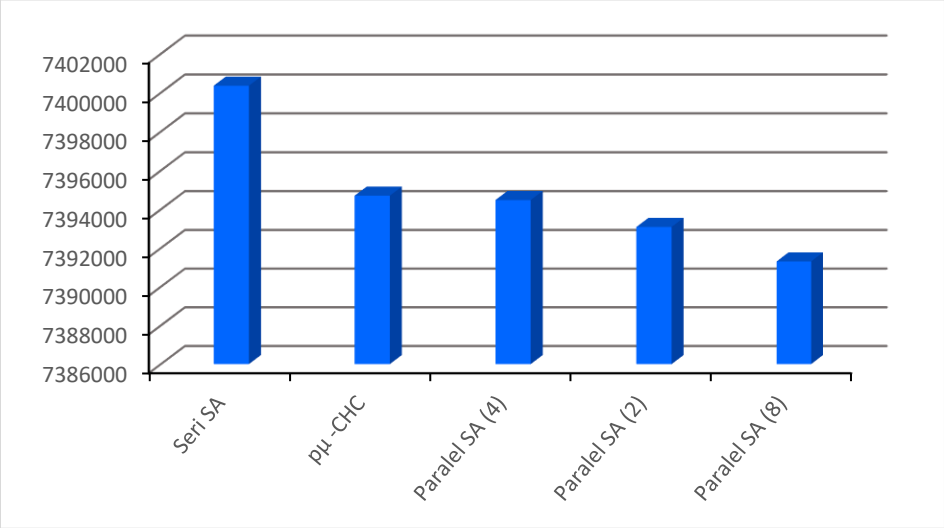
Modeldeki “u_s_lolo” veri seti için en iyi gecikme değerleri Şekil 5.36’da gösterilen grafikte yer almaktadır. Grafik incelendiğinde dört iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.23 oranında daha iyi olduğu görülmektedir.



Şekil 5.36. u_s_lolo için en iyi gecikme değerleri

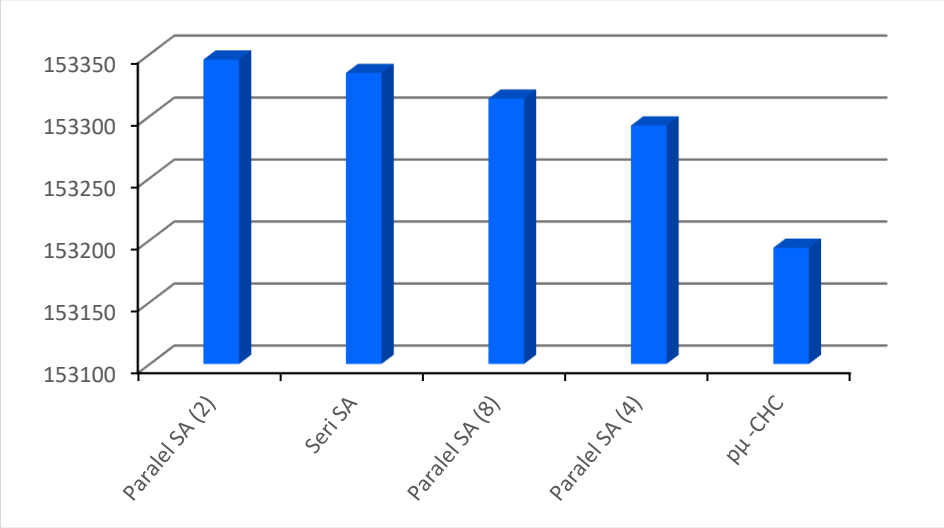
Benzer şekilde programın Braun modelinde yer alan her bir veri seti için ulaştığı ortalama gecikme değerleri ile literatürde bu model için en iyi ortalama gecikme değerlerinin rapor edildiği pμCHC yaklaşımının ulaştığı sonuçlar aşağıda yer alan grafiklerde gösterilmiştir.

Modeldeki “u_c_hihi” veri seti için ortalama gecikme değerleri Şekil 5.37’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde sekiz iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.04 oranında daha iyi olduğu görülmektedir.



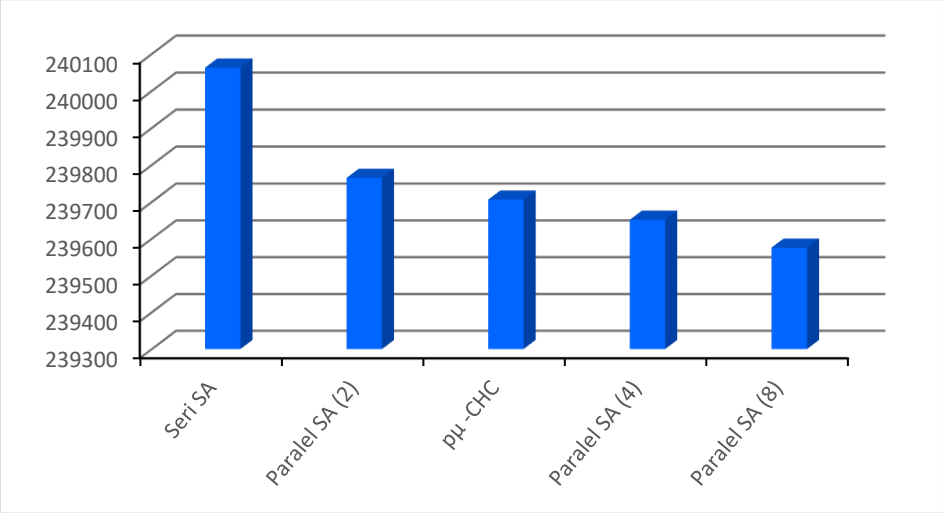
Şekil 5.37. u_c_hihi için en iyi ortalama gecikme değerleri

Modeldeki "u_c_hilo" veri seti için ortalama gecikme değerleri Şekil 5.38'de gösterilen grafikte yer almaktadır. Grafik incelendiğinde seri ve paralel SA'nın belirtilen veri seti için pμCHC yaklaşımına göre en iyi ortalama gecikme değerinden daha düşük kalitede sonuçlara ulaştığı görülmektedir.



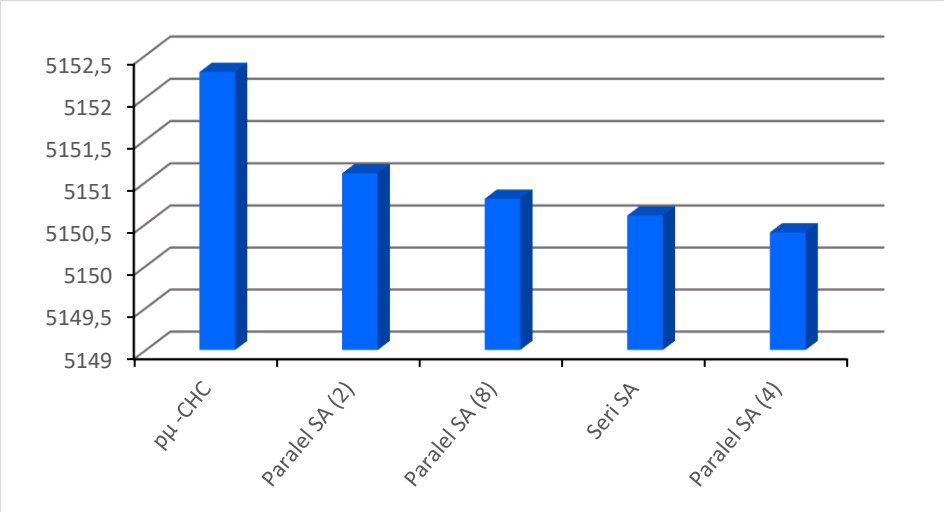
Şekil 5.38. u_c_hilo için en iyi ortalama gecikme değerleri

Modeldeki “u_c_lohi” veri seti için ortalama gecikme değerleri Şekil 5.39’da gösterilen grafikte yer almaktadır. Grafik incelendiğinde sekiz iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.05 oranında daha iyi olduğu görülmektedir.



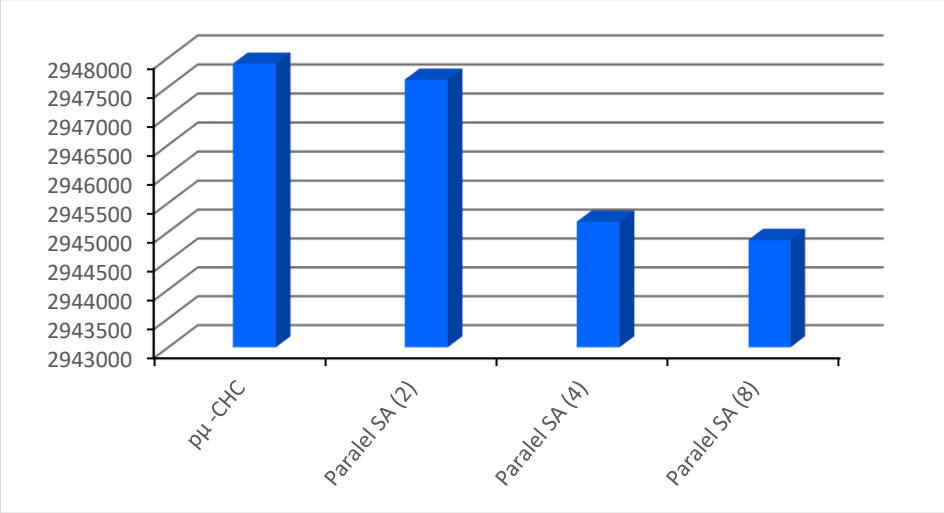
Şekil 5.39. u_c_lohi için en iyi ortalama gecikme değerleri

Modeldeki “u_c_lolo” veri seti için ortalama gecikme değerleri Şekil 5.40’da gösterilen grafikte yer almaktadır. Grafik incelendiğinde dört iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.03 oranında daha iyi olduğu görülmektedir.



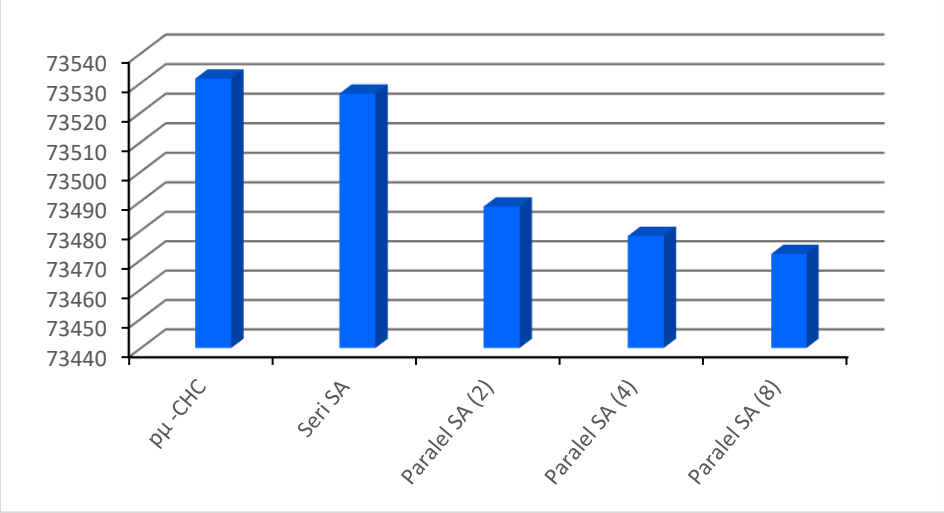
Şekil 5.40. u_c_lolo için en iyi ortalama gecikme değerleri

Modeldeki “u_i_hihi” veri seti için ortalama gecikme değerleri Şekil 5.41’da gösterilen grafikte yer almaktadır. Grafik incelendiğinde sekiz iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.10 oranında daha iyi olduğu görülmektedir.



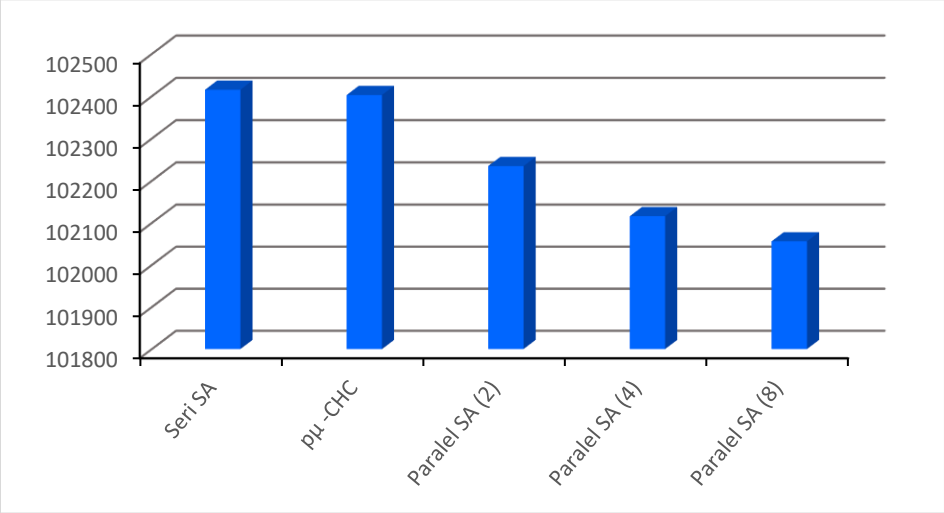
Şekil 5.41. u_i_hihi için en iyi ortalama gecikme değerleri

Modeldeki “u_i_hilo” veri seti için ortalama gecikme değerleri Şekil 5.42’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde sekiz iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.08 oranında daha iyi olduğu görülmektedir.



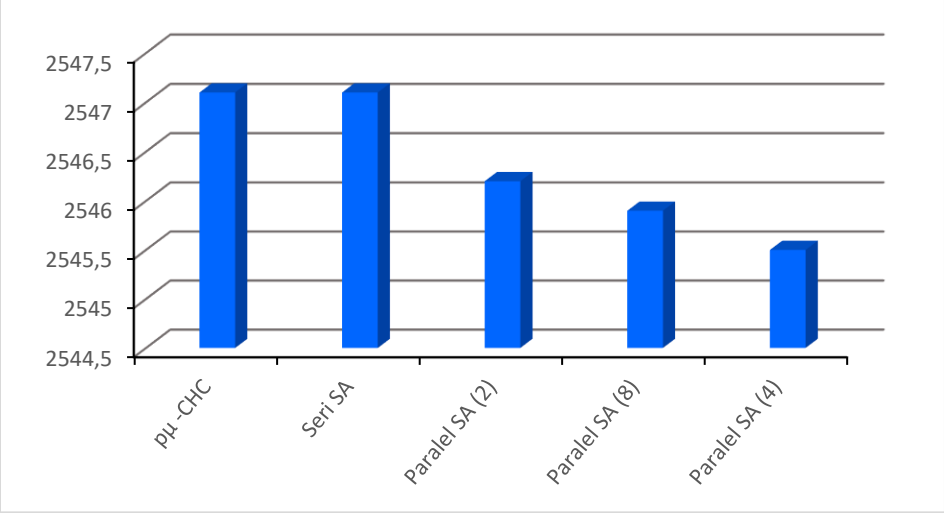
Şekil 5.42. u_i_hilo için en iyi ortalama gecikme değerleri

Modeldeki “u_i_lohi” veri seti için ortalama gecikme değerleri Şekil 5.43’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde sekiz iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.33 oranında daha iyi olduğu görülmektedir.



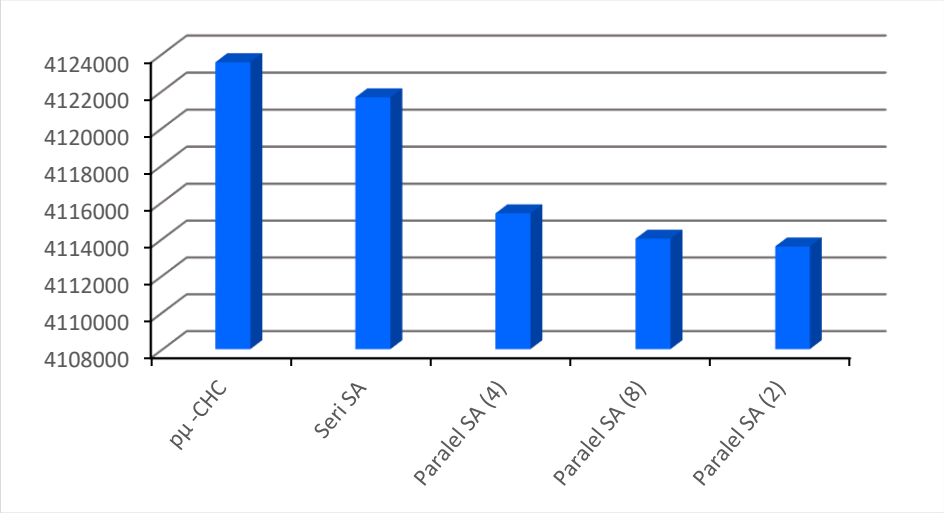
Şekil 5.43. u_i_lohi için en iyi ortalama gecikme değerleri

Modeldeki “u_i_lolo” veri seti için ortalama gecikme değerleri Şekil 5.44’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde sekiz iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.06 oranında daha iyi olduğu görülmektedir.



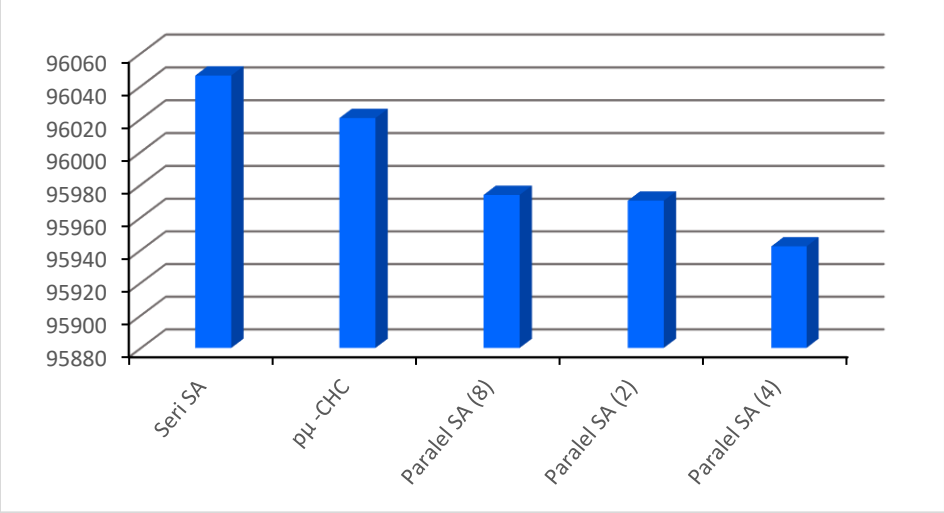
Şekil 5.44. u_i_lolo için en iyi ortalama gecikme değerleri

Modeldeki “u_s_hihi” veri seti için ortalama gecikme değerleri Şekil 5.45’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde iki iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.24 oranında daha iyi olduğu görülmektedir.



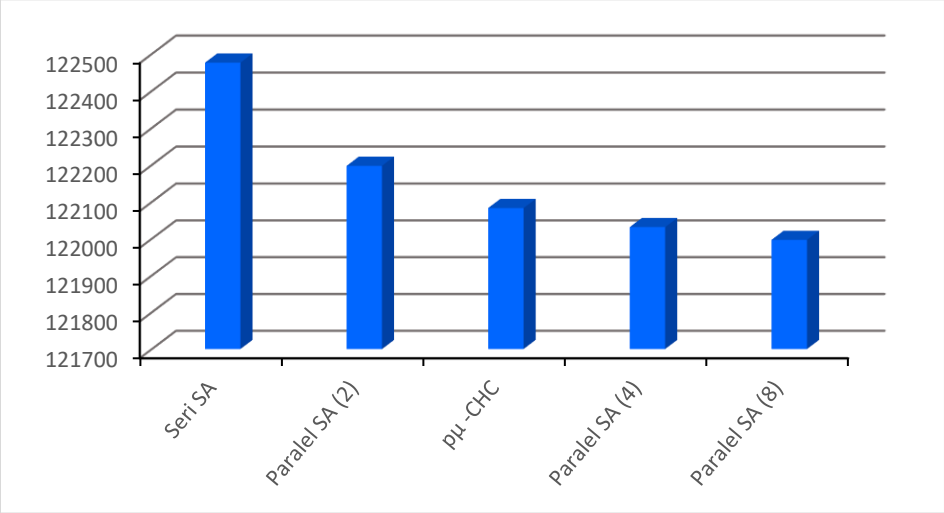
Şekil 5.45. u_s_hihi için en iyi ortalama gecikme değerleri

Modeldeki “u_s_hilo” veri seti için ortalama gecikme değerleri Şekil 5.46’da gösterilen grafikte yer almaktadır. Grafik incelendiğinde dört iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.08 oranında daha iyi olduğu görülmektedir.



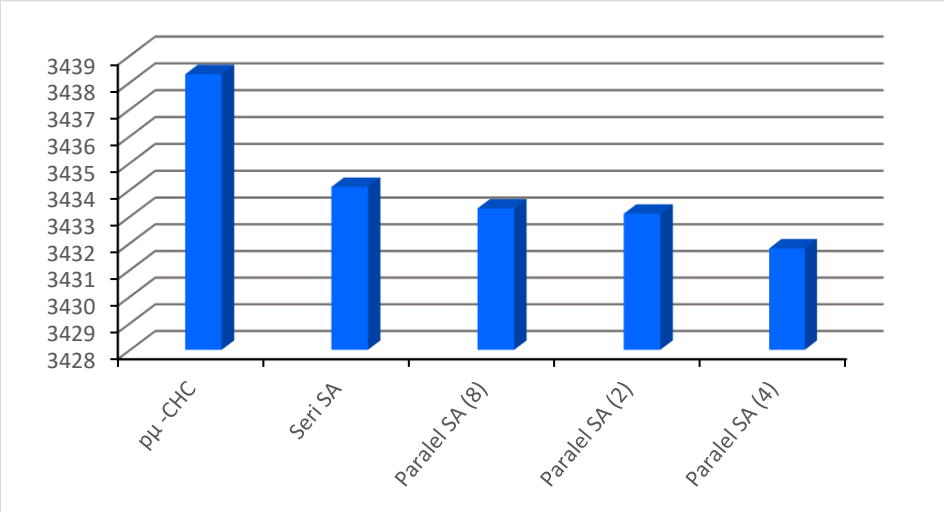
Şekil 5.46. u_s_hilo için en iyi ortalama gecikme değerleri

Modeldeki “u_s_lohi” veri seti için ortalama gecikme değerleri Şekil 5.47’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde sekiz iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.07 oranında daha iyi olduğu görülmektedir.



Şekil 5.47. u_s_lohi için en iyi ortalama gecikme değerleri

Modeldeki “u_s_lolo” veri seti için ortalama gecikme değerleri Şekil 5.48’de gösterilen grafikte yer almaktadır. Grafik incelendiğinde dört iş parçacığı ile çalıştırılan paralel SA’nın belirtilen veri seti için en iyi ortalama gecikme değerine ulaştığı ve pμCHC yaklaşımına göre %0.18 oranında daha iyi olduğu görülmektedir.



Şekil 5.48. u_s_lolo için en iyi ortalama gecikme değerleri

Geliştirilen bilgisayar programının seri ve paralel versiyonu ile Braun modeline ait her bir veri seti için ulaşılan süre, en iyi gecikme değerleri, ortalama gecikme değerleri ve literatürde aynı model için rapor edilen pμCHC yaklaşımına ait en iyi gecikme değerleri, ortalama gecikme değerleri sırasıyla Çizelge 5.5, Çizelge 5.6 ve Çizelge 5.7'de verilmiştir. Çizelgelerde seri ve paralel versiyonlarda ulaşılan ve literatürde şu ana kadar rapor edilen en iyi gecikme değerlerinden daha iyi sonuçlar kırmızı renkle, en iyi ortalama gecikme değerlerinden daha iyi sonuçlar mavi renkle gösterilmiştir.

Çizelge 5.5. En iyi ve ortalama gecikme değerleri 1

BENCHMARK	pμ-CHC (En İyi Gecikme, Ortalama Gecikme)	Seri SA (Süre, En İyi Gecikme, Ortalama Gecikme)	Paralel SA (2 İş Parçacığı) (Süre, En İyi Gecikme, Ortalama Gecikme)
u_c_hihi.0	7381570.0, 7394702.7	90 sn, *7377378.2*, 7400359.9	90 sn, *7381181.3*, *7393090.7*
u_c_hilo.0	153105.4, 153193.7	86sn, *153088.7*, 153334,7	90 sn, *153090.9*, 153345.5
u_c_lohi.0	239260.0, 239706.2	88 sn, *239069.9*, 240062.8	87 sn, *239117.9*, 239765.0
u_c_lolo.0	5147.9, 5152.3	90 sn, *5145.5*, *5150.6*	90 sn, *5147.0*, *5151.1*
u_i_hihi.0	2938380.8, 2947896.4	89 sn, *2931630.9*, 2953919.5	90 sn, *2933767.7*, *2947620.8*
u_i_hilo.0	73378.0, 73531.4	92 sn, *73294.3*, *73526.3*	90 sn, *73338.2*, *73488.1*
u_i_lohi.0	102050.6, 102402.8	90 sn, *101746.3*, 102415.3	90 sn, *101737.7*, *102234.9*
u_i_lolo.0	2541.4, 2547.1	90 sn, *2539.1*, *2547.1*	90 sn, *2539.5*, *2546.2*
u_s_hihi.0	4103500.3, 4123537.3	90 sn, *4098580.9*, *4121642.4*	90 sn, *4093541.4*, *4113577.9*
u_s_hilo.0	95787.4, 96020.5	90 sn, *95764.9*, 96046.4	90 sn, *95778.6*, *95970.2*
u_s_lohi.0	122083.3, 122744.4	88 sn, *121484.8*, *122476.4*	90 sn, *121694.5*, *122197.5*
u_s_lolo.0	3433.5, 3438.3	85 sn, *3425.9*, *3434.1*	90 sn, *3428.9*, *3433.1*

Çizelge 5.6. En iyi ve ortalama gecikme değerleri 2

BENCHMARK	pμ-CHC (En İyi Gecikme, Ortalama Gecikme)	Seri SA (Süre, En İyi Gecikme, Ortalama Gecikme)	Paralel SA (4 İş Parçacığı) (Süre, En İyi Gecikme, Ortalama Gecikme)
u_c_hihi.0	7381570.0, 7394702.7	90 sn, *7377378.2*, 7400359.9	79 sn, *7381036.3*, *7394480.0*
u_c_hilo.0	153105.4, 153193.7	86sn, *153088.7*, 153334,7	90 sn, *153095.9*, 153292.3
u_c_lohi.0	239260.0, 239706.2	88 sn, *239069.9*, 240062.8	87 sn, *239165.5*, *239651.5*
u_c_lolo.0	5147.9, 5152.3	90 sn, *5145.5*, *5150.6*	90 sn, *5145.9*, *5150.4*
u_i_hihi.0	2938380.8, 2947896.4	89 sn, *2931630.9*, 2953919.5	81 sn, *2933858.1*, *2945171.5*
u_i_hilo.0	73378.0, 73531.4	92 sn, *73294.3*, *73526.3*	90 sn, *73366.1*, *73478.1*
u_i_lohi.0	102050.6, 102402.8	90 sn, *101746.3*, 102415.3	90 sn, *101841.8*, *102116.1*
u_i_lolo.0	2541.4, 2547.1	90 sn, *2539.1*, *2547.1*	90 sn, *2540.4*, *2545.5*
u_s_hihi.0	4103500.3, 4123537.3	90 sn, *4098580.9*, *4121642.4*	87 sn, *4097940.9*, *4115372.3*
u_s_hilo.0	95787.4, 96020.5	90 sn, *95764.9*, 96046.4	87 sn, *95770.3*, *95942.2*
u_s_lohi.0	122083.3, 122744.4	88 sn, *121484.8*, *122476.4*	90 sn, *121605.4*, *122031.5*
u_s_lolo.0	3433.5, 3438.3	85 sn, *3425.9*, *3434.1*	90 sn, *3425.6*, *3431.8*

Çizelge 5.7. En iyi ve ortalama gecikme değerleri 3

BENCHMARK	pμ-CHC (En İyi Gecikme, Ortalama Gecikme)	Seri SA (Süre, En İyi Gecikme, Ortalama Gecikme)	Paralel SA (8 İş Parçacığı) (Süre, En İyi Gecikme, Ortalama Gecikme)
u_c_hihi.0	7381570.0, 7394702.7	90 sn, *7377378.2*, 7400359.9	90 sn, *7376385.2*, *7391308.5*
u_c_hilo.0	153105.4, 153193.7	86sn, *153088.7*, 153334,7	85 sn, 153150.3, 153314.0
u_c_lohi.0	239260.0, 239706.2	88 sn, *239069.9*, 240062.8	90 sn, *239114.9*, *239575.6*
u_c_lolo.0	5147.9, 5152.3	90 sn, *5145.5*, *5150.6*	90 sn, *5147.4*, *5150.8*
u_i_hihi.0	2938380.8, 2947896.4	89 sn, *2931630.9*, 2953919.5	85 sn, *2934715.8*, *2944859.4*
u_i_hilo.0	73378.0, 73531.4	92 sn, *73294.3*, *73526.3*	90 sn, *73367.4*, *73472.0*
u_i_lohi.0	102050.6, 102402.8	90 sn, *101746.3*, 102415.3	90 sn, *101773.9*, *102056.1*
u_i_lolo.0	2541.4, 2547.1	90 sn, *2539.1*, *2547.1*	90 sn, *2540.9*, *2545.9*
u_s_hihi.0	4103500.3, 4123537.3	90 sn, *4098580.9*, *4121642.4*	87 sn, *4096688.9*, *4113997.2*
u_s_hilo.0	95787.4, 96020.5	90 sn, *95764.9*, 96046.4	90 sn, *95786.8*, *95973.7*
u_s_lohi.0	122083.3, 122744.4	88 sn, *121484.8*, *122476.4*	90 sn, *121589.1*, *121996.8*
u_s_lolo.0	3433.5, 3438.3	85 sn, *3425.9*, *3434.1*	88 sn, *3428.3*, *3433.3*

5.3. Seri ve Paralel SA ile pμCHC ve LB Arasındaki Fark

Seri ve paralel SA yaklaşımında ulaşılan gecikme değerlerinin pμCHC yaklaşımının ulaştığı değerler (literatürde rapor edilen en iyi gecikme değerleri) ile arasındaki fark/boşluk (GAP) aşağıda belirtilen şekilde hesaplanarak çözümün kalitesine karar verilmiştir.

$$GAP = \frac{p\mu CHC - SA}{p\mu CHC} * 100$$

Seri SA ile ulaşılan gecikme değerlerinin pμCHC yaklaşımının ulaştığı değerler ile arasındaki fark/boşluk (GAP) Çizelge 5.8'de yer almaktadır. Bu sonuçlar seri SA yaklaşımının pμCHC yaklaşımına göre en en az %0.01 oranında bir farkla daha iyi gecikme değerlerine ulaştığını göstermektedir.

Çizelge 5.8. Seri SA ile pμCHC yaklaşımı arasındaki fark/boşluk

BENCHMARK	pμCHC	Seri SA	GAP (%)
u_c_hihi.0	7381570.0	7377378.2	0.056
u_c_hilo.0	153105.4	153088.7	0.010
u_c_lohi.0	239260.0	239069.9	0.079
u_c_lolo.0	5147.9	5145.5	0.046
u_i_hihi.0	2938380.8	2931630.9	0.230
u_i_hilo.0	73378.0	73294.3	0.114
u_i_lohi.0	102050.6	101746.3	0.299
u_i_lolo.0	2541.4	2539.1	0.090
u_s_hihi.0	4103500.3	4098580.9	0.120
u_s_hilo.0	95787.4	95764.9	0.023
u_s_lohi.0	122083.3	121484.8	0.492
u_s_lolo.0	3433.5	3425.9	0.221

İki iş parçacığı ile çalıştırılan paralel SA ile ulaşılan gecikme değerlerinin pμCHC yaklaşımının ulaştığı değerler ile arasındaki fark/boşluk (GAP) Çizelge 5.9'da yer almaktadır. Bu sonuçlar iki iş parçacığı ile çalıştırılan paralel SA yaklaşımının pμCHC yaklaşımına göre en en az %0.005 oranında bir farkla daha iyi gecikme değerlerine ulaştığını göstermektedir.

Çizelge 5.9. Paralel SA (2) ile pμCHC yaklaşımı arasındaki fark/boşluk

BENCHMARK	pμCHC	Paralel SA (2)	GAP (%)
u_c_hihi.0	7381570.0	7381181.3	0.005
u_c_hilo.0	153105.4	153090.9	0.009
u_c_lohi.0	239260.0	239117.9	0.059
u_c_lolo.0	5147.9	5147.0	0.017
u_i_hihi.0	2938380.8	2933767.7	0.156
u_i_hilo.0	73378.0	73338.2	0.054
u_i_lohi.0	102050.6	101737.7	0.306
u_i_lolo.0	2541.4	2539.5	0.074
u_s_hihi.0	4103500.3	4093541.4	0.242
u_s_hilo.0	95787.4	95778.6	0.009
u_s_lohi.0	122083.3	121694.5	0.318
u_s_lolo.0	3433.5	3428.9	0.133

Dört iş parçacığı ile çalıştırılan paralel SA ile ulaşılan gecikme değerlerinin pμCHC yaklaşımının ulaştığı değerler ile arasındaki fark/boşluk (GAP) Çizelge 5.10'da yer almaktadır. Bu sonuçlar dört iş parçacığı ile çalıştırılan paralel SA yaklaşımının pμCHC yaklaşımına göre en en az %0.006 oranında bir farkla daha iyi gecikme değerlerine ulaştığını göstermektedir.

Çizelge 5.10. Paralel SA (4) ile pμCHC yaklaşımı arasındaki fark/boşluk

BENCHMARK	pμCHC	Paralel SA (4)	GAP (%)
u_c_hihi.0	7381570.0	7381036.3	0.007
u_c_hilo.0	153105.4	153095.9	0.006
u_c_lohi.0	239260.0	239165.5	0.039
u_c_lolo.0	5147.9	5145.9	0.038
u_i_hihi.0	2938380.8	2933858.1	0.153
u_i_hilo.0	73378.0	73366.1	0.016
u_i_lohi.0	102050.6	101841.8	0.204
u_i_lolo.0	2541.4	2540.4	0.039
u_s_hihi.0	4103500.3	4097940.9	0.135
u_s_hilo.0	95787.4	95770.3	0.017
u_s_lohi.0	122083.3	121605.4	0.391
u_s_lolo.0	3433.5	3425.6	0.230

Sekiz iş parçacığı ile çalıştırılan paralel SA ile ulaşılan gecikme değerlerinin pμCHC yaklaşımının ulaştığı değerler ile arasındaki fark/boşluk (GAP) Çizelge 5.11'de yer almaktadır. Bu sonuçlar sekiz iş parçacığı ile çalıştırılan paralel SA yaklaşımının pμCHC yaklaşımına göre (pμCHC yaklaşımından daha iyi sonuçlara ulaşılan veri setleri içerisinde) en en az %0.02 oranında bir farkla daha iyi gecikme değerlerine ulaştığını göstermektedir.

Çizelge 5.11. Paralel SA (8) ile pμCHC yaklaşımı arasındaki fark/boşluk

BENCHMARK	pμCHC	Paralel SA (8)	GAP (%)
u_c_hihi.0	7381570.0	7376385.2	0.05
u_c_hilo.0	153105.4	153150.3	-0.02
u_c_lohi.0	239260.0	239114.9	0.07
u_c_lolo.0	5147.9	5147.4	0.04
u_i_hihi.0	2938380.8	2934715.8	0.23
u_i_hilo.0	73378.0	73367.4	0.11
u_i_lohi.0	102050.6	101773.9	0.29
u_i_lolo.0	2541.4	2540.9	0.09
u_s_hihi.0	4103500.3	4096688.9	0.12
u_s_hilo.0	95787.4	95786.8	0.02
u_s_lohi.0	122083.3	121589.1	0.49
u_s_lolo.0	3433.5	3428.3	0.22

Braun modelindeki veri setleri için (Nesmachnow *et al.* 2010)'da bahsedilen dinamik programlama ile bulunmuş teorik optimum değerlere alt sınır (*lower bound*) veya LB denilmektedir. Seri ve paralel SA yaklaşımında ulaşılan gecikme değerleri ile (Anonymous 2012)'den alınan LB değerleri arasındaki fark/boşluk (GAP) aşağıda belirtilen şekilde hesaplanmıştır.

$$GAP = \frac{SA - LB}{LB} * 100$$

Seri SA yaklaşımında ulaştığımız gecikme değerlerinin LB (Lower Bound) ile arasındaki fark/boşluk (GAP) Çizelge 5.12'de yer almaktadır. Bu sonuçlar iki iş parçacığı ile çalıştırılan paralel SA yaklaşımının LB ile arasında en fazla %0,86 oranında bir fark olduğunu göstermektedir.

Çizelge 5.12. Seri SA ile LB arasındaki fark/boşluk

BENCHMARK	LB	Seri SA	GAP (%)
u_c_hihi.0	7346524.2	7377378.2	0.41
u_c_hilo.0	152700.4	153088.7	0.25
u_c_lohi.0	238138.1	239069.9	0.39
u_c_lolo.0	5132.8	5145.5	0.24
u_i_hihi.0	2909326.6	2931630.9	0.76
u_i_hilo.0	73057.9	73294.3	0.32
u_i_lohi.0	101063.4	101746.3	0.67
u_i_lolo.0	2529.0	2539.1	0.39
u_s_hihi.0	4063564.7	4098580.9	0.86
u_s_hilo.0	95419.0	95764.9	0.36
u_s_lohi.0	120452.3	121484.8	0.85
u_s_lolo.0	3414.8	3425.9	0.32

İki iş parçacığı ile çalıştırılan paralel SA yaklaşımında ulaştığımız gecikme değerlerinin LB ile arasındaki fark/boşluk (GAP) Çizelge 5.13'de yer almaktadır. Bu sonuçlar iki iş parçacığı ile çalıştırılan paralel SA yaklaşımının LB ile arasında en fazla %1.02 oranında bir fark olduğunu göstermektedir.

Çizelge 5.13. Paralel SA (2) ile LB arasındaki fark/boşluk

BENCHMARK	LB	Paralel SA (2)	GAP (%)
u_c_hihi.0	7346524.2	7381181.3	0,46
u_c_hilo.0	152700.4	153090.9	0,25
u_c_lohi.0	238138.1	239117.9	0,40
u_c_lolo.0	5132.8	5147.0	0,27
u_i_hihi.0	2909326.6	2933767.7	0,83
u_i_hilo.0	73057.9	73338.2	0,38
u_i_lohi.0	101063.4	101737.7	0,66
u_i_lolo.0	2529.0	2539.5	0,41
u_s_hihi.0	4063564.7	4093541.4	0,73
u_s_hilo.0	95419.0	95778.6	0,37
u_s_lohi.0	120452.3	121694.5	1,02
u_s_lolo.0	3414.8	3428.9	0,41

Dört iş parçacığı ile çalıştırılan paralel SA yaklaşımında ulaştığımız gecikme değerlerinin LB ile arasındaki fark/boşluk (GAP) Çizelge 5.14'de yer almaktadır. Bu sonuçlar dört

iş parçacığı ile çalıştırılan paralel SA yaklaşımının LB ile arasında en fazla %0.94 oranında bir fark olduğunu göstermektedir.

Çizelge 5.14. Paralel SA (4) ile LB arasındaki fark/boşluk

BENCHMARK	LB	Paralel SA (4)	GAP (%)
u_c_hihi.0	7346524.2	7381036.3	0,46
u_c_hilo.0	152700.4	153095.9	0,25
u_c_lohi.0	238138.1	239165.5	0,42
u_c_lolo.0	5132.8	5145.9	0,25
u_i_hihi.0	2909326.6	2933858.1	0,83
u_i_hilo.0	73057.9	73366.1	0,42
u_i_lohi.0	101063.4	101841.8	0,76
u_i_lolo.0	2529.0	2540.4	0,44
u_s_hihi.0	4063564.7	4097940.9	0,83
u_s_hilo.0	95419.0	95770.3	0,36
u_s_lohi.0	120452.3	121605.4	0,94
u_s_lolo.0	3414.8	3425.6	0,31

Sekiz iş parçacığı ile çalıştırılan SA yaklaşımında ulaştığımız gecikme değerlerinin LB ile arasındaki fark/boşluk (GAP) Çizelge 5.15'de yer almaktadır. Bu sonuçlar iki iş parçacığı ile çalıştırılan paralel SA yaklaşımının LB ile arasında en fazla %0.93 oranında bir fark olduğunu göstermektedir.

Çizelge 5.15. Paralel SA (8) ile LB arasındaki fark/boşluk

BENCHMARK	LB	Paralel SA (8)	GAP (%)
u_c_hihi.0	7346524.2	7376385.2	0,40
u_c_hilo.0	152700.4	153150.3	0,29
u_c_lohi.0	238138.1	239114.9	0,40
u_c_lolo.0	5132.8	5147.4	0,28
u_i_hihi.0	2909326.6	2934715.8	0,86
u_i_hilo.0	73057.9	73367.4	0,42
u_i_lohi.0	101063.4	101773.9	0,69
u_i_lolo.0	2529.0	2540.9	0,46
u_s_hihi.0	4063564.7	4096688.9	0,80
u_s_hilo.0	95419.0	95786.8	0,38
u_s_lohi.0	120452.3	121589.1	0,93
u_s_lolo.0	3414.8	3428.3	0,39

5.4. Parametreler

Seri ve paralel benzetilmiş tavlama yaklaşımı ile oluşturulan bilgisayar programında kullanılan arama parametreleri Çizelge 5.16'da yer almaktadır.

Çizelge 5.16. Parametreler

BENCHMARK	Seri SA	Paralel SA (2)	Paralel SA (4)	Paralel SA (8)
	$\alpha, \varepsilon, Beta$	$\alpha, \varepsilon, Beta$	$\alpha, \varepsilon, Beta$	$\alpha, \varepsilon, Beta$
u_c_hihi.0 **u_i_hihi.0** **u_s_hihi.0**	0.996, 0.01, 3.75	0.996, 0.01, 3	0.996, 0.01, 2	0.996, 0.01, 1.50
u_c_hilo.0 **u_i_hilo.0** **u_s_hilo.0** **u_c_lohi.0** **u_i_lohi.0** **u_s_lohi.0**	0.997, 0.01, 3.75	0.997, 0.01, 3	0.997, 0.01, 2	0.997, 0.01, 1.50
u_c_lolo.0 **u_i_lolo.0** **u_s_lolo.0**	0.998, 0.01, 3.75	0.998, 0.01, 3	0.998, 0.01, 2	0.998, 0.01, 1.50

5.5.Test Ortamı

Benzetilmiş tavlama yaklaşımı ile oluşturulan bilgisayar programının çalıştırılarak testlerinin yapıldığı platformun özellikleri aşağıda verilmiştir.

- Intel Core i7 CPU 2.5 GHz
- 16 GB 1600 MHz DDR3 bellek
- 64 Bit Mimarisi
- Windows 10 işletim sistemi
- Visual Studio 2017,
- g++ derleyici, *Ofast* optimizasyon bayrağı

KAYNAKLAR

Ali, S., Siegel, H.J., Maheswaran, M., Hensgen, D., Ali, S., 2000. Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems. Tamkang Journal of Science and Engineering, Vol. 3, No. 3, 195-207.

Al-Qadhi, A.K., Ariffin, A.A., Latip, R., Al-Zubaidi, A., 2018. Two Stages Transfer Algorithm (TSTT) for Independent Tasks Scheduling in Heterogeneous Computing Systems. Journal of Physics Conference Series, 1018 (1).

Anonim, 2013. Paralel Programlama Nedir. https://www.hermesiletisim.net/dev/paralel-programlama-nedir-2#.W8InQmgzYdU (9.10.2018).

Anonymous, 2012. https://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/index.htm (16.09.2018)

Anonymous, 2014. What Is Apache Hadoop. http://hadoop.apache.org/ (8.8.2018).

Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B. 2001. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing, 61(6), 810–837.

Erbay, H., Kör, H., 2016. Büyük Veri ve Büyük Verinin Analizi. Uluslararası Bilim ve Teknoloji Konferansı 3-6 Ekim, Ankara.

Flórez, E., Barrios C.J., Pecero J.E., 2015. Methods for Job Scheduling on Computational Grids: Review and Comparison. High Performance Computing, In: Osthoff, C., Navaux, P.O.A., Hernandez C.J.B., Dias P.L.S., (eds). Communications in Computer and Information Science, Cham, 19-33.

Gogos, C., Valouxis, C., Alefragis, P., Goulas, G., Voros, N., Housos, E., 2016. Scheduling independent tasks on heterogeneous processors using heuristics and Column Pricing. Future Generation Computer Systems. 60(2016), 48-66.

Hashem, I.A.T., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A., Khan, S.U., 2014. The rise of “big data” on cloud computing: Review and open research issues. Information Systems, 47 (2015), 98-115.

İlter, H., 2013. Hadoop. http://devveri.com/hadoop-nedir (8.8.2018).

Li, J., Qiu, M., Ming, Z., Quan, G., Qin, X., Gu, Z., 2012. Online optimization for scheduling preemptable tasks on IaaS cloud systems. Journal of Parallel and Distributed Computing. 72 (5), 666-677.

Merendino, S., Celebi, M.E., 2013. A Simulated Annealing Clustering Algorithm Based on Center Perturbation Using Gaussian Mutation. Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference, Florida.

Nagina, Dhingra, S., 2016. Scheduling Algorithms in Big Data: A Survey. International Journal Of Engineering And Computer Science, 5(8), 17737-17743.

Nesmachnow, S., Cancela, H., Alba, E., 2010. Heterogeneous computing scheduling with evolutionary algorithms. Soft. Comput. 15(4), 685–701.

Nesmachnow, S., Cancela, H., Alba, E., 2012. A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. Appl. Soft Comput. 12(2), 626–639.

Okutucu, B.O., 2012. Bulut Bilişim ve Teknolojileri. Yüksek Lisans Tezi, Okan Üniversitesi Fen Bilimleri Enstitüsü, İstanbul.

Panda, S.K., Nag, S., Jana, P.K., 2014. A smoothing based task scheduling algorithm for heterogeneous multi-cloud environment. 2014 International Conference on Parallel, Distributed and Grid Computing.

Panda, S.K., Jana, P.K., 2015. Efficient task scheduling algorithms for heterogeneous multi-cloud environment. The Journal of Supercomputing 71(4),1505-1533.

Panda, S.K., Pande, S.K., Das, S., 2017. Task Partitioning Scheduling Algorithms for Heterogeneous Multi-Cloud Environment. Arabian Journal for Science and Engineering, 43 (2), 913-933.

Patel, A.B., Birla, M., Nair, U., 2012. Addressing big data problem using Hadoop and Map Reduce. 2012 Nirma University International Conference on Engineering, Nuicone.

Sagiroglu, S., Sinanc, D., 2013. Big data: A review. Collaboration Technologies and Systems (CTS), 2013 International Conference on, San Diego, CA, USA.

Senthilkumar, M., Ilango, P., 2016. A Survey on Job Scheduling in Big Data. Cybernetics and Information Technologies, 16(3), 1311-9702.

Subashini, G., Bhuvaneswari, M.C., 2010. A Fast and Elitist Bi-Objective Evolutionary Algorithm for Scheduling Independent Tasks on Heterogeneous Systems. ICTACT, Journal on Soft Computing, Vol. 1, 9-17.

Syed, H.A., Kamran, R., Usman, A., Syed, S.A.A., Manzoor, H., 2015. Cloud Task Scheduling Using Nature Inspired Meta-Heuristic Algorithm. International Conference on Open Source Systems and Technologies (ICOSST), Lahore, Pakistan.

Xhafa, F., 2007a. A Hybrid Evolutionary Heuristic for Job Scheduling on Computational Grids. Hybrid Evolutionary Algorithms, Abraham, A., Grosan, C., Ishibuchi, H. (eds). Studies in Computational Intelligence, Berlin, 269-311.

Xhafa, F., Durresi, A., Barolli, L., 2007b. Batch mode scheduling in grid systems. International Journal of Web and Grid Services, Vol. 3, No. 1, 19–37.

Xhafa, F., Alba, E., Dorronsoro, B., Duran, B., 2008a. Efficient Batch Job Scheduling in Grids Using Cellular Memetic Algorithms. Metaheuristics for Scheduling in Distributed Computing Environments, 273–299.

Xhafa, F., Carretero. J., Alba, E., Dorronsoro, B., 2008b. Design and evaluation of tabu search method for job scheduling in distributed environments. Proceedings of the 22th International Parallel and Distributed Processing Symposium, 1–8.

Xhafa, F., Duran, B., Abraham, A., Dahal, K.P., 2008c. Tuning Struggle Strategy in Genetic Algorithms for Scheduling in Computational Grids. Neural Network World, 18 (3), 209-225.

Wolfram|Alpha, 2018, http://www.wolframalpha.com/ (9.10.2018)

ÖZGEÇMİŞ

Esra ÇELİK 1993 yılında Erzurum'un Ilıca ilçesinde doğdu. İlk, orta ve lise eğitimini Ilıca'da tamamladı. 2011 yılında Atatürk Üniversitesi Bilgisayar Mühendisliği bölümünde öğrenime başlayıp 2015 yılında Bilgisayar Mühendisi olarak mezun oldu. 2015 yılında Atatürk Üniversitesi Fen Bilimleri Ensitüsünde Yüksek Lisans öğrenimine başladı.