ÇOKLU KOVAN TEMELLİ PARALEL BİR GENETİK ALGORİTMA İLE ÇOKLU İŞLEMCİLERE YÖNELİK İLETİŞİM MALİYETLİ GÖREV ÇİZELGELEME PROBLEMİNİN OPTİMİZASYONU

Raşid MORADİ

Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Yrd. Doç. Dr. Deniz DAL
2014

ATATÜRK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

ÇOKLU KOVAN TEMELLİ PARALEL BİR GENETİK ALGORİTMA İLE ÇOKLU İŞLEMCİLERE YÖNELİK İLETİŞİM MALİYETLİ GÖREV ÇİZELGELEME PROBLEMİNİN OPTİMİZASYONU

Raşid MORADİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ERZURUM
2014

T.C.
ATATÜRK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

TEZ ONAY FORMU

ÇOKLU KOVAN TEMELLİ PARALEL BİR GENETİK ALGORİTMA İLE ÇOKLU İŞLEMCİLERE YÖNELİK İLETİŞİM MALİYETLİ GÖREV ÇİZELGELEME PROBLEMİNİN OPTİMİZASYONU

Yrd. Doç. Dr. Deniz DAL danışmanlığında, Raşid MORADİ tarafından hazırlanan bu çalışma 14/02/2014 tarihinde aşağıdaki jüri tarafından, Bilgisayar Mühendisliği Ana Bilim Dalı'nda Yüksek Lisans tezi olarak oybirliği (3./0) ile kabul edilmiştir.

Başkan	: Yrd. Doç. Dr. Deniz DAL	İmza	:
Üye	: Yrd. Doç. Dr. Tolga AYDIN	İmza	:
Üye	: Yrd. Doç. Dr.Gökay AKKAYA	İmza	:

Yukarıdaki sonucu onaylıyorum

Prof. Dr. İhsan EFEOĞLU
Enstitü Müdürü

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

ÖZET

Yüksek Lisans Tezi

ÇOKLU KOVAN TEMELLİ PARALEL BİR GENETİK ALGORİTMA İLE ÇOKLU İŞLEMCİLERE YÖNELİK İLETİŞİM MALİYETLİ GÖREV ÇİZELGELEME PROBLEMİNİN OPTİMİZASYONU

Raşid MORADİ

Atatürk Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Yrd. Doç. Dr. Deniz DAL

Çoklu işlemciye sahip sistemler için önemli sorunlardan biri de "Görev Çizelgeleme"dir. Görev çizelgeleme bir NP-zor problemdir ve görev çizgesini oluşturan görevlerin tamamının en kısa zamanda işletilmesini sağlayacak bir çizelgelemenin geliştirilmesini hedeflemektedir. Burada çizelgelemeden kastedilen çizgedeki her bir görevin hangi zaman aralığında ve hangi işlemci tarafından işletileceğini belirlemektir. Görevlerin sayısı ve birbirine bağlılıkları bir yönlendirilmiş çevrimsiz çizge (DAG) ile gösterilmektedir.

Genetik Algoritmalar (GA) birçok NP-zor problemin çözümü için kullanılan araçlardır. Paralel Genetik Algoritmalar (PGA) ise performans ve ölçeklendirilebilirlik açısından önemli kazançlar sağlayan, GA'nın paralel uygulamalarıdır. Öte yandan PGA'nın üç modeli vardır: 1. efendi-köle modeli 2. çoklu kovan modeli 3. hibrid hiyerarşik model.

Bu çalışmada "çoklu kovan modeli" kullanılmıştır. Bu modelde her işlemci bir arı kovanını temsil etmektedir ve her bir işlemcinin genetik algoritmayı bağımsız olarak çalıştırması, göç oranı olarak da bilinen belirli sayıda iterasyon sonrası elindeki en kötü (uygunluk fonksiyon değeri en yüksek) kromozomu komşu işlemcinin en iyi kromozomu ile bir mesaj geçen arayüz (MPI) mesajlaşması sonrası değiştirmesi hedeflenmektedir.

Bu çalışma ayrıca genetik algoritmanın önemli bir parçası olan kromozom kodlanması için de eldeki problemin çözümüne yönelik iki parçalı yeni bir öneri sunmaktadır.

2014, 117 sayfa

Anahtar Kelimeler: Görev Çizgesi, Yönlendirilmiş Çevrimsiz Çizge, Çizelgeleme, Genetik Algoritma, Paralel Programlama

ABSTRACT

MASTER THESIS

OPTIMIZATION OF MULTIPROCESSOR TASK SCHEDULING WITH COMMUNICATION COSTS USING A MULTI-HIVE BASED PARALLEL GENETIC ALGORITHM

Raşid MORADİ

Ataturk University
Institute of Applied Sciences
Department of Computer Engineering

Supervisor: Asst. Prof. Dr. Deniz DAL

One of the major problems for the multiprocessor systems is task graph scheduling. Task graph scheduling is a NP-hard problem and aims to develop a schedule to operate all the tasks of the graph in the shortest time possible. A schedule has two aspects: assignment and binding. The assignment determines a time slot for each task to be executed whereas the binding determines which processor is going to execute a particular task. The task graph is represented as a Directed Acyclic Graph (DAG).

Genetic Algorithms (GA) are commonly used for finding optimal or near optimal solutions for many NP-hard problems. On the other hand, Parallel Genetic Algorithms (PGA), that are parallel implementations of the GA, provide significant improvements in terms of performance and scalability. There are three models commonly employed in the literature as far as PGA are concerned: 1. the master-slave model 2. the multi-hive with migration 3. the hybrid hierarchical model.

In this study, the multi-hive with migration approach has been used. In this model, each processor represents a bee hive and runs the genetic algorithm independently. After some specific number of iterations, that is also known as migration rate, an MPI (Message Passing Interface) messaging occurs between neighboring processors. By this way, each processor replaces its worst chromosome with the best chromosome that it gets from its neighbor.

This study also provides a new two-part encoding scheme towards the solution of the problem in hand.

2014, 117 pages

Keywords: Task Graph, Directed Acyclic Graph (DAG), Scheduling, Genetic Algorithm, Parallel Programming

TEŞEKKÜR

Yüksek Lisans tezimin belirlenmesi ve tamamlanması aşamalarında, öncelikle tezimi değerlendirerek, değerli görüş ve eleştirileriyle her türlü ilgi ve yardımı esirgemeyen tez danışmanım Sayın Yrd. Doç. Deniz DAL'a en içten teşekkürlerimi sunarım.

Tez çalışmam esnasında bana destek olan ve bilgi alışverişinde bulunduğum arkadaşım Sayın Saed HOSEINGOLIZADEH'ye, tez çalışmamın her aşamasında bana verdiği destekle çalışma azmi kazandıran aileme en derin sevgi ve saygılarımı sunarım.

Raşid MORADİ
Şubat, 2014

.

İÇİNDEKİLER

SİMGELER ve KISALTMALAR DİZİNİ

ABC	Artificial Bee Colony Algorithm
ALAP	As Late As Possible
ASAP	As Soon As possible
bl	Bottom-Level
CP	Critical Path
DAG	Directed Acyclic Graph
DL	Dynamic Level
DRT	Dominant Request Tree
DST	Dominant Service Tree
FT	Finishing Time
GA	Genetic Algorithm
LAN	Local-Area Network
MHA	Multi-Hive Algorithm
MH-ABC	Multi-Hive Artificial Bee Colony Algorithm
MH-ES-ABC	Multi-Hive Enhanced Sequential Artificial Bee Colony Algorithm
MPI	Message Passing Interface
PGA	Parallel Genetic Algorithm
PMC	Priority bases Multi Chromosome
SL	Static Level
SSH	Secure Shell
ST	Start Time
tl	Top-Level
VAG	Virtual Architecture Graph
WAN	Wide Area Network

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Birkaç on yıl öncesine kadar, süper bilgisayar sistemlerinin konsolları merkezi bir bilgisayara bağlıydı. Bu tip bir sistem merkezi olarak çalışır ve veriler ile işlemler merkezi bilgisayar üzerine yerleştirilmiştir. Kişisel bilgisayarlar ve iş istasyonları sistemleri geldikten sonra eski tip bilgisayarların kullanımında büyük bir düşüş oldu ve birden fazla bilgisayar ile bağlanan ağların kullanımında artış oldu. Bilgisayar ağları kulanılarak çoklu bilgisayarlar ve sunucular bir düğümle birbirlerine bağlanır. Fakat bu bilgisayarlar birbirlerine bağlandığı zaman işlem gücü ve depolama durumu yetersiz kalır. Çünkü farklı kaynaklar sistemler arasında dağıtılmış bir şekilde yerleştirilmiş ve bu kaynakları iyi kulanmak için onların üzerinde iyi bir yönetim gereklidir. Bu sebeple dağıtık sistemler ortaya çıktı. Aslında, burada merkezi sistemlerden dağıtık sistemlere çeviren hesaplama model değişimiyle karşılaşıyoruz (Anonymous 2012).

1.1. Dağıtık Sistemler

Ağ üzerindeki bilgisayarlarda bulunan donanım veya yazılım bileşenlerinin yalnız mesaj göndererek haberleştikleri sistemdir. Dağıtık bir sistem, kullanıcılara tek bir sistem olarak görünen, bağımsız bilgisayarlar bütünüdür. Çeşitli işlemciler arasında fiziksel hesaplama dağıtımıdır. Her işlemcinin kendi yerel belleği vardır; İşlemciler hızı yüksek otobüs yada telefon hatları gibi çeşitli iletişim hatları aracılığıyla birbirleriyle iletişim kurarlar (Cantú-Paz 1998).

Dağıtık Sistemin avantajları:

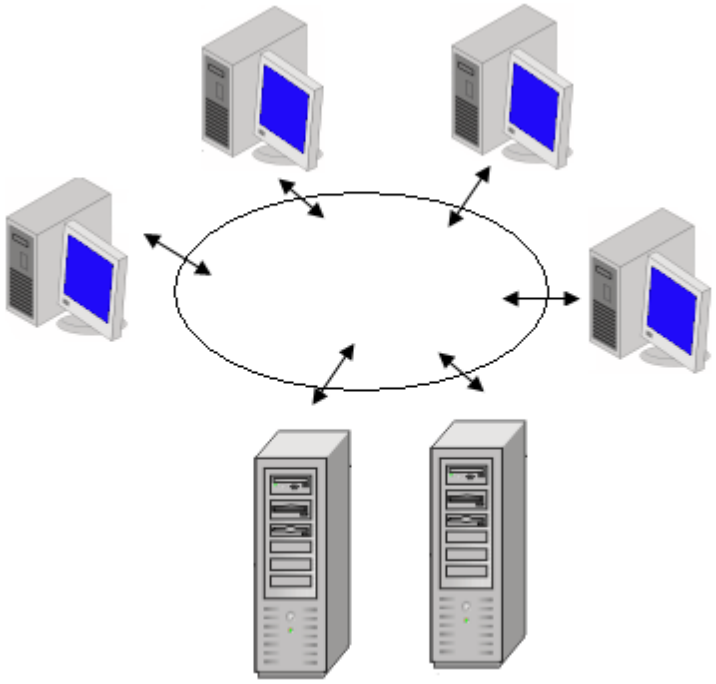
1. Kaynak paylaşımı
2. Hesaplama hızı yüksek – yük paylaşımı
3. Güvenilirlik
4. İletişim

Dağıtık Sistemi ağ alt yapısı gerçekleştirir. Yerel Alan Ağları (LAN) ya da Geniş Alan Ağları (WAN) İşlemci–Sunucu ya da peer–to–peer (Birbirleriyle eş olan) sistemi olabilir (Colouris *et al.* 2011).

Bir dağıtık sistem, hafıza ve saati paylaşmayan işlemciler kümesidir. Her işlemci kendisine ait yerel hafızaya sahiptir. İşlemciler birbirleriyle iletişim ağları ile haberleşirler. Dağıtık sistemlerde bilgi işleme, tek bir makinede sınırlanmamış birkaç bilgisayar üzerine dağıtılmıştır (Ghader 2010).

Büyük bilgisayar tabanlı sistemlerin çoğu dağıtık sistemlere geçmiştir. Dağıtık sistemlerde; donanım ve yazılım kaynaklarının paylaşımı yapılabilir, bir hata oluştuktan sonra işleme devam edilebilir, eş zamanlı işleme sayesinde performans arttırılabilir, farklı sağlayıcılardan gelen yazılım ve donanımlar kullanılabilir. Tüm bu faydaları dışında dağıtık sistemler genelde merkezi sistemlere göre daha karmaşıktırlar ve sistem yönetimi için daha fazla uğraş gerekir (Gülsün vd 2009).

Çoklu işlemci mimarileri en basit dağıtık sistem modelidir. Dağıtık sistem, Şekil 1'de görüldüğü gibi bir ağ üzerine kuruludur. Yazılımla, ağdaki bileşenler arası işlevsel uyumluluk ve kullanıcı açısından saydamlık sağlanır. Bir ağdan farklı olarak ağ üzerinde gerçekleşen işlemler kullanıcıya bir görünmektedir yani kullanıcı karşısında, tek ve bütünleşmiş bir yapı görülmektedir (Transparency). Kullanıcı, bir işlemci ve bir arayüz olduğunu düşünse de bu sistemler, değişik bilgisayar sistemleri üzerindeki verileri ve işlemleri bir bütün olarak işleyebilir ve çalıştırabilirler. Dağıtık sistemlere, sunucu-istemci, peer-to-peer (noktadan noktaya) modelleri örnek gösterilebilir (Yavuz vd 2012).

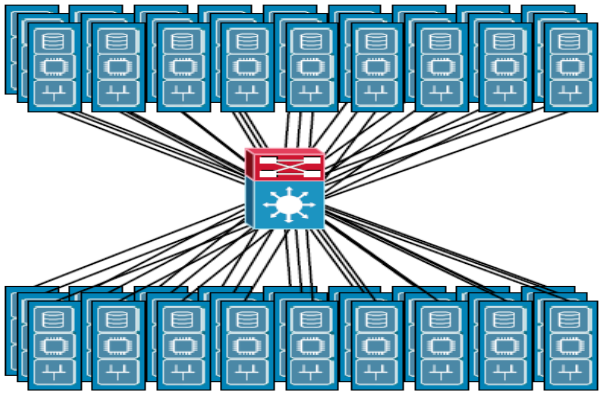


Şekil 1.1. Dağıtık sistem modeli (Yavuz vd 2012)

1.2. Küme Hesaplama

Bu yöntemde bilgisayarlar ağ üzerinde birbirine bağlanarak tek bir bilgisayar gibi çalıştırılırlar. Bu yöntemle birleştirilen bilgisayarların her biri bu ağdan bağımsız da çalışabilen (standalone) bilgisayarlardır. Bilgisayar kümelerinin en çok kullanılan tipi Beowulf kümeleridir (Beowulf clusters). Beowulf kümelerinde günlük hayatta alışık olduğumuz bilgisayar kasaları, ethernet kullanılarak TCP/IP üzerinden yerel alan ağı ile birbirine bağlanır. Dünyanın en hızlı ilk 500 bilgisayarı içinde en çok kullanılan yöntem budur (Gözütok ve Özdemir 2004).

Hesaplamada küme bilgisayar kullanımı 1994 senesinde NASA'da Beowulf projesi ile başlamıştır. 16 Intel 486 DX4 işlemci ethernet ile bağlanmıştır. Yüksek performanslı hesaplama, artık küme bilgisayarlarla hesaplama halini almıştır. Küme bilgisayar, birlikte çalışmak üzere bağlanmış birden fazla sunucudan oluşur. En önemli dezavantajı kullanıcıya tek sistem arayüzü sunamamasıdır (Hwang *et al.* 2006).



Şekil 1.2. Küme hesaplama (Temirci 2009)

Küme hesaplamanın tanımından da belirtildiği gibi kümeli sistem kullanıcıya tek bir sistem olarak görünmektedir; fakat burada dikkat edilmesi gereken sistemdeki tüm bilgisayarların kullanıcıların kullanımına açık olduğudur. Ayrıca küme hesaplamada tüm bilgisayarlar dış dünyaya açıktır. Bu tür sistemler için bir yerel ağa bağlı NT iş istasyonları kümesi örnek olarak verilebilir (Kim *et al.* 2007).

Küme hesaplamanın avantajları:

1. Birden çok bilgisayarın kaynakları kullanıldığından bir bilgisayar sisteminin kullanılmasından elde edilebilecek çok daha yüksek seviyede başarım ve işlem gücü elde edilmesi küme hesaplamanın sistemlerde kullanılmasını cazip kılan en önemli özelliktir.
2. Küme hesaplama çökmelere karşı etkin bir koruma sunmaktadır. Bir bilgisayarın çökmesi durumunda o bilgisayara verilmiş olan iş hemen başka bir bilgisayara yönlendirilerek yapılmakta olan işin aksaması önlenmektedir.
3. Kümeli sisteme bir bilgisayar dâhil edilmek istendiği zaman bu bilgisayarın işletim sistemi sistemdeki diğer bilgisayarlardan kopyalanarak kurulabilmekte ve ufak ayar değişiklikleriyle bu bilgisayar sisteme kolaca dâhil edilmektedir.

Dezavantajları:

1. Küme hesaplamaya tabi tutulacak sistemler doğaları gereği paralel işleme uygun olmalılar ve bu tür uygulamaların sayısı oldukça sınırlıdır.
2. Küme hesaplama bilgisayar ağlarının gelişmesi sonucu ortaya çıkmış bir hesaplama türüdür. Ancak küme hesaplamadaki en büyük sorunlar ağ iletişiminden kaynaklanmaktadır (Temirci 2009).

1.3. Amaç

Bu çalışmada çoklu işlemciler için yeni bir görev çizelgeleme algoritması ortaya çıkaracağız. Çoğu önerilen algoritma programları Yönlendirilmiş Çevrimsiz Çizge (DAG) ile gösterilir. Bizim hazırlayacağımız bu algoritma, bu programı (DAG) giriş olarak alıp, hem paralel hem seri olarak çoklu işlemcili sistemlerde uygulayacaktır. Bu uygulama zamanı başka algoritmalara kıyasla en az olacaktır.

1.4. Kapsam

Bu çalışmada yeni bir evrimsel algoritmayla çoklu işlemcili sistemlerde zaman uygulamasını bulacağız. Bu zamanı hem paralel genetik yöntemle (Paralel Çoklu Kovan Modeli) hem de seri olarak buluyoruz. Paralel genetik yöntem her işlemci uzerinde uygulanacaktır. Paralel genetik yöntemde her işlemci başka işlemciye mpi komutlarıyla mesaj gönderecektir. Yani bu algoritmayı her işlemci başka işlemcilerden bağımsız olarak uygulayacaktır. Bu algoritmayı kodladıktan sonra ssh secure shell client programıyla çalıstıracğız ve uygulama zamanını tespit etmiş olacağız.

2. KURAMSAL TEMELLER

2.1. Çizelgeleme (Zamanlama)

Farklı işlemciler arasında görev dağılımına göre programın yürütme zamanının azaltılmasına çizelgeleme (zamanlama) denir. Genel olarak zamanlama algoritmaları statik ve dinamik olmak üzere iki kategoride sınıflandırılır (Liu *et al.* 2006).

2.1.1. Dinamik çizelgeleme

Dinamik çizelgelemenin amacı sadece bir çizelgelemenin yüksek kaliteli olmasını sağlamak değil, aynı zamanda programın çalışma zamanı yükünü en aza indirmektir.

2.1.2. Statik çizelgeleme

Statik çizelgeleme algoritmalarında, ilk olarak, çizelgeleme bilgilerinin belirlenmiş olmalar gereklidir. Bu bilgiler şunlardır: Sistemin yapısı, her işlemin uygulama zamanlarının bağımsızlığı, işlemciler arasındaki iletişim maliyeti. Statik çizelgeleme çeviriden ve paralel programın uygulanmasından önce gerçekleşir (Daoud and Kharma 2008).

Bu çalışmada statik çizelgelemeyi inceleyeceğiz. Belirli bir uygulama için, farklı çizelgelemeler yapılabilir. Fakat şu konuya dikkat etmek gerekir: Her çizelgelemenin toplam uygulama zamanı diğerlerinden farklı olmalıdır. Örneğin, Şekil 2.2 (a) ve 2.2 (b) gibi iki olası farklı çizelgeleme Şekil 2.1 için örnek olarak gösterilebilir. Gördüğünüz gibi Şekil 2.2 (a) çizelgelemesi Şekil 2.2(b)'den fazladır (Sarkar 1987).

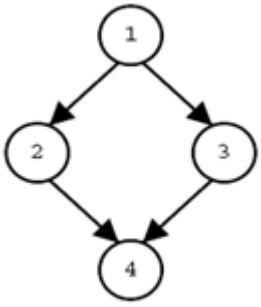
2.1.2.a. Tamamlanma zamanı (makespan)

İşlemciler üzerinde çalışan görevlerin Gantt grafiğine tamamlanma zamanı denir.

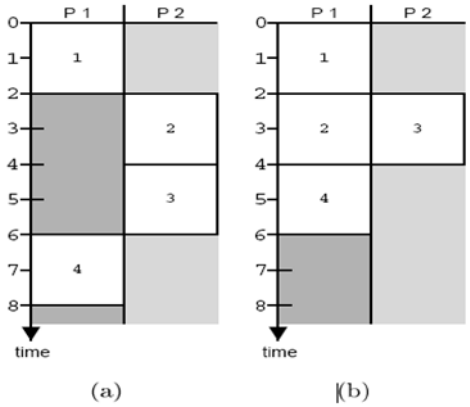
2.1.2.b. Bitiş zamanı (FT)

Bir programın en son biten görevinin zamanına bitiş zamanı denir ve FT ile gösterilir.

Örneğin, Şekil 2.2 (a) bitiş zamanı 8'e ve Şekil 2.2(b) bitiş zamanı 6'ya eşittir.



Şekil 2.1. Bağımlılık çizelgesi (Garey and Johnson 1979)

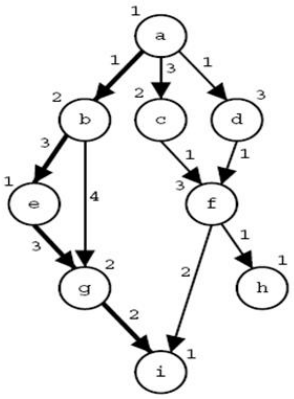


Şekil 2.2. Yukarıda gösterilen iki farklı çizelgelemede her görev çalışma süresi 2 zaman birimidir (Garey and Johnson 1979)

Uygun bir çizelgeleme bulmak çok zor ve bir NP-zor sorundur ve bu sorun polinom zamanda çözülemez (Garey and Johnson 1979). Bu nedenle sezgisel yöntemler kullanılarak uygun bir çizelgeleme bulunabilir. Son yıllarda çeşitli çizelgeleme algoritmaları önerilmiştir.

Çoğu önerilen algoritma programları Yönlendirilmiş Çevrimsiz Çizge (DAG) ile

gösterilir. DAG'lar bağımlılık çizgelerinin gelişmiş halleridirler. Şekil 2.1'de bir DAG örnek olarak gösterilmiştir. Düğümlere atanan ağırlıklar yürütme zamanlarını temsil ediyor. Kenarları atanan ağırlıklar, farklı işlemciler üzerinde çalıştırılacak düğümler arasındaki iletişim maliyetini temsil ediyor. Şekil 2.3'de bir DAG örneği görebilirsiniz.



Şekil 2.3. İletişim maliyetleri ve hesaplama maliyetleri ile ilgili bir görev çizgesi (Daoud and Kharma 2008)

Bu iki görevin farklı işlemciler üzerinde uygulandığı durumlar da iletişim maliyeti olarak adlandırılabilir. Eğer iki görev aynı işlemci üzerinde uygulanırsa iletişim maliyeti sıfır olacaktır (Gerçekte, iletişim maliyeti sıfır değil, fakat çok küçük bir zamandır o yüzden sıfır kabul ediyoruz) (Daoud and Kharma 2008).

2.2. Genetik Algoritmalar

Genetik algoritmalar, Darwin' in doğal seçim ve evrim teorisi ilkelerine dayanan bir arama ve optimizasyon yöntemidir. Bu yöntem ilk olarak, John Holland ve arkadaşlarının yaptığı çalışmalarda (1970'li yıllarda) ortaya çıkmıştır. Önbilgi ve varsayımlar olmadan, sadece uygunluk fonksiyonu ile çalışabilmektedir (Ütün 2009).

2.2.1. Biyolojik altyapı

Gen: Kalıtsal molekülde bulunan ve organizmanın karakterlerinin tayininde rol oynayan kalıtsalbirimlere denir.

Kromozom (Birey): Birden fazla genin bir araya gelerek oluşturduğu diziye denir.

Popülasyon: Kromozomlardan oluşan topluluğa denir. Popülasyondaki kromozom sayısı arttıkça çözüme ulaşma süresi (iterasyon sayısı) azalır (Lozano *et al.* 2005).

Başlangıç Popülasyonu

GA, çözüm adımlarına belirlenen gösterim şekline uygun kodlanmış bireylerden oluşan bir başlangıç popülasyonu oluşturarak başlar. Başlangıç popülasyonundaki her bir kromozom, problemin olası bir çözümünü temsil eder. Popülasyon sürekli daha iyi çözümler oluşturmaya çalıştığı için, zaman içinde değişir. Popülasyon büyüklüğü, problemin yapısına göre belirlenmelidir (Polratanasuk *et al.* 2010).

2.3. Kodlama

Kodlama genetik algoritmanın çok önemli bir kısmını oluşturmaktadır. Bir problemin çözümüne başlarken sorulması gereken ilk sorudur. Kurulan genetik modelin hızlı ve güvenilir çalışması için bu kodlama doğru yapılmalıdır (Mostafa *et al.* 2011).

2.3.1. Kodlama türleri

2.3.1.a. İkili kodlama

Çizelge 2.1. İkili kodlama

Kromozom1	1101100100110110
Kromozom2	1101111000011110

2.3.1.b. Değer kodlama

Çizelge 2.2. Değer kodlama

Kromozom1	1.2324 5.3243 0.4556 2.3293 2.4545
Kromozom2	ABDJEIFJDHDIERJFDLDFLFEGT
Kromozom3	(arka), (arka), (sağ), (ön), (sol)

2.3.1.c. Permütasyon kodlama

Çizelge 2.3. Permütasyon kodlama

Kromozom1	1 5 3 2 6 4 7 9 8
Kromozom2	8 5 6 7 2 3 1 4 9

2.4. Genetik İşlemler

2.4.1. Seçim

Seçilen uygunluk fonksiyonuna ve seçim yöntemine göre elimizdeki popülasyondan yeni bir neslin bireylerini seçmektir. Ebeveynler uygunluk değerlerine göre eşleşmek üzere seçilirler. Uygunluğu yüksek olan bireyin, yeni nesle aktarılma ihtimali daha yüksektir (Yalçın 2011).

Seçim yöntemleri

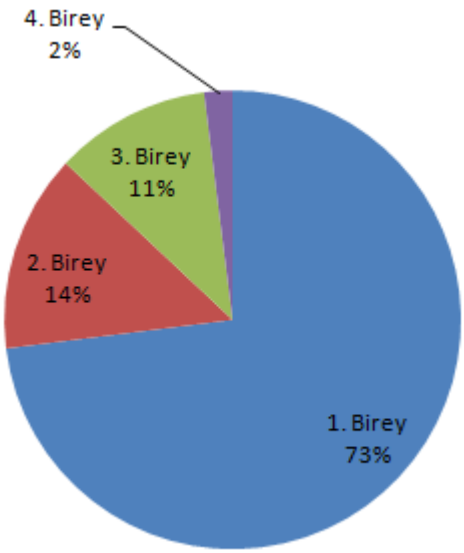
Seçim yöntemleri, oluşturulacak bir sonraki nesil için çaprazlanacak bireylerin belirlenmesinde kullanılır. Aşağıda rulet tekeri seçim yöntemi ya da uygunluk orantılı seçim yöntemi (Roulette Wheel Selection or fitness proportionate selection), turnuva seçim yöntemi (tournament selection) ve rütbe seçim yöntemi (rank selection) anlatılmaktadır. Anlatımlar örnek üzerinden yapılacaktır. Seçilecek bireyler ve uygunlukları aşağıdaki tabloda gösterilmiştir:

Çizelge 2.4. Seçilecek bireyler

Birey	Uygunluk Değeri
1.Birey	73
2.Birey	14
3.Birey	11
4.Birey	2

2.4.1.a. Rulet tekeri seçim yöntemi (roulette wheel selection)

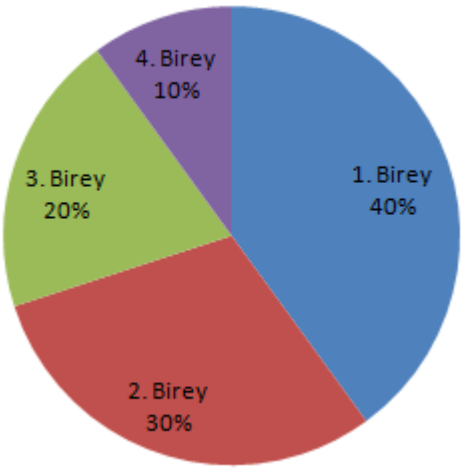
Bu seçim yönteminde bireylerin seçilme olasılıkları uygunluk fonksiyonlarıyla orantılıdır. Örnek olarak verilen bireylerin seçilme olasılıkları bu seçilim yöntemi uyarınca aşağıdaki gibi olacaktır:



Şekil 2.4. Rulet tekeri seçim (Anonim 2013b)

2.4.1.b. Rütbe seçim yöntemi (rank selection)

Bu seçilim yönteminde bireylerin seçilme olasılıkları uygunluk derecelerinin sıralamasıyla orantılıdır. Örnek olarak verilen bireylerin bu seçilim yöntemine göre seçilme olasılıkları aşağıdaki gibi olacaktır:



Şekil 2.5. Rütbe seçim (Anonim 2013b)

2.4.1.c. Turnuva seçim yöntemi (tournament selection)

Bu yöntemde popülasyondan rassal olarak belirli sayıda birey seçilir ve kendi aralarında uygunluklarına göre turnuvaya sokulur. Örnek olarak popülasyonumuzdan rastgele iki birey seçelim: 4. birey ve 2. birey olsun, bu bireyler kendi aralarında turnuvaya sokulduğunda çaprazlanacak bireylerden biri seçilmiş olur: 2. Birey (Anonim 2013b).

2.4.2. Çaprazlama

Ata kromozomun yerlerini değiştirerek çocuk kromozomlar üretmek ve böylelikle zaten uygunluk değeri yüksek olan ata kromozomlardan daha yüksek uygunluklu çocuk kromozomlar üretmektir (Nedjah *et al.* 2006).

Çizelge 2.5. Çaprazlama

Kromozom1	11011 | 00100110110
Kromozom2	11011 | 11000011110
Yavru1	11011 | 11000011110
Yavru2	11011 | 00100110110

2.4.2.a. Çaprazlama çeşitleri

a. Tek noktalı caprazlama

A1: 0 1 0 0 1 1 1 | 0 1 0 1 0
A2: 1 0 0 1 1 1 0 | 0 0 1 0 0
A'1: 0 1 0 0 1 1 1 | 0 0 1 0 0
A'2: 1 0 0 1 1 1 0 | 0 1 0 1 0

b. İki noktalı caprazlama

A1: 0 1 0 0 | 1 1 1 0 | 1 0 1 0
A2: 1 0 0 1 | 1 1 0 0 | 0 1 0 0
A'1; 0 1 0 0 | 1 1 0 0 | 1 0 1 0
A'2:1 0 0 1 | 1 1 1 0 | 0 1 0 0

2.4.3. Mutasyon

Yeniden ve sürekli yeni nesil üretimi sonucunda, belli bir süre sonra nesildeki kromozomlar birbirlerini tekrar edebilir. Böylece farklı kromozom üretimi durur veya azalır. İşte bu nedenle nesildeki kromozom çeşitliliğini artırmak için kromozomlardan bazıları mutasyona tabi tutulur (Yalçın 2011).

A1: 0 | 1 | 0 0 1 1 1 0 1 | 0 | 1 0
A'1: 0 | 0 | 0 0 1 1 1 0 1 | 1 | 1 0

2.5. Yerleştirme

Basit veya kuşaksal GA'lar tüm nüfusu, seçim şemasını komut aşamasında yerleştirir. Kararlı durum ya da online GA'lar farklı yerleştirme şemaları kullanırlar.

2.5.1. Yerleştirme çeşitleri

1. Kötü yerleştirme, 2. İyi yerleştirme, 3. Ebeveyn yerleştirme, 4. Rastgele yerleştirme, 5. En benzer yerleştirme (kalabalık). Bunlardan sadece kötü yerleştirme ve en benzer yerleştirme genel olarak çok etkilidirler (Ebeveyn yerleştirme çalıştığında, bunun sebebi ebeveynlerin birbirlerine benzer olmasıdır) (Yaeger 2013).

2.6. Genetik Algoritmanın Parametreleri

2.6.1. Çaprazlama olasılığı

Çaprazlamanın ne kadar sıklıkla yapılacağını belirtir.

2.6.2. Mutasyon olasılığı

Kromozom parçalarının ne kadar sıklıkla mutasyon geçireceğini belirtir.

2.6.3. Popülasyon büyüklüğü

Toplumdaki birey sayısını belirtir.

2.6.4. Uygunluk fonksiyonu

Bu fonksiyon ile kromozom içerisindeki kodlanmış ya da kodlanmamış bilgiler çözümlenerek sayısal bir değer elde (çıktı) edilir.

2.7. GA'nın Avantaj ve Dezavantajları

2.7.1. Avantajları

1. Çok amaçlı optimizasyon yöntemleri ile kullanılabilmesi. 2. Çok karmaşık ortamlara uyarlanması. 3. Kısa sürelerde iyi sonuçlar verebilmesi.

2.7.2. Dezavantajları

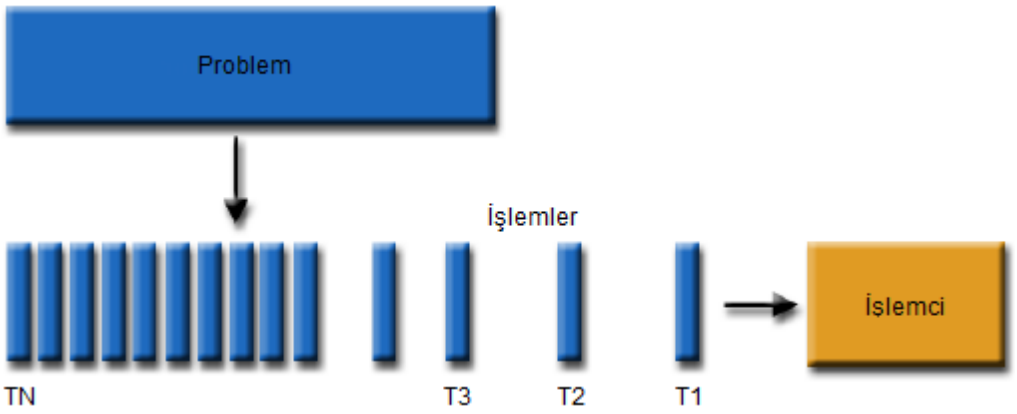
1. Son kullanıcının modeli anlaması güç. 2. Problemi GA ile çözmeye uygun hale getirmek zor 3. Uygunluk fonksiyonunu belirlemek zor. 4. Çaprazlama ve mutasyon tekniklerini belirlemek zor.

2.8. GA'nın Uygulama Alanları

1. Optimizasyon (Bakım, servis, depo toplama). 2. Görüntü işleme.3. Makine öğrenmesi (Yapay Sinir Ağlarında öğrenmeyisağlayan ağırlık hesaplamaları, robotlar). 4. Sosyal sistemler (Böcek kolonileri, Çok etmenli sistemlerde işbirliği). 5. Yapay sinir ağları. 6. Gezgin satıcı problem 7. Popülâsyon genetiği. 8. Evrim ve öğrenme (Yalçın 2011).

2.9. Seri Programlama

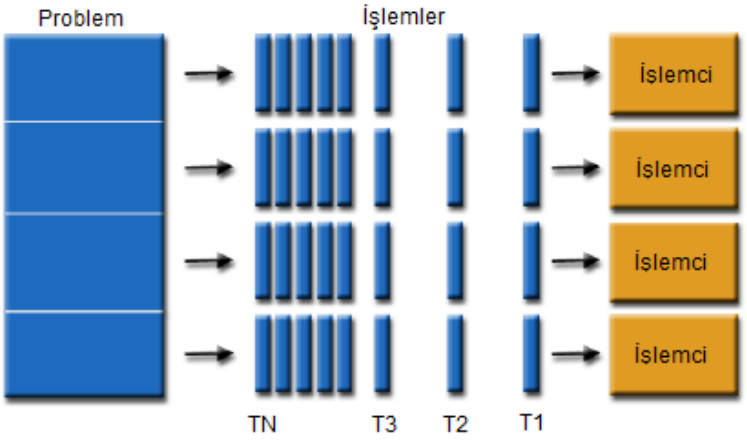
Yapılacak olan işlemin tek bir bilgisayarda ve tek işlemci üzerinde çalıştırılmasına "Seri Programlama" denir. Burada aynı anda sadece tek bir işlem yapılabilmektedir. Bir işlem bitmeden diğer bir işlem çalıştırılamaz. Seri programlamanın çalışma mantığı, Şekil 2.6'da gösterilmiştir (Ercan vd 2009).



Şekil 2.6.Seri programlama çalışma mantığı (Ercan vd 2009)

2.10. Paralel Programlama

Paralel programlama ise, bir problemi çözmek için birden fazla bilgisayar kaynaklarının aynı anda kullanılmasıdır. Bir işlem çoklu işlemci kullanarak çalıştırılır ya da çoklu çekirdek teknolojisine sahip işlemci üzerinde gerçekleştirilir. Bir problem parçalara ayrılarak aynı zamanda çözülebilir. Her parça farklı işlemci üzerinde aynı anda çalıştırılır. Paralel programlamanın çalışma mantığı, Şekil 2.7'de gösterilmiştir (Ercan vd 2009).



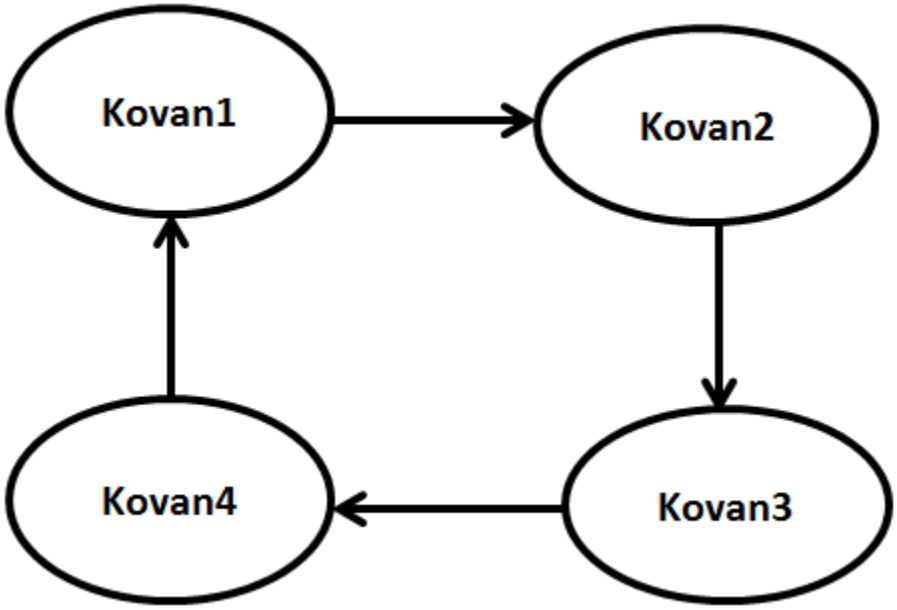
Şekil 2.7. Paralel programlama çalışma mantığı (Ercan vd 2009)

2.11. Paralel Genetik Algoritma

Paralel genetik algoritmalar (PGA'lar) farklı bireylere sahip birden fazla alt popülasyon üzerinde genetik algoritma (GA) çalıştırarak arama yapan bir en iyileme algoritmasıdır. PGA' ların başarılı bir arama yapmasını etkileyen en önemli unsurlardan biri kullanılan göç yöntemidir. Göç yöntemleri, seçilen bireylerin hangi alt popülasyonlara gönderileceğini belirler. Göç eden birey, alt popülasyondaki arama kalitesini ve buna bağlı olarak algoritma başarısını etkiler. PGA'ların GA'lardan daha başarılı sonuçlar üretmesi, göç işleminin farklılığa olan katkısının bir sonucudur. Bu nedenle, tercih edilen göç yönteminin farklılığı arttıracak bir etkisinin olması istenir (Adar ve Kuvat 2012).

2.11.1. Paralel çoklu kovan modeli

Çoklu kovan modeli (MH-ABC) farklı rastgele çekirdeklerle aynı zamanda başlatılmış iki veya daha fazla kovanı kulanan iri taneli çoklu nüfus sistemidir. Her nüfus bir ada olarak görülür. Kovanlar birbirinden bağımsız olarak çalışırlar ve her biri bir ABC algoritmasını çalıştırır, ama göç süreci periyodik olarak gerçekleşir. MH-ABC yaklaşımı her kovan için bir işlem çekirdeği kullanır. İki versiyon uygulanmıştır. Biri temel ABC (MH-ABC) kullanır ve diğeri ES-ABC (MH-ES-ABC) kullanır. Göçler, kovanlar arasında gerçekleşir ve kulanıcı tarafından tanımlanmış bazı parametreleri kapsayan göç politikasi aracıyla tanımlanmıştır. En önemli iki parametre şunlardır: göç boşluğu: başarılı göçler arasındakı nesillerin sayısıdır, göç oranı: Her bir göç olayında göç edecek bireylerin sayısıdır. Göçmenlerin seçilimi ve göçmenlerin yer değiştirmeleri için bazı kriterler tanımlanmıştır. Buna rağmen topoloji ve adalar arasında bağlanabilirlik algorıtmanın yakınsamasında ve niteliğinde bazı etkilere sahip olan en kolay yaklaşımı tercih ettik. Buna tek yönlü halka denir. Bu demektir ki her bir ada yalnızca bir diğerine göçmen gönderir ve her bir ada yalnızca bir diğerinden göçmen alır. Şekil 2.8'de dört kovanlı çöklu kovan modeli gösterilmiştir (Parpinelli *et al.* 2010).



Şekil 2.8. Çoklu kovan modeli (Parpinelli *et al.* 2010)

2.12. Paralel Hesaplama

1. Bir problemi, birden fazla bilgisayar ve/veya işlemci kullanarak çözmek denir. 2. Bir problemi çözmek için çok sayıdaki hesaplama kaynağını aynı anda kullanmaktır. 3. Program çok sayıda CPU üzerinde çalışmaktadır. 4. Problem eşzamanlı olarak çözülebilecek komutlara ayrılır. 5. Her bir parça kümesi komutlara ayrılır. 6. Her bir parçanın komutları aynı anda farklı CPU'lar üzerinde çalıştırılır (Dönmez 2013).

2.13. Paralel Programlama Modelleri

Bir paralel programlama modeli, paralel algoritmaları açıklayan bir yazılım teknolojileri kümesidir. Bu model, uygulamalar, diller, derleyiciler, kütüphaneler, iletişim sistemleri ve paralel giriş/çıkış alanlarını kapsar. Programcılar, kendileri ve uygulamaları için uygun bir model veya karma bir model seçip, uygulamalarını geliştirirler. Paralel modeller çok farklı şekillerde uyarlanırlar: klasik sıralı dillerden çağrılan kütüphaneler şeklinde, dil uzantıları şeklinde ya da tamamen yeni işleme modelleriyle. Bu modeller

kabaca ikiye ayrılırlar: Paylaşımlı hafıza sistemleri ve dağıtık hafıza sistemleri. Günümüzde bu iki sistem arasındaki çizgi oldukça bulanıklaşmıştır.

Sık kullanılan paralel programlama modelleri şöyledir:

1. Dağıtık Bellek: PVM, MPI.2.Paylaşımlı Bellek: Pthread, OpenMP. 3.Paylaşımlı Bellekli Dağıtık Sistem: MPI+OpenMP.4. Global Arrays, Co-Array Fortran, UPC, HPF, SHMEM, Occam, Linda, Cilk (Dönmez 2013).

2.14. MPI (Message Passing Interface)

MPI standartları belirlenmiş mesajlaşma kütüphanesidir. Arayüzü mesajlaşmalı programlama için geniş kullanıma sahip standarttır.

1. Noktadan-noktaya gönderir. 2. Senkron (synchronous). 3. Asenkron (asynchronous) 4. Bloke eden (blocking) 5. Bloke etmeyen (nonblocking) 6. Senkron (synchronous) 7. Asenkron (asynchronous). 8. Amaçlanarak oluşturulan bir kütüphane standardıdır. 9. Sadece rutinleri belirler, implementasyonla alakası yoktur (Blelloch and Maggs 1996).

2.14.1. MPI implementasyonlar

1. MPICH. 2. www-unix.mcs.anl.gov/mpi/mpich. 3. lam-mpi. 4.www.lam-mpi.org. 5.open mpi. 6. www.open-mpi.org. 7.Optimum mpi projesi. 8. Pardus pakedi mevcut. 9. ft-mpi, la-mpi, pacx-mpi. 10. fortran, C++, pyhton, Java vs bindingleri mevcut.

2.14.2. MPI derleme ve çalıştırma

1.ssh / rsh hazırlanır. 2. hesaplamada kullanılacak bilgisayarların listesi (hostfile) hazırlanır. 3.mpicc/mpicxx ile programlar derlenir. 4.mpirun ile çalıştırılır (Eraslan 2007).

3. MATERYAL ve YÖNTEM

3.1. Önerilen Algoritma

Bu çalışmada yeni bir evrimsel algoritma tanıtacağız. Bu algoritmanın birkaç girişi ve çıkışı olacak. Bu algoritmayı C++ program dilinde yazacağız. Programı hem seri ve hem paralel olarak çalıştıracağız. Programı paralel hale getirmek için mpi komutlarını kulanacağız. Bu tez içerisindeki çizgelerin tamamı dot dili ve graphviz programı kullanılarak görselleştirilmiştir ve kullanılmıştır. Hazırlanan bu programlar **EK 1** ve **EK 2**'de gösterilmiştir.

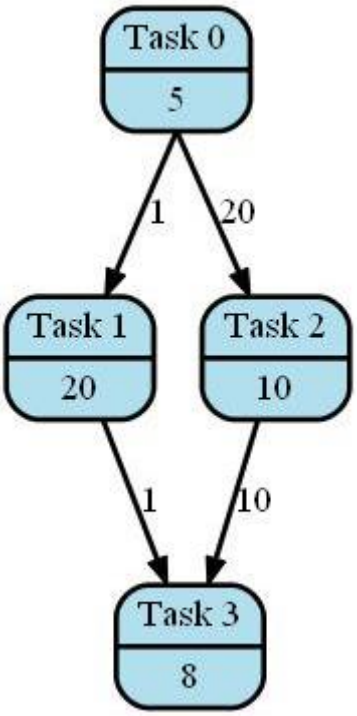
3.1.1. Girişler

1. Dosya: Programımız bir görev çizgesini dosya şeklinde alacak. Bu dosya DAG'ın (görev çizgesi) kaynağı (bu DAG'ı aldığımız makalenin adı) kaç tane görev olduğunu, kaç tane ok olduğunu, görevlerin birbirlerine bağlı olduğunu, her bir görevin yürütme zamanı ve her iki görevin arasındaki okları ve bu okların ağırlığını (iletişim malıyeti) gösterecek. Bu dosyaya bir örnek Şekil 3.2'de gösterilmektedir, bir dosyayı örnek olarak şu şekilde anlatıyoruz:

- Görevlerin sayısı okların sayısı
- Her görevin yürütme zamanı
- İki görevin arasında olan iletişim maliyeti

Bir yönlendirilmiş çevrimsiz çizgesi (DAG)'ne örnek Şekil 3.1'de gösterilmektedir. Bu DAG'ın dosyası Şekil 3.2'de anlatılmaktadır. Şekil 3.2'de ilk sırada DAG'ın kaynağı (DAG' ın olduğu makalenin adı), ikinci sırada ve ilk sütun "4" görevlerin sayısı; ikinci sıradaki ikinci sütun "4" okların sayısıdır. Ayrıca, üçüncü sıradan altıncı sıraya kadar her görevin yürütme zamanını gösterilmektedir. Mesela "0 5" sıfırıncı görevin yürütme zamanının beş olduğunu gösterir. Altıncı sıra ve sonrasındaki her bir görevin diğer

görevle arasındaki oklar ve bu okların ağırlıkları (iletişim maliyetleri) gösterilmektedir. Örneğin "0 1 4" anlamı: Sıfırıncı görevden birinci göreve olan okun ağırlığı 4'tür. Tezimizde toplamda 11 tane yönlendirilmiş çevrimsiz çizgesi (DAG) bulunmaktadır ve bu DAG'ların hepsinin birer dosyası olacaktır.



Şekil 3.1. 4 görevden oluşan görev çizgesi (DAG) (Hwang *et al.* 2008)

//Comparison of Multiprocessor Task Scheduling Algorithms with Communication Costs, Figure2

```
4    4
0    5
1    20
2    10
3    8
0    1    1
0    2    20
1    3    1
2    3    10
```

Sekil 3.2. 4 görevden oluşan görev çizgesine (Şekil 3.1) ait dosya

2. İşlemcilerin sayısı : Programımızın üzerinde bulunacağı işlemcilerin adedidir.

3. Popülasyonun büyüklüğü: Kromozomların sayısını gösteriyor.

4. İterasyon sayısı: Yeni kuşak çoğalma, çaprazlama ve mutasyon işlemlerinden sonra tanımlanmakta ve bir sonraki kuşağın ebeveynleri olmaktadırlar. Süreç yeni kuşakla çoğalma için belirlenen uygunluk ile devam eder. Bu süreç, önceden belirlenen kuşak sayısı kadar veya bir hedefe ulaşılıncaya kadar ya da başka bir durdurma kriteri sağlanana kadar devam eder. İstenen hassasiyet derecesine göre de maksimum iterasyon sayısı belirlenebilmekte ve iterasyon bu sayıya ulaştığında döngü durdurulabilmektedir. Durdurma kriteri iterasyon sayısı olabileceği gibi hedeflenen uygunluk değeri de olabilmektedir (Tang *et al.* 2006).

5. Göç oranı: Her bir göç olayında göç edecek bireylerin sayısıdır.

6. Çaprazlama Olasılığı: Çaprazlamanın amacı, mevcut iyi kromozomların özelliklerini birleştirerek daha uygun kromozomlar yaratmaktır. Kromozom çiftleri P(c) olasılığı ile çaprazlamaya uğramak üzere seçilirler. Çaprazlamanın artması, yapı bloklarının artmasına neden olmakta fakat aynı zamanda bazı iyi kromozomların da bozulma olasılığını arttırmaktadır (Yang and Yu 2007).

7. Mutasyon Olasılığı: Mutasyonun amacı populasyondaki genetik çeşitliliği korumaktır. Mutasyon P(m) olasılığı ile bir kromozomdaki her bitte meydana gelebilir. Eğer mutasyon olasılığı artarsa, genetik arama rastsal bir aramaya dönüşür. Fakat bu aynı zamanda kayıp genetik malzemeyi tekrar bulmada yardımcı olmaktadır (Gökay ve Çağatan 2002).

3.1.2. Çıkışlar

1. Programın bitiş zamanı (FT) (best latency)

2. Son görevin uygulanışının aldığı süre (makespan)
3. Duvar saati zamanı (wall clock time)
4. Her görevin başlangıç ve bitiş zamanı
5. Her görevin öncelleri ve ardılları (predecessors ve successors)
6. Komşuluk matrisi (adjacency matrix)
7. Her görevin yürütme zamanı (execution times)
8. Görevlerin üzerinde uygulanacakları işlemciler (task bounds to processors)
9. Görev çizgelerinin görev sayısı
10. Görevlerin bağlı olduğu diğer görevler
11. Görev grafiklerinin ok sayısı (edges of the task graph)

3.1.3. Seri versiyon

Bu yöntemde her kromozomun büyüklüğü görevlerin sayısının iki katıdır. Örneğin: eğer 4 tane görev olursa kromozomun büyüklüğü 8 olacak. Her kromozom iki eşit parçaya ayrılır. İlk parçada görev sayısını n kabul edersek, genlere verilecek sayılar 0'dan başlamak üzere en büyük sayı n-1 olacak ve sayılar tekrarsız olmak üzere rastgele atanacaktır. Benzer şekilde, ikinci parçada işlemci sayısını n dersek genlere atılacak sayılar yine aynı şekilde 0 ile n-1 arasında olacaktır ancak bu defa sayılar tekrarlı olabilirler. Mesela, 4 görev ve 2 işlemciyle uygulamak istediğimizde kromozom şu şekilde olacaktır: 3021|0110.

Yani üçüncü görev sıfırıncı işlemci, sıfırıncı görev birinci işlemci, ikinci görev birinci işlemci, birinci görev sıfırıncı işlemci üzerinde uygulanacak. Kromozomların değerlendirilmesi ise şu şekilde olur: ilk sıradaki görevin zamanlaması için bu görevin öncel görevlerinin de zamanlama olması gerekir eğer zamanlama olmazsa ikinci sıradaki göreve gidilir eğer bu görevin öncel görevleri zamanlamışsa bu görev zamanlanacaktır. Yani bir görevin öncel görevi eğer zamanlanmamışsa o görev zamanlanmayacaktır. Bu işlem tüm görevler zamanlanıncaya kadar sürecektir. Kromozom sayısı nüfusun büyüklüğü kadardır. İlk adımda bu algoritma Başlangıç Popülasyonun her kromozomu için bitiş zamanını (FT) bulacaktır. Sonra iterasyon

sayısı kadar yeni Popülasyon üretecek ve her yeni popülasyonda çaprazlama, mutasyon, yerleştirme, Turnuva Seçim işlemleri yapacak. Sonunda en kısa bitiş zamanına sahip olan kromozom bizim cevabımız olacaktır. Programımızın fonksiyonları aşağıda tanıtılmıştır:

Çizelge 3.1. Seri versiyonun fonksiyonları

GABasedSerialScheduling ()	Yapıcı Fonksiyon
~GABasedSerialScheduling ()	Yıkıcı Fonksiyon
CreateTaskGraphAndDataStructures (string)	Görev çizgesi (DAG) ve veri yapıları oluşturma (girdi dosyasının adı)
Print ExecutionTimes Of Tasks ()	Görevlerin yürütme zamanlarını ekrana yazdırma
PrintEdges()	Okları ekrana yazdırma
PrintAdjacencyMatrix():	Bir görevden diğer bir göreve ok olursa bir matrix de ekrana yazdırma
FindPredecessors()	Her görevin öncül görevlerini bulma.
PrintPredecessors ()	Her görevin öncül görevlerini ekrana yazdırma.
FindSuccessors()	Her görevin ardıl görevlerini bulma.
PrintSuccessors()	Her görevin ardıl görevlerini ekrana yazdırma.
ObtainSchedulingParameters (int,int,int,double,double)	Çizelgeleme parametrelerini gösterme ilk parametre işlemci sayısı, ikinci parametre popülasyonun büyüklüğü, üçüncü parametre iterasyon sayısını, dördüncü parametre çaprazlama olasılığı, beşinci parametre mutasyon olasılığı, altıncı parametre göç olasılığıdır
DynamicallyAllocateSpaceForPopulationAndCostArrays()	Popülasyon ve maliyet dizileri için dinamik yer tahsis etmek.
Perform Scheduling()	Çizelgelemeyi uygulamak
PrintTheBestChromosome()	En iyi kromozomu ekrana yazdırmak.
PrintStartFinishTimesOfTasks()	En iyi kromozomun görevlerinin başlangıç ve bitiş zamanlarını ekrana yazdırmak
PrintOrderedProcessorBindings()	En iyi kromozomun görevlerinin uygulanacağı işlemleri ekrana yazdırmak.
DrawGanttChart():	Gantt grafığı ekrana yazdırmak
InitializePopulation():	Başlangıç Popülasyon belirlemek
UpdateTheBestScheduleAndLatency()	En iyi çizelgelemeyi ve FT(bitiş zamanı) güncellemek
ApplyTournamentSelectionOperator()	Turnuva Seçim işlemi uygulamak
ApplyCrossoverOperation()	Çaprazlama işlemini uygulamak.
ApplyOnePointCrossoverWithinTheFirstHalf(int,int)	Kromozomun ilk parçasında tek noktalı çaprazlama uygulamak

Çizelge 3.1 (devam)

ApplyTwoPointCrossoverWithinTheSecondHalf(int,int)	Kromozomun ikinci parçasında iki noktalı çaprazlama uygulamak
ApplyMutationOperator()	Mutasyon işlemi uygulamak
ApplyWorstReplacementOperator()	Kötüyü yerleştirmek işlemi uygulamak
EvaluatePopulation(int*,int*)	Popülasyonu ölçmek
FindAReadyTaskAndScheduleIt(int*)	Bir hazır ve zamanlanmış görev bulmak
ScheduleTask(int)	Görevi zamanlamak.
GetBindingInformationFromTheChromosome(int*)	Kromozomdan bağlayıcı bilgi almak.
IfAllTasksHaveBeenScheduled()	Tüm görevler zamanlanmış olup olmadığını test etmek
IfAllPredecessorsScheduled(int)	Görevlerin tüm öncellerinin zamanlanmış olup olmadığını test etmek
EvaluateAChromosome(int*)	Kromozomu ölçmek
Generate A Random Integer (int,int)	Rasgele bir tamsayı üretmek
GenerateARandomNumberBetweenZeroAndOne():	Sıfır ve bir arasında rastgele sayı üretmek

3.1.4. Paralel versiyon

Bu yöntemde her kromozomun büyüklüğü görevlerin sayısının iki katıdır. Örneğin: eğer 4 tane görev olursa kromozomun büyüklüğü 8 olacak. Her kromozom iki eşit parçaya ayrılır. İlk parçada görev sayısını n kabul edersek, genlere verilecek sayılar 0'dan başlamak üzere en büyük sayı n-1 olacak ve sayılar tekrarsız olmak üzere rastgele atanacaktır. Benzer şekilde, ikinci parçada işlemci sayısını n dersek genlere atılacak sayılar yine aynı şekilde 0 ile n-1 arasında olacaktır. Ancak bu defa sayılar tekrarlı olabilirler. Mesela, 4 görev ve 2 işlemciyle uygulamak istediğimizde kromozom şu şekilde olacaktır: 3021|0110.

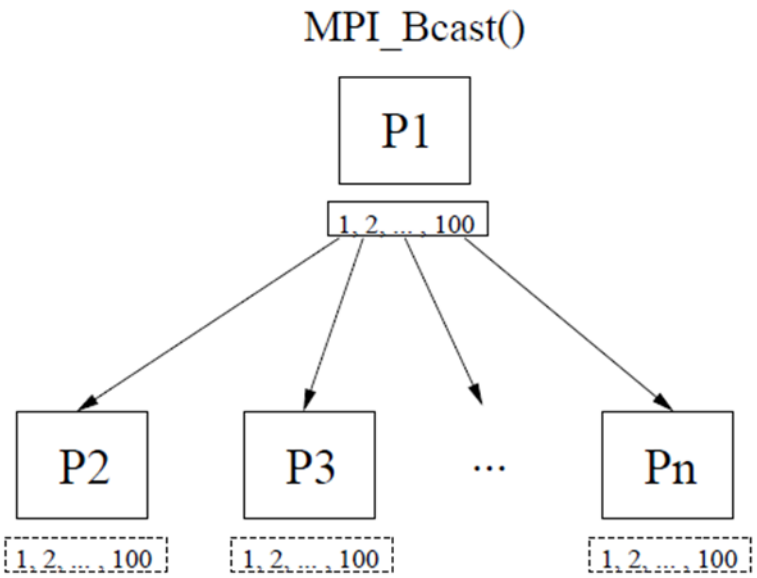
Yani üçüncü görev sıfırıncı işlemci, sıfırıncı görev birinci işlemci, ikinci görev birinci işlemci, birinci görev sıfırıncı işlemci üzerinde uygulanacak. Kromozomların değerlendirilmesi ise şu şekilde olur: İlk sıradaki görevin zamanlaması için bu görevin öncel görevlerinin de zamanlama olması gerekir eğer zamanlama olmazsa ikinci sıradaki göreve gidilir eğer bu görevin öncel görevleri zamanlamışsa bu görev

zamanlanacaktır. Yani bir görevin öncel görevi eğer zamanlanmamışsa o görev zamanlanmayacaktır. Bu işlem tüm görevler zamanlanıncaya kadar sürecektir. Her işlemcinin üzerinde farklı nüfus olacak yani bu nüfuslar iterasyon sayısı kadar farklı yeni nüfuslar üretecekler (çaprazlama, mutasyon, yerleştirme, turnuva işlemlerle). Bu yöntemde paralel çoklu kovan modelini kulanacağız. Yani her işlemci bir kovan olacak ve bu işlemciler halka topolojiyle birbirine bağlanacak. Bu modelde göç oranı adında bir değişken olacaktır. Örnek olarak bu değişken 5 olduğu zaman her işlemci beşinci nesile geldiği zaman (genelikle iterasyon sayısı göç oranına bölündüğu zaman kalan sıfır olursa iterasyon sayısı %göç oranı=0) yani her beşinci nesilde her bir işlemcinin en iyi kromozumu (bitiş zamanı en az olan) ve en kötü kromozomu (bitiş zamanı en çok olan) belirlenmiş olur ve yine her beşinci nesilde halka topolojide her işlemcinin en iyi kromozomu başka işlemcinin en kötü kromozomuna mpi komutlarıyla mesaj gönderir ve iterasyon bitene kadar bu işlemler devam eder. Sonunda en iyi kromozom sıfırıncı işlemciye gönderilir ve bu kromozom bizim cevabımız olur. Paralel fonksiyonlarin çoğu seri fonksiyonlar gibidir. Farklı olanlar aşağıda gösterilmiştir:

Çizelge 3.2. Paralel versiyonun fonksiyonları

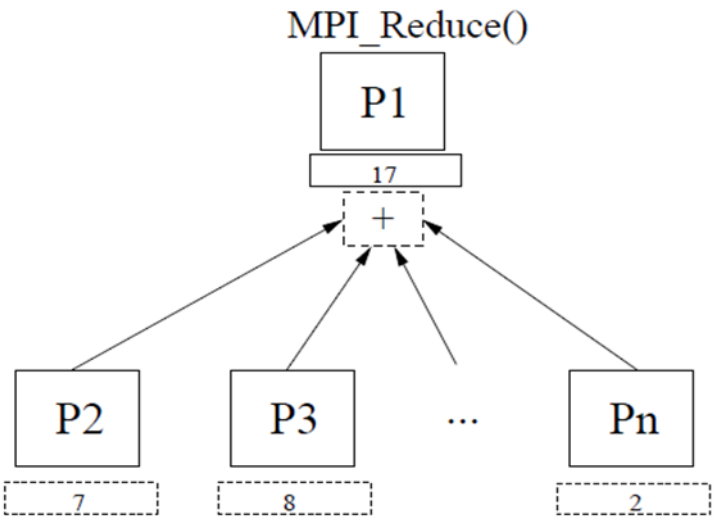
MPI_Status status	MPI_Status yapılar mesajla ilgili verileri döndürmek için mesaj alıcısının (message receiving) işlevleri tarafından kullanılır
MPI_Init(&argc,&argv):	1. MPI çevresini hazırlar. 2. Her MPI programında mutlaka bulunur.3. MPI programı içinde sadece bir kere çağrılır.
MPI_Comm_size(mpi_comm_world,&size)	Haberleşme dünyası ile ilişkili grubun boyutunu belirtir. (işlemci sayısı).
MPI_Comm_rank(mpi_comm_world,&myRank)	Sürecin haberleşme dünyasındaki sırasını belirtir. 0 dan başlar ardışık ve tam sayıdır
MPI_comm_world	MPI süreçlerinin tamamını kapsayan MPI_comm_world değişkeni cevap verir
MPI_Wtime	Arama işlemcisi üzerinde geçen zamanı döndürür
MPI_Send	Bir mesaj gönderir.
MPI_Recv	Bir mesaj alir.
MPI_Finalize():	1.MPI çalışma çevresini sonlandırır. 2. Bu fonksiyondan sonra çağrılan MPI fonksiyonları haliyle geçersizdir.

MPI_Bcast: Süreçlere gönderilecek veri için "Birisinden-Herkese" (One-to-All) yayımlama işlemi, MPI'da MPI_Bcast fonksiyonu ile yapılır.



Şekil 3.3. Bir düğümden diğer düğümlere yayımlama (Parpinelli *et al.* 2010)

MPI_Reduce: Süreçlerden elde edilen veri için Herkesden-Birisine indirgeme işlemi, MPI'da MPI_Reduce fonksiyonu ile yapılır.



Şekil 3.4. Düğümlerdeki verinin tek bir düğüm üzerindeki tek bir veri olarak indirgenmesi (Parpinelli *et al.* 2010)

Bu çalışmada yerleştirme işlemi mutasyon ve çaprazlama işlemlerinden sonra yeni nüfus ortaya çıkaracak şekilde olacaktır. Bu nüfusun tüm kromozomlarını önceki nüfusun kromozomlarıyla kıyaslanacak ve eger yeni nüfustan bir kromozomun bitiş zamanı önceki nüfusun ilgili kromozomun bitiş zamanından daha kısa ise, bahsettiğimiz bu eski nüfusun yerine kopyalanacaktır. Bu algoritmamızda mutasyon ve çaprazlama işlemlerini kromozomun farklı parçalarında kulandık. Yani her kromozomu iki parçaya böldük ve her parçada farklı mutasyon ve çaprazlama işlemleri uyguladık. İlk parçada bir noktalı çaprazlama (permutation encoding) ve takas mutasyon (swap mutation) kulandık.

3.1.5. Bir noktalı çaprazlama (permutation encoding)

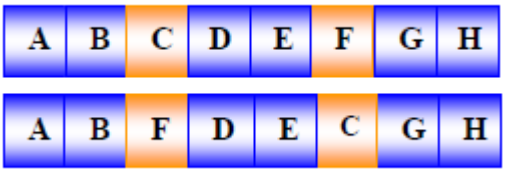
Permütasyon kodlama biraz farklı ve oldukça karmaşıktır. Tek noktalı çaprazlama halinde döl kromozomun ilk kısmı bir ebeveynden gelir, kalan kısmı diğer ebeveyndeki dölün henüz kullanılmamış genleri tarafından tamamlanır. Aşağıdaki örnekte detaylı olarak gösterilmiştir (Anonim 2013a).

Ebeveyn 1	**1 2 3 4 5 6 7 8 9**
Ebeveyn 2	**5 7 2 4 9 6 3 8 1**
döl 1	**1 2 3 4 5 7 9 6 8**
döl 2	**5 7 2 4 9 1 3 6 8**

Şekil 3.5. Bir noktalı çaprazlama (Anonim 2013a)

3.1.6. Takas mutasyonu (swap mutation)

İki genin kromozom üzerindeki iki bloğu rastgele seçilir ve pozisyonları değiştirilir**.** Bir takas mutasyonuna örnek olarak aşağıdaki şekilde verilmiştir.

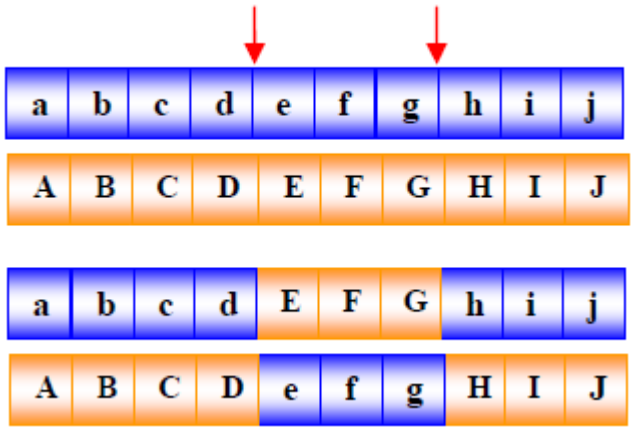


Şekil 3.6. Takas mutasyon (swap mutation) (Ghader *et al.* 2008)

İkinci parçada iki noktalı çaprazlama (N-Point Crossover) ve ekleme mutasyonu (insertion mutation) yaptık.

3.1.7. İki noktalı çaprazlama (N-Point Crossover)

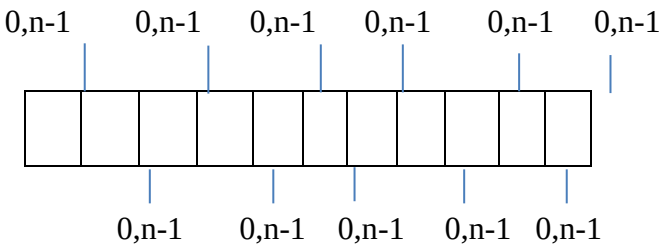
İki noktalı çaprazlama, ebeveynin canlı kordonlarındaki seçilmis iki noktayı gerektirir. İki nokta arasındaki tüm genler çaprazlanı ve iki yavru kromozumu oluşturur. İki noktalı çaprazlama Şekil 3.7'de gösterilmiştir.



Şekil 3.7. İki noktalı çaprazlama (Ghader *et al.* 2008)

3.1.8. Ekleme mutasyonu (insertion mutation)

İşlemci sayısını n kabul edersek, genlere verilecek sayılar 0'dan başlamak üzere en büyük sayı n-1 olacak ve sayılar tekrarlı olmak üzere rastgele atanacaktır. Ekleme mutasyonu (insertion mutation) Şekil 3.8'de gösterilmiştir (Ghader *et al.* 2008).

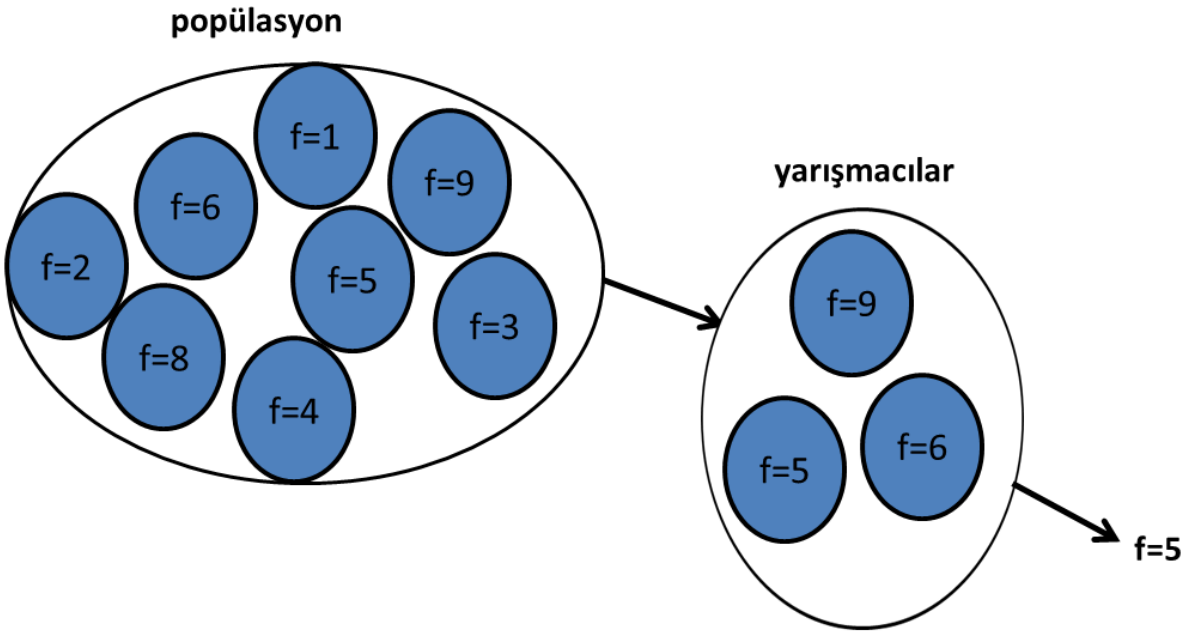


Şekil 3.8. Ekleme mutasyonu (insertion mutation) (Ghader *et al.* 2008)

3.1.9. Turnuva seçimi (tournament selection)

Turnuva seçiminde (tournament selection) 3 rastgele kromozom seçildi ve sonrasında bu 3 kromozomdan bitiş zamanı en az olan doğrudan sonraki popülasyona kopyalandı. Bu işlem popülasyon büyüklüğu boyunca devam ettirilip, sonunda yeni popülasyon ortaya çıkarıldı.

Turnuva seçiminin şeması aşağıda gösterilmiştir.



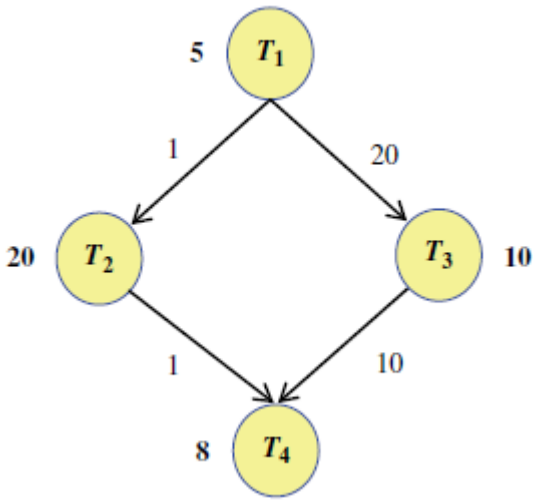
Şekil 3.9. Turnuva seçimi (Ghader *et al.* 2008)

4. ARAŞTIRMA BULGULARI

Bu bölümde diğer algoritmaların görev-grafik (DAG) kısmından bahsedeceğiz.

4.1. HLFET (Highest Level First with Estimate Times) Algoritma

Bu algoritmada öncelikle her düğümün çıkış düğümüne olan ağırlık düğümlerinin maksimum toplamını bulacağız. En yüksek değere sahip düğüm en önceliğe sahiptir. Düğümler öncelliklerine göre sıralandırıldıktan sonra, değerlerine göre yüksekten aza doğru bir listed sıralanırlar. Düğümlerin işlemcilere tahsisi bu listeye göre yapılır. Şekil 4.1'de bir HLFET algoritma örneği gösterilmiştir.



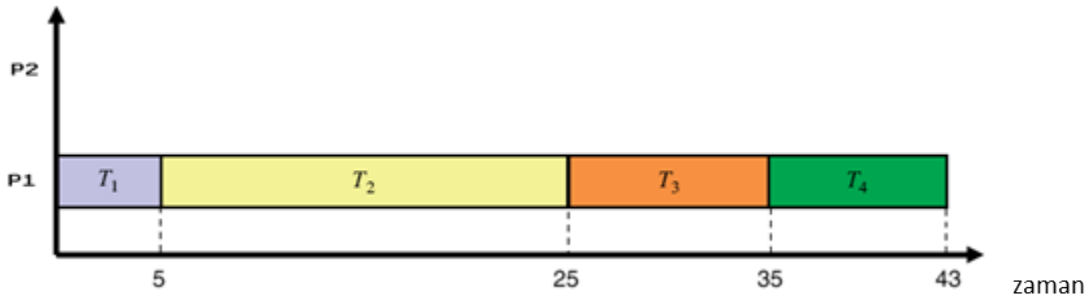
Şekil 4.1. Dört düğümlü bir DAG örneği (Hwang *et al.* 2008)

Şekil 4.1'de gösterilen görev çizgesi için düğümlerin toplam ağırlıkları ve öncelikleri Çizelge 4.1'de gösterilmiştir.

Çizelge 4.1. Düğümlerin toplam ağırlık ve öncelikleri

Düğümlerın öncelikleri	Toplam ağırlıkları	Düğümlerin numarası
1	Max(W(T1)+W(T2)+W(T4) , (T1)+W(T3)+W(T4))=max(33,23)=33	T1
2	W(T2)+W(T4)=28	T2
3	W(T3)+W(T4)=18	T3
4	W(T4)=8	T4

Şekil 4.1'e ait Gantt grafiği Sekil 4.2 'de, iki işlemci üzerinde gösterilmiştir (Hwang *et al.* 2008).



Şekil 4.2. HLEFT algoritmasıyla hazırlanmış Şekil 4.1' deki DAG için Gantt grafiği (Hwang *et al.* 2008)

4.2. MCP (Modified Critical Path) Algoritma

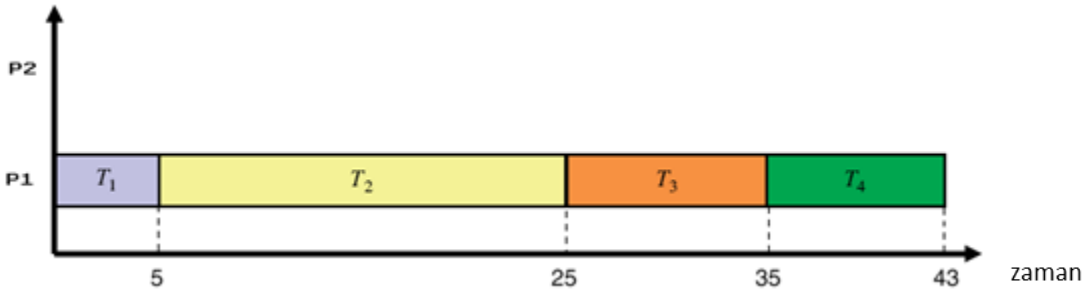
MCP algoritması bir düğümün geç başladığı zamana göre tasarlanmış algoritmadır. Düğümün geç başladığı zaman ALAP'la gösterilir. ALAP'la bulunan her düğümün ALAP'la bulunmuş geç başlanılan zamanlar Şekil 4.1 için aşağıda gösterilmiştir.

$ALAP(n_i)=len(cp(n_i))-tl(n_i)$

Çizelge 4.2. Şekil 4.1 için ALAP'la bulunmuş geç başlanılan zamanlar

Ti	ALAP
1	0
2	24
3	25
4	45

Başlangıç zamanı en az olan düğümler işlemcilere tahsis edlir. Şekil 4.1'deki DAG zamanlaması için MCP algoritmasıyla hazırlanmış Gantt grafiği Şekil 4.3'de gösterilmiştir (Mccreary *et al.* 1994).



Şekil 4.3. Şekil 4.1'deki DAG zamanlaması için MCP algoritmasıyla hazırlanmış Gantt grafiği (Mccreary *et al.* 1994)

4.3. DLS (Dynamic Level Scheduling) Algoritma

Düğümlerin zamanlama önceliği her adımda dinamik olarak hesaplanır. Bu algoritmada düğümün önceliği DL adlı karakter tarafından belirlenir. DL, ni düğümü ve pj işlemci üzerinde DL(ni,pj) şeklinde gösterilir.

DL'nin özellikleri aşağıdaki terimler kullanılarak belirlenir:

İlk terim tüm düğümlerin SL'idir.

Düğümlerden çıkış düğümüne toplam maliyeti en yüksek olanın maliyetine SL denir. İkinci terim ST, (start time) ni düğümü pj işlemcinin üzerindedir. DL formülü aşağıda

gösterilmiştir:

$$DL(ni,pj)=SL(ni)-ST(ni,pj)$$

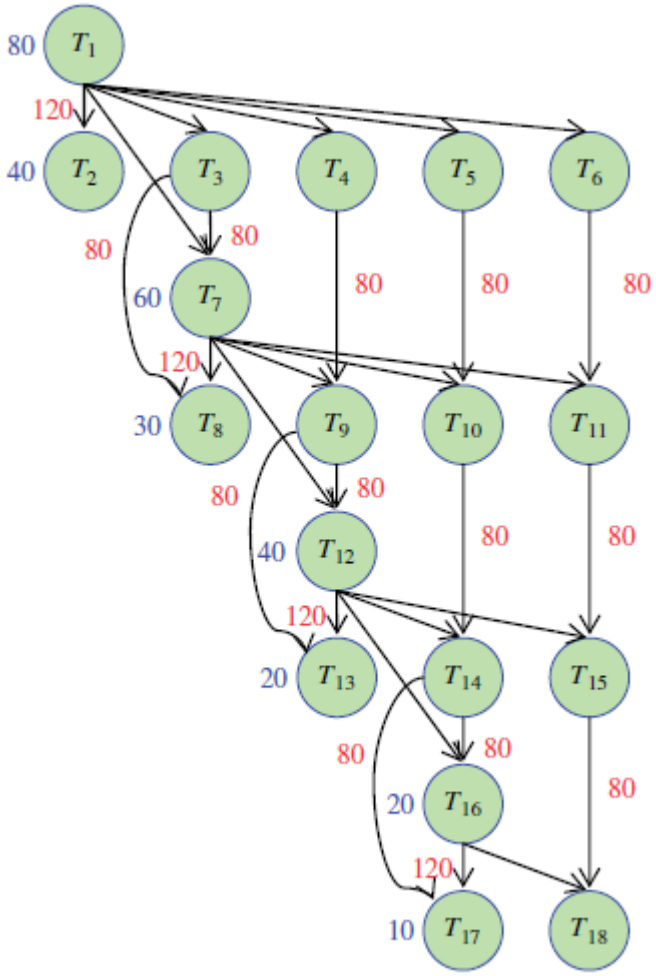
Her zamanlama aşamasında DLS algoritması hazır düğümlerdeki ve işlemcilerdeki DL miktarını buluyor. Sonrasında en yüksek DL'ye sahip işlemciler ve düğümler birbirleriye eşleşir (Kwok and Ahmad 1999).

4.4. DCP (Dynamic Critical Path) Algoritma

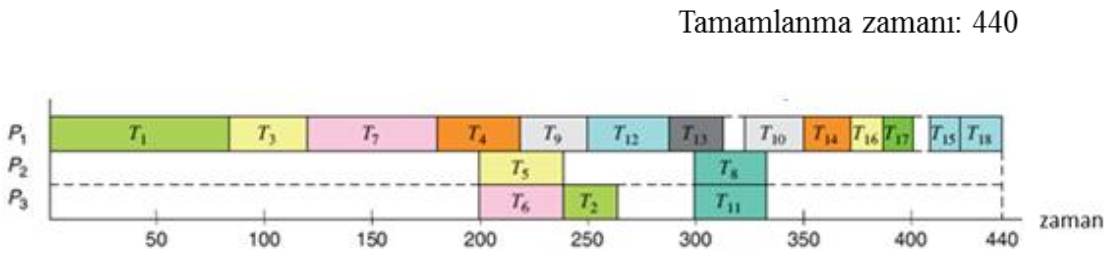
Bu algoritma bir sezgisel yöntemi liste zamanlamasına tarif ediyor. Bu yöntemin iki özelliği vardır:

1. Bu algoritma dinamik bir yöntemle düğümleri işlemcilerle eşleştirmek için kullanılıyor. Bu algoritma düğümlerin önceliklerini dinamik olarak ve her zamanlama adımında DCP özelliğini kullanarak belirliyor. Yani zamanlama tekdüze azalmaktadır. Bir düğümün kritik yolu: Çıkış düğümlerine karşılık gelen düğümler arasındaki tüm yollardan en büyük toplam hesaplama maliyeti ve iletişim maliyetine sahip yoldur. Kritik yol zamanlama aşamasında ise, buna dinamik kritik yol denir. DCP'yi her zamanlama aşamasında azaltmak için seçilen düğümün ardılının zamanlamasının olmaması gerekir.

2. Düğümler ile eşleşmiş olan işlemciler kontrol edilir. Yani en uygun işlemci düğümlere tahsis edilir (Kwok 1999). DCP ve 3 işlemcinin zamanlama algoritması kullanılarak hazırlanan Şekil 4.4'e ait Gantt grafiği Şekil 4.5'te gösterilmiştir (Hwang *et al.* 2008).



Şekil 4.4. 18 düğümlü bir DAG örneği (Hwang *et al.* 2008)



Şekil 4.5. Şekil 4.4'e ait Gantt grafiği (Hwang *et al.* 2008)

4.5. HEFT (Heterogeneous Earliest Finish Time) Algoritma

Bu algoritmada bir sezgisel yöntem sunulmuştur. Bu yönteme en yakın bitten dağıtılan zamana HEFT algoritması denir. Bu algoritmada düğümlerin öncelikleri düğümlerin düşük miktarına (bl) göre belirlenir. Düğümlerin listesi düşük miktarlarının (bl) artışı şeklinde sıralanmaktadır. Düğümler düşük miktarlarını azaltacak işlemciler ile eşleştirilirler. Düğüm ekleme yöntemi düğümleri diğer düğümlere bağlı olarak inceler ve o düğümü işlemcilerin boş yerlerine bırakır.

Bu yöntemin amacı zamanlama uzunluğunu azaltmaktır. Eğer işlemcilerde boş yer olmazsa, düğümler en kısa başlangıç zamanını sağlayacak işlemciye tahsis edilir (Kwok and Ahmad 1999).

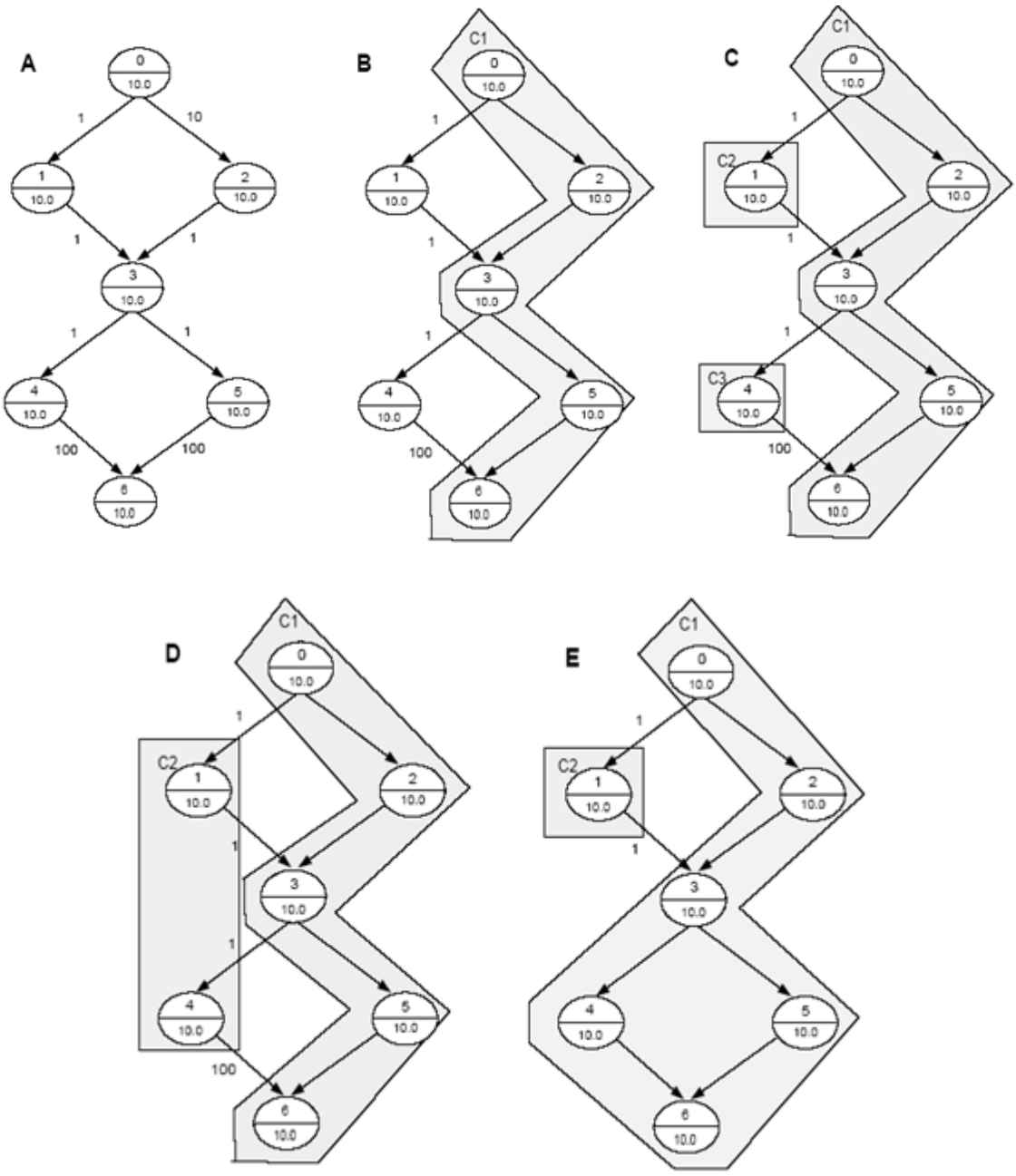
4.6. LC (Linear Clustering) Algoritma

Bu algoritma bir doğrusal küme algoritmasıdır. Bu algoritmada şu adımlar gerçekleşir:

1. Maliyet fonksiyonu tüm okları kontrol ederek en uzun yolu belirler. Maliyet fonksiyonu bu algoritmada hesaplama maliyeti ve kritik yolun toplamıdır. Mevcut düğümler kritik yol (CP) boyunca aynı kümede bulunacaklar. Aynı kümede bulunan düğümler arasındaki iletişim maliyeti sıfıra eşittir.
2. Kritik yol üzerindeki oklar kontrol edilmiş oklardır.
3. Tüm oklar kontrol edilene kadar 2. ve 3. adımlar tekrar edilir.

Yukarıdaki adımlar bittikten sonra bir dizi doğrusal küme elde edilecektir. Kümeleme adımından sonra algoritma iki veya daha fazla kümeyi birleştirmek için çalışır. Bu birleştirme İşleminin amacı işlemciyi dengeye ulaştırmak ve işlemciler arasındaki iletişim maliyetini azaltmaktır. Kümeleme aşaması tamamlandıktan sonra, görev çizgesi (DAG) sanal mimari grafik ile (VAG) değiştirilecektir. VAG, DAG'ın çoklu-işlemci mimarisini gösterir. Sonra DRT yapılır. DRT, maksimum yayılan ağacın VAG'ıdır. Gerekli kaynaklar hakkında bilgi edinmek için önceki grafik mimarisine geri gitmek

zorundayız. Bu iş bir maksimum yayılan ağaç algoritmasıyla bir DST oluşturarak yapılabilir. Ana fikir kümelerinin işlemcilerle eşlenmesi için DRT'ten DST'ye bir subgraph bulunmalıdır. Yani, zamanlama uzunluğu azaltılabilir (Kwok and Ahmad 1999).



Şekil 4.6. Kümelenmenin LC'ye göre zamanlama algoritması (Kwok and Ahmad 1999)

4.7. EZ (Edge Zeroing) Algoritma

Bu algoritma küme entegrasyonu ölçüsü olarak okların ağırlığını kulanır. Bu algoritmada zaman uzunluğunu azaltmak için iki küme birleştirilir. EZ algoritma aşağıdaki adımlarda açıklanır:

1. Bütün okların ağırlıkları azalan olarak düzeltilir.
2. Eğer zamanlamanın artmadığı bir durumda bir okun ağırlığı tüm okların ağırlığından yüksek olursa o okun ağırlığı sıfır olacak. İki küme birbirine birleştirilir ve sonuçta yeni bir küme oluşacaktır. Yeni kümede tüm oklar sıfır olacaktır. Bir kümede düğümlerin sırası bl (düşük seviye)'ye dayanarak listelenecektir.
3. İkinci adım tüm oklara kaydırma yapılarak tekrarlanacaktır (Vanneeschi *et al.* 2011).

4.8. MD (Mobility Directed) Algoritma

Bu algoritma yeni bir özellik olan Göreli dinamiği (relative mobility) tüm düğümler için tanımlıyor. Düğümlerin göreli dinamiği, aşağıdaki formül kullanılarak hesaplanır:

$$mobility(n_i) = \frac{ALAP(n_i) - ASAP(n_i)}{W(n_i)}$$

$ASAP(n_i)=tl(n_i)$ $\quad ALAP(n_i) = len(cp(n_i)) - bl(n_i)$

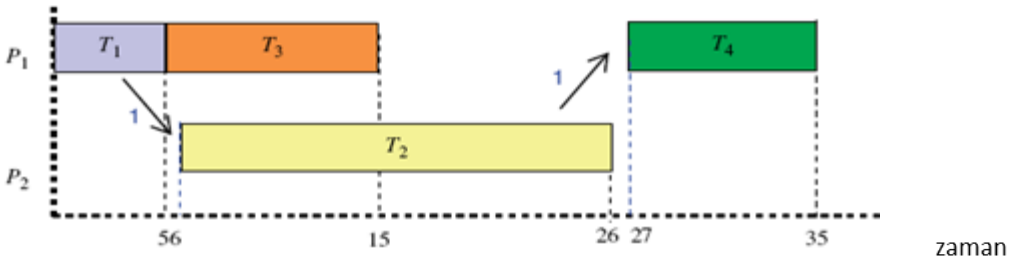
$W(n_i)$ düğümlerin ağırlığıdır. $Len(cp(n_i))$ kritik yolun uzunluğudur.

Algoritma aşağıdaki gibi çalışır:

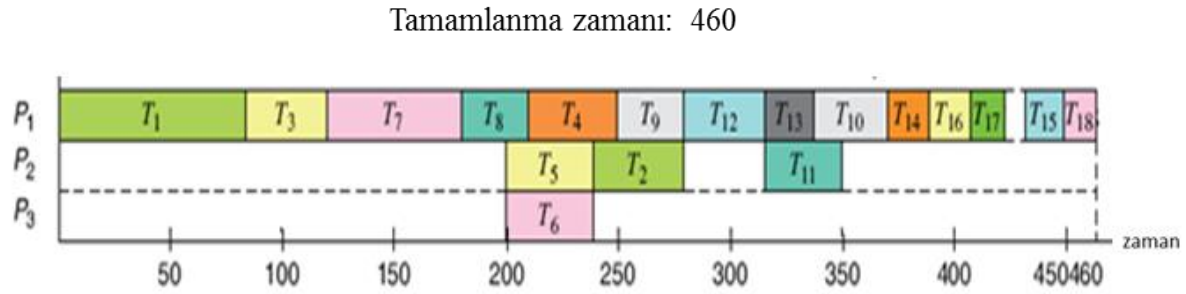
Başlangıçta, bütün düğümlerin göreli dinamikleri hesaplanır. Farz edelim ki, L hiç kontrol edilmemiş set oklarını göstersin. İlk olarak, tüm düğümler L setinde olacak ve

yine farz edelim ki; L' göreli dinamikleri en az olan set düğümlerini gösterir. Ayrıca, n_i düğümü L' setindendir ve bu düğümden once gelen bir düğüm yoktur. Eğer n_i düğümünün birinci işlemciyle eşleşme ihtimali varsa, bu düğüm birinci işlemciye tahsis edilecektir. Aksi takdirde, bir sonraki işlemciye gönderilecek ve ilk işlemciyle ona uygun olan bir düğüm eşlenir. Burada uygun bir işlemci şu demektir: n_i düğümü bu işlemciye ve bu işlemcinin diğer düğümlerinin arasına eklenebilsin. bir düğümü bir işlemciye eklemek için işlemcideki tüm açık pozisyonlarda ASAP(n_i) ve ALAP(n_i)'ler kontrol edilir. Eğer işlemci tahsisinde herhangi bir boş yer bulunmazsa, n_i düğümü işlemcinin sonuna tahsis edilecektir.

Örneğin, n_i düğümü p_j işlemciye zamanlandı, bu durumda, düğümlerin ağırlıklarının sıfıra değiştirilmesi için n_i düğümüne yolu olmalı ve p_j üzerinde çalışmalıdır. Eğer n_i düğümü n_j düğümünden önce zamanlanırsa ve p_j üzerinde çalışırsa, n_i düğümunden n_j düğümüne bir ok çekiyoruz. Benzer şekilde, eğer n_i düğümü n_j düğümünden sonra p_j işlemci üzerinde zamanlanırsa, n_j düğümünden n_i düğümüne bir ok çekiyoruz. Bu işlemler sonucu hiç halka oluşmaz. Daha sonra göreli dinamiği her düğüm için tekrar hesaplanır. n_i düğümü L setten kaldırılır ve yukarıdaki adımlar boş olmayan L setleri sayısınca tekrarlanır (Kwok and Ahmad 1999). Şekil 4.1, 4.4 için MD algoritmasıyla hazırlanmış iki Gantt grafiği Şekil 4.7 ve 4.8'de gösterilmiştir (Hwang *et al.* 2008).



Şekil 4.7. Şekil 4.1'de gösterilen görev çizgesi üzerinde 2 işlemci kullanarak, MD algoritması kullanılarak hazırlanmış bir Gantt grafiği (Hwang *et al.* 2008)



Şekil 4.8. Şekil 4.4'te gösterilen görev çizgesi üzerinde 3 işlemci kullanarak MD algoritması ile hazırlanmış Gantt grafiği (Hwang *et al.* 2008)

4.9. DSC (Dominant Sequence Clustering) Algoritma

Bu algoritmayı göstermek için, öncelikle bazı gerekli kısıtlamalardan bahsetmek gerekir:

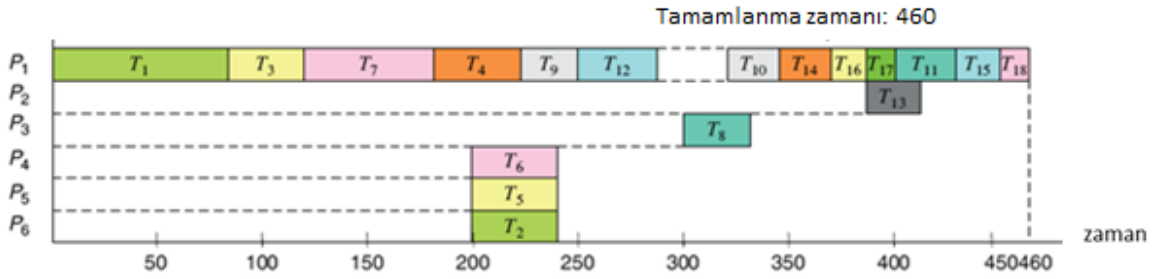
Çizelgelenmiş düğüm (scheduled node): Bir işlemci üzerinde bulunan düğüme çizelgelenmiş düğüm denir.

Serbest düğümler (free node): Kendinden önceki düğümler çizelgelenmiş olduğu halde kendisi çizelgelenmiş olmayan düğüme serbest düğüm denir.

Nispeten serbest düğümler (partial free node): Kendinden önceki düğümlerden en az birinin ve kendisinin çizelgeli yok ise buna nispeten serbest düğüm denir (Mccreary *et al.* 1994).

Bu algoritmada, serbest ve nispeten serbest düğümlere öncelik tanıyarak düzeltme yapacağız.

Şekil 4.9'da 6 tane işlemci kullanarak Şekil 4.4'te gösterilen görev çizelgesi üzerinde DSC algoritması ile hazırlanan Gantt grafiği gösterilmektedir (Wille *et al.* 2011).



Şekil 4.9. 6 tane işlemci kullanarak Şekil 4.4'te gösterilen görev çizgesi üzerinde DSC algoritması ile hazırlanan Gantt grafiği (Wille *et al.* 2011)

4.10. DSH (Duplication Scheduling Heuristic) Algoritma

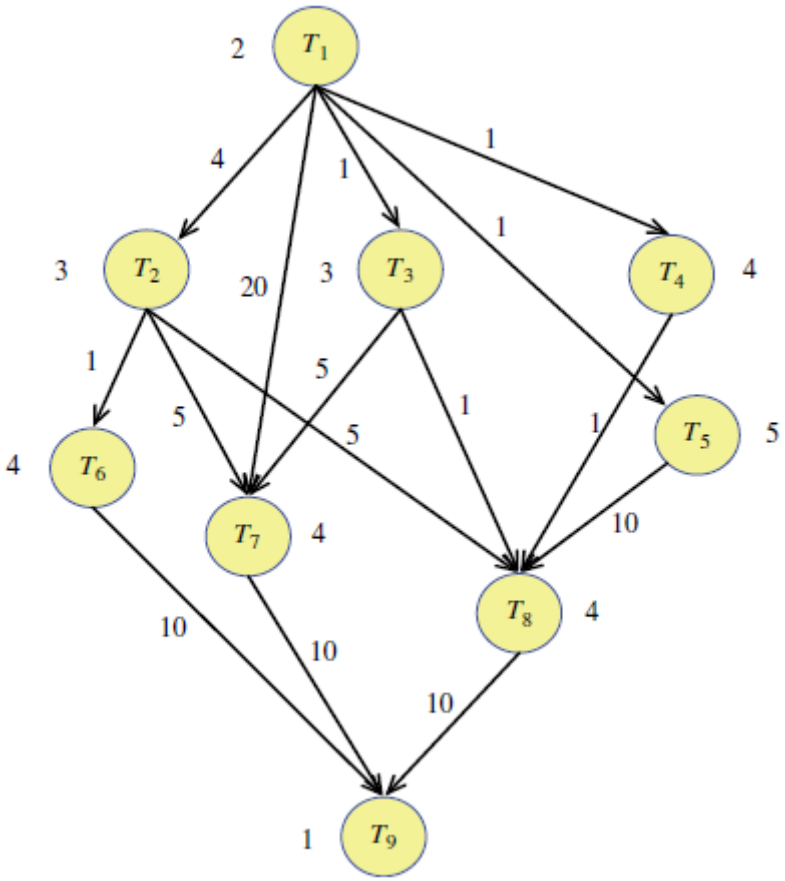
DSH algoritması zamanlama uzunluğunu en aza indirmek için tekrarlanan girişimler fikrine dayanmaktadır. DSH algoritması aşağıda şekilde gerçekleşir:

1. Daha düşük seviyedeki düğümlerin her biri için (bl) hesaplanır. Düğümleri L listesini bl (düşük seviye miktarı)'ye bağlı olarak azalan şekilde düzenliyoruz.
2. L listesinden n düğümünü seçeriz.
3. Tüm işlemciler için aşağıdaki adımları uygulayacağız:
a) Şimdi P işlemcisi üzerinde n düğümünün başlangıç zamanı hesaplanır. n düğümünün zamanlaması için uygun işlemci bulunur.
b) n düğümünden önceki düğüm kümesini ele alalım. n' düğümü n düğümün öncesidir ve n' düğümünden n düğümüne olan iletim süresi n düğümden önceki düğümlerinkinden büyüktür. bu nedenle n' düğümü p işlemci üzerindeki ilk serbest zaman diliminde tekrarlanacaktır.
c) Eğer serbest zaman dilimi olmazsa, başlangıç zamanı (st) p işlemci için kaydedilip, üçüncü adımın ilk aşamasına gidilir. Aksi halde, şimdiki başlangıç zamanı önceki başlangıç zamanından az olursa, şimdiki başlangıç zamanı seçilir. Geçerli düğümünün n düğümü olduğu farz edilir. Ve üçüncü adımın ilk aşamasına gidilir.
4. n düğümü başlangıç zamanını en az verecek işlemci üzerinde zamanlanır ve bu işlemci n düğümünden önceki tüm düğümler için tekrarlanır.

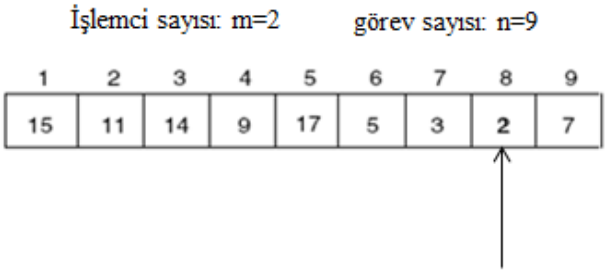
5. 2 ile 4 arasındaki adımlar tüm düğümler zamanlanana kadar tekrarlanır (Kwok and Ahmad 1999).

4.11. Genetik Algoritma

Genetik algoritmalar kullanarak çoklu işlemci zamanlama problemleri için çeşitli yöntemler sunulmuştur. Bir kromozom kodlama sorununun çözümün, genetik algoritmalar için önemli özelliklerden biridir. Geniş yöntemi (PMC) dizileri (kromozom) göstermek için kullanıyoruz. Bu dizilerin her biri düğümlerin önceliğini ve hangi düğümlerin işlemcilere tahsis edileceklerini gösterecekler. Örneğin Şekil 4.11'de Şekil 4.10'daki görev çizgesine ait basit bir PMC şekli gösterilmiştir (Hwang *et al.* 2008).



Şekil 4.10. 9 düğümlü bir DAG örneği (Hwang *et al.* 2008)

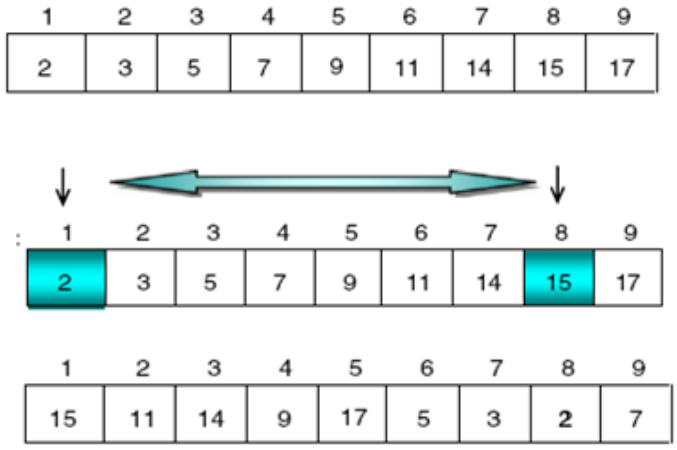


Şekil 4.11. PMC'den bir örnek (Hwang *et al.* 2008)

Şekil 4.11'de 2 sayısı iki şeyi anlatıyor:

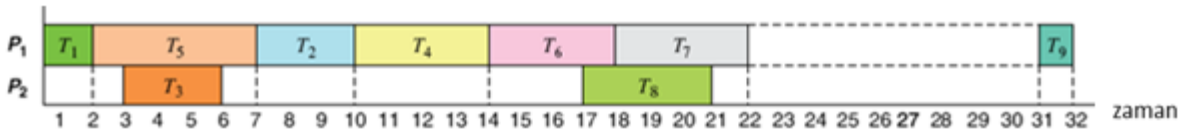
1) Düğümün önceligini gösteriyor.
2) Düğümün tahsis edileceği işlemciyi gösteriyor.

Genel olarak düğümlerin öncelikleri tek olursa birinci işlemciye ve çift olursa ikinci işlemciye tahsis edilir. Düğümler arasındaki bağların izleyeceği kodlama prosedürünün hazırlanması önemli bir adımdır. Önerilen kodlama usulu, öncelikli düğümlere dayanır. Her kromozomun büyüklüğü görevlerin sayısına eşittir. Her kromozomda her gen random sayılarla doldurulacak ve bu sayılar hem önceliğini hem de düğümlerin hangi işlemciye tahsis edileceklerini gösterecektir. Sonra iki gen n/2 kez seçilecek ve bu iki genin random sayıları birbiriyle değiştirilecektir (Hwang *et al.* 2008). En son kromozom Şekil 4.12'deki gibi olacaktır.



Şekil 4.12. PMC algoritmasıyla türetilmiş kromozomlar (Hwang *et al.* 2008)

Son olarak, düğümler işlemcilerle eşleştirildikten sonra, Şekil 4.13'de gösterilen zamanlama elde edilecek.



Şekil 4.13. Şekil 4.10'daki DAG'dan elde edilmiş Gantt grafiği (Hwang *et al.* 2008)

5. SONUÇLAR ve ÖNERİLER

Bu bölümde önerilen algoritmanın seri ve paralel performansları incelenip, test edilmiştir. SSH (Secure Shell Client) programı ile paralel hesap kümesine bağlantı yapılmıştır ve söz konusu kodlar küme üzerindeki hesap düğümleri üzerinde çalıştırılmıştır. Hem seri hem de paralel program ilk adımda bir metin dosyası içerisinde saklanan görev çizgesini alır ve algoritmayı uyguladıktan sonra aşağıdaki çıktıları üretir (Bu çıktı ve girdiler için **EK 1** ve **EK 2**'deki programlar kullanılmıştır).

1. Programın bitiş zamanı (FT) (best latency)
2. Son görevin uygulanışının aldığı süre (makespan)
3. Duvar saati zamanı (wall clock time)
4. Her görevin başlangıç ve bitiş zamanı
5. Her görevin öncül ve ardıllar (predecessors ve successors)
6. Komşuluk matrisi (adjacency matrix)
7. Her görevin yürütme zamanı (execution times)
8. Görevlerin üzerinde uygulanacakları işlemciler (task bounds to processors)
9. Görev grafiklerinin görev sayısı
10. Görevlerin bağlı olduğu diğer görevler
11. Görev grafiklerinin ok sayısı (edges of the task graph)

Seri versiyon

Seri versiyonunda bulunan aşağıdaki komutları SSH ortamında çalıştırmalıyız:

1) g++ GABasedSerialScheduling.h GABasedSerialScheduling.cpp ExecutiveSerial.cpp -std=c++0x -o out.x.

2) ./out.x.

Paralel versiyon

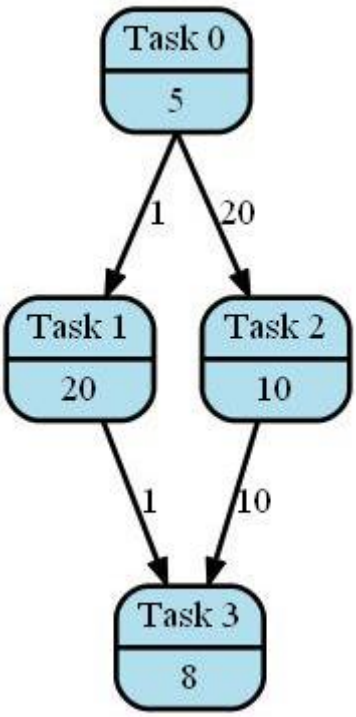
Paralel versiyonunda bulunan aşağıdaki komutları SSH ortamında çalıştırmalıyız:

1) mpicxx GABasedParallelScheduling.h GABasedParallelScheduling.cpp ExecutiveParallel.cpp -std=c++0x -o out.x

2) mpirun -np 2 ./out.x

3) mpiexec –f ~cluster_hosts -n 5 ./out.x

Programı paralel ve seri versiyonları ile test ettiğimizde aynı zaman sonucunu elde etmekteyiz. Görev çizgelerinde görevlerin numarası 1'den başlarken, programda 0'dan başlar. Aşağıdaki görev çizgesi boyut bakımından küçük olduğu için tüm çıkışları yazıyoruz. Ancak, bunun haricindeki görev çizgelerinin çıkışları çok uzun olduğu için sadece önemli çıkışları yazıyoruz.



Şekil 5.1. 4 görevden oluşan görev çizgesi (DAG) (Hwang *et al.* 2008)

Name of the Task Graph:? TG4Tasks.txt

Number of Processors for the Schedule:? 2

THE NUMBER OF TASKS ON THE TASK GRAPH IS -> 4

THE NUMBER OF EDGES ON THE TASK GRAPH IS -> 4

EXECUTION TIMES OF THE TASKS:

Task0 -> 5

Task1 -> 20

Task2 -> 10

Task3 -> 8

EDGES OF THE TASK GRAPH:

There is an edge between 0 and 1

There is an edge between 0 and 2

There is an edge between 1 and 3

There is an edge between 2 and 3

ADJACENCY MATRIX:

0 1 20 0

0 0 0 1

0 0 0 10

0 0 0 0

PREDECESSORS OF THE TASKS:

Predecessors of Task0 ->

Predecessors of Task1 -> 0

Predecessors of Task2 -> 0

Predecessors of Task3 -> 1 2

SUCCESSORS OF THE TASKS:

Successors of Task0 -> 1 2

Successors of Task1 -> 3

Successors of Task2 -> 3

Successors of Task3 ->

THE BEST LATENCY IS -> 34

ELAPSED TIME IS -> 106.837 seconds

THE BEST CHROMOSOME IS -> 2 1 0 3 0 1 0 1

SCHEDULING TIMES OF TASKS:

TASK0 is scheduled at time interval (0,5)

TASK1 is scheduled at time interval (6,26)

TASK2 is scheduled at time interval (5,15)

TASK3 is scheduled at time interval (26,34)

TASKS-PROCESSORS BINDINGS:

TASKS BOUND TO PROCESSOR0 -> 0 2

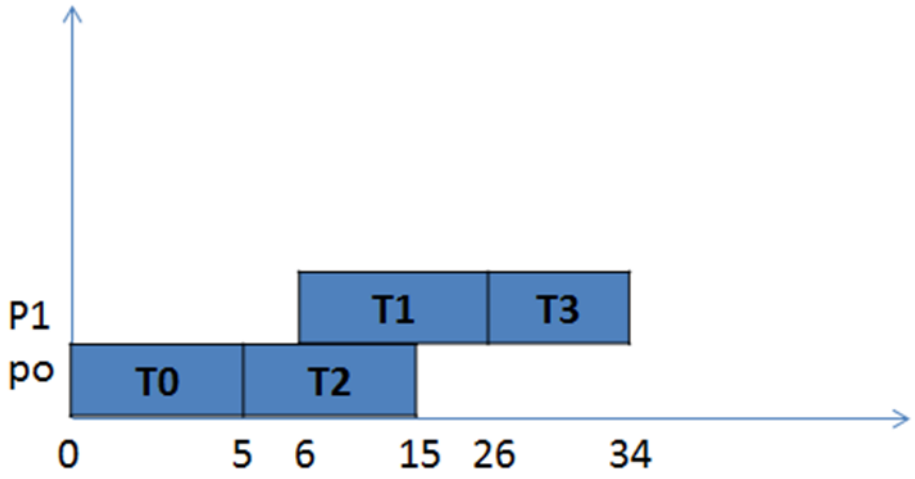
TASKS BOUND TO PROCESSOR1 -> 1 3

THE GANTT CHART OF THE SCHEDULE:

TIME | 0| 1| 2| 3| 4| 5| 6| 7| 8|9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24| 25|26|27|28|29|30|31|32|33|34|

PROCESSOR0| 0| 0| 0| 0| 0| 2| 2| 2| 2| 2| 2| 2|2|2|

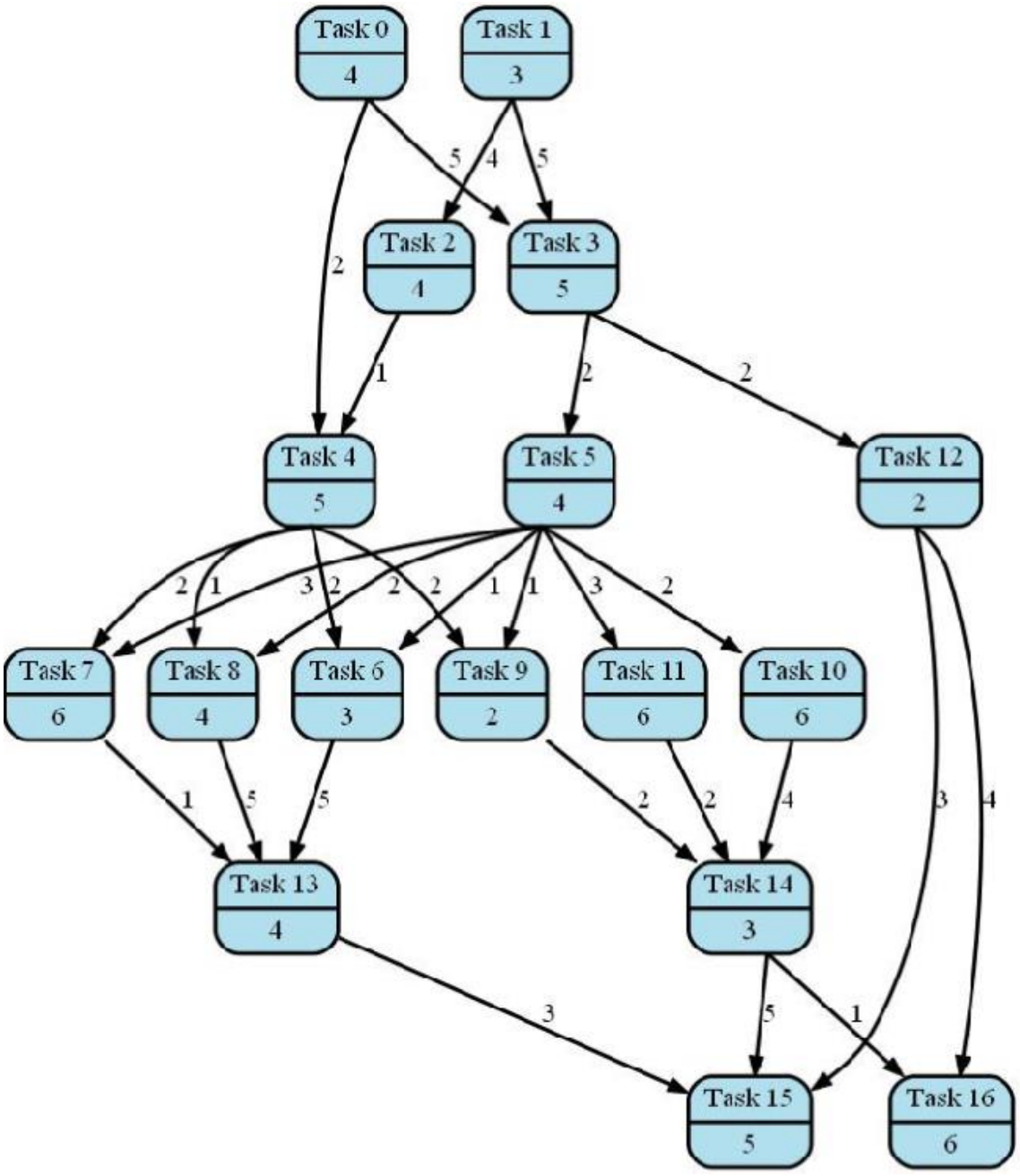
PROCESSOR1| | | | | | 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 3| 3| 3| 3| 3| 3| 3| 3|



Şekil 5.2. 4 görevden oluşan görev çizgesine (Şekil 5.1) ait Gantt grafiği

Bu bölümde önerilen algoritma (MHA) diğer ünlü algoritmalarla, mesela HLFET, EFT, MCP, DLS, GA, görev grafikler (DAG) kullanılarak kıyaslandırılır ve kontrol edilir. İlk

olarak, Şekil 5.3 görev çizgesi ele alınarak, önerilen algoritma ile elde edilen çıkışlar Şekil 5.4'te gösterilmiştir. Çizelge 5.1'in algoritması tarafından elde edilen FT sonuçları diğer algoritmalar ile kıyaslandırılır. Önerilen algoritma için parametreler Çizelge 5.2'de gösterilmiştir (Algoritmamızın tüm aşamaları **EK 1** ve **EK 2**'de program olarak mevcuttur).



Şekil 5.3. 17 görevden oluşan görev çizgesi (DAG) (Nezhad *et al.*2011)

Name of the Task Graph:? TG17Tasks.txt

Number of Processors for the Schedule:? 3

THE BEST LATENCY IS -> 39

ELAPSED TIME IS -> 435.013 seconds

THE BEST CHROMOSOME IS -> 9 4 12 8 1 14 2 6 10 3 15 7 5 11 13 16 0 1 2 2 1 1 2 2 2 0 0 2 0 0 2 2 0 0

SCHEDULING TIMES OF TASKS:

TASK0 is scheduled at time interval (0,4)

TASK1 is scheduled at time interval (0,3)

TASK2 is scheduled at time interval (7,11)

TASK3 is scheduled at time interval (8,13)

TASK4 is scheduled at time interval (11,16)

TASK5 is scheduled at time interval (13,17)

TASK6 is scheduled at time interval (18,21)

TASK7 is scheduled at time interval (23,29)

TASK8 is scheduled at time interval (20,24)

TASK9 is scheduled at time interval (18,20)

TASK10 is scheduled at time interval (17,23)

TASK11 is scheduled at time interval (21,27)

TASK12 is scheduled at time interval (16,18)

TASK13 is scheduled at time interval (30,34)

TASK14 is scheduled at time interval (27,30)

TASK15 is scheduled at time interval (34,39)

TASK16 is scheduled at time interval (31,37)

TASKS-PROCESSORS BINDINGS:

TASKS BOUND TO PROCESSOR0 -> 0 3 5 10 7 16

TASKS BOUND TO PROCESSOR1 -> 1 9 8

TASKS BOUN TO PROCESSOR2 -> 2 4 12 6 11 14 13 15

THE GANTT CHART OF THE SCHEDULE:

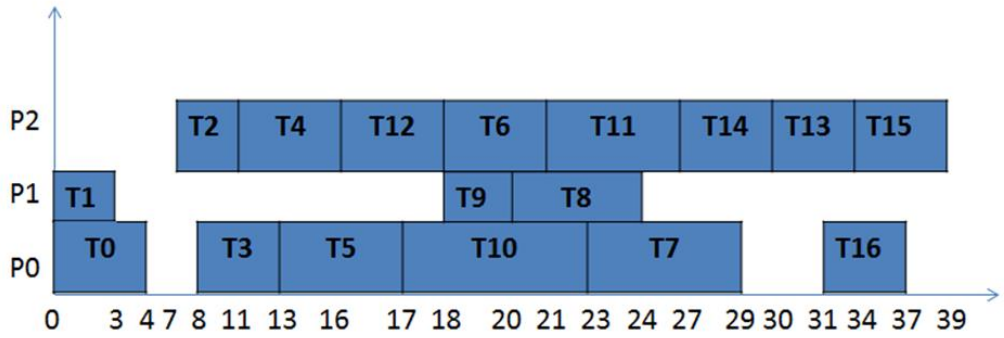
TIME | 0| 1| 2| 3| 4| 5| 6| 7| 8|
9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|36|37|38
|39|

PROCESSOR0| 0| 0| 0| 0| | | | |3| 3| 3| 3| 3| 5| 5| 5| 5|10|10|10|10|10|10| 7| 7| 7| 7| 7| 7| |
|16|16|16|16|16|16|

PROCESSOR1| 1| 1| 1| | | | | | | | | | | | | | | |9| 9| 8| 8| 8| 8|

PROCESSOR2| | | | | | | |2| 2| 2| 2| 4| 4| 4| 4| 4|12|12| 6| 6|
6|11|11|11|11|11|11|14|14|14|13|13|13|13|15|15|15|15|15|

Şekil 5.4. 17 görevden oluşan görev çizgesine (Şekil 5.3) ait programın çıktısı



Şekil 5.5. 17 görevden oluşan görev çizgesine (Şekil 5.3) ait Gantt grafiği

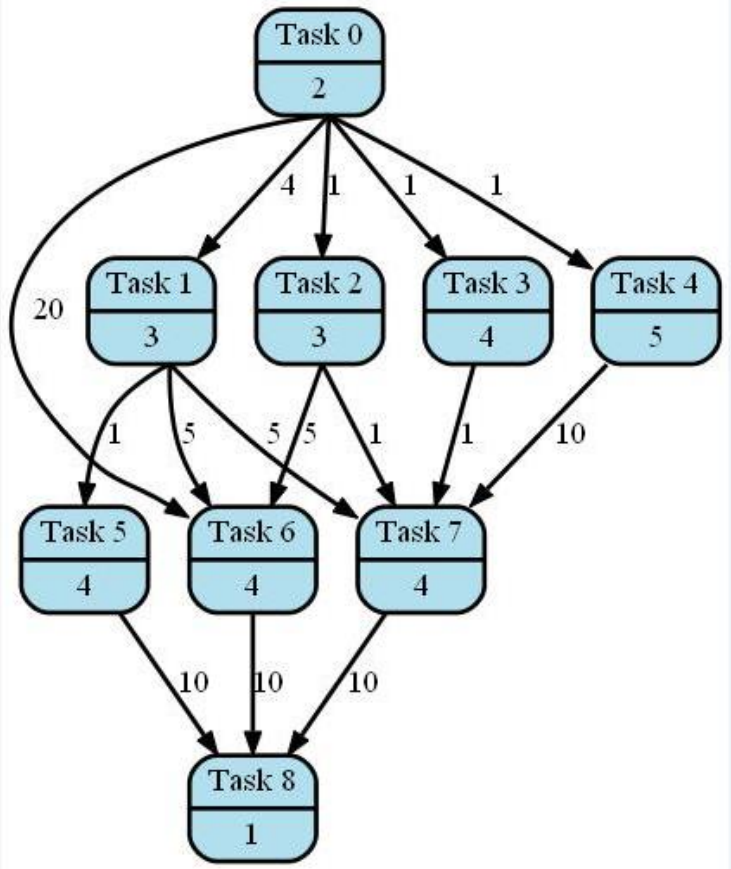
Çizelge 5.1. Şekil 5.3 görev çizgesi için kıyaslamalar

Şekil 5.3. görev çizgesi için								
Algoritmanın adı	**MHA**	**GA**	**LC**	**MCP**	**EZ**	**HLEFT**	**EFT**	**LAST**
Algoritmanın numarası	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
İşlemcinin sayısı	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
Bitiş zamanı	**39**	**39**	**39**	**40**	**40**	**41**	**41**	**43**

Çizelge 5.2. Önerilen algoritmanın özellikleri

Önerilen algoritmanın özellikleri	**İlk Popülasyon**	**İterasyon Sayısı**	**Çaprazlama Olasılığı**	**Mutasyon Olasılığı**
	50	**100**	**0.7**	**0.3**

Şimdi Şekil 5.6'ya göre MCP, GA, MHA sonuçları Çizelge 5.3'te görülebilir.



Şekil 5.6. 9 görevden oluşan görev çizgesi (DAG) (Hwang *et al.* 2008)

Çizelge 5.3. Şekil 5.6 için kıyaslamalar

Şekil 5.6 için			
Algoritmanın adı	MHA	GA	Mcp
Algoritmanın numarası	1	2	3
İşlemci sayısı	2	2	2
Bitiş zamanı	21	23	25

Çizelge 5.4. Şekil 5.6 için önerilen algoritmanın özellikleri

Önerilen algoritmanın özellikleri	**İlk Popülasyon**	**İterasyon Sayısı**	**Çaprazlama Olasılığı**	**Mutasyon Olasılığı**
	50	**100**	**0.7**	**0.3**

Bu DAG'ın program çıkışı aşagıda gibi olacak:

Name of the Task Graph:? TG9Tasks-1.txt

Number of Processors for the Schedule:? 2

THE BEST LATENCY IS -> 21

ELAPSED TIME IS -> 290.239 seconds

THE BEST CHROMOSOME IS -> 5 0 2 1 3 4 6 7 8 0 1 0 1 0 1 1 1 1

SCHEDULING TIMES OF TASKS:

TASK0 is scheduled at time interval (0,2)

TASK1 is scheduled at time interval (2,5)

TASK2 is scheduled at time interval (3,6)

TASK3 is scheduled at time interval (10,14)

TASK4 is scheduled at time interval (5,10)

TASK5 is scheduled at time interval (6,10)

TASK6 is scheduled at time interval (11,15)

TASK7 is scheduled at time interval (15,19)

TASK8 is scheduled at time interval (20,21)

TASKS-PROCESSORS BINDINGS:

TASKS BOUND TO PROCESSOR0 -> 2 5 3

TASKS BOUND TO PROCESSOR1 -> 0 1 4 6 7 8

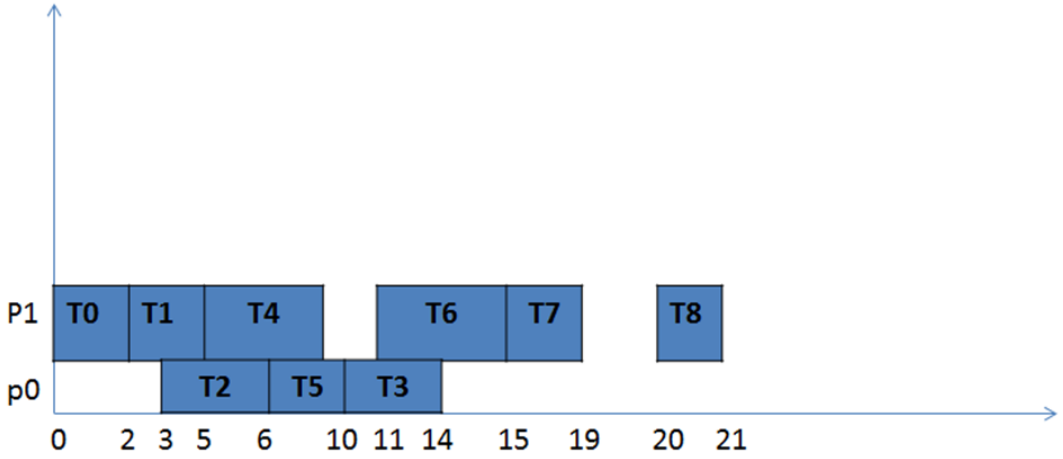
THE GANTT CHART OF THE SCHEDULE:

TIME | 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|16|17|18|19|20|21|

PROCESSOR0| | | | 2| 2| 2| 5| 5| 5| 5| 3| 3| 3| 3|

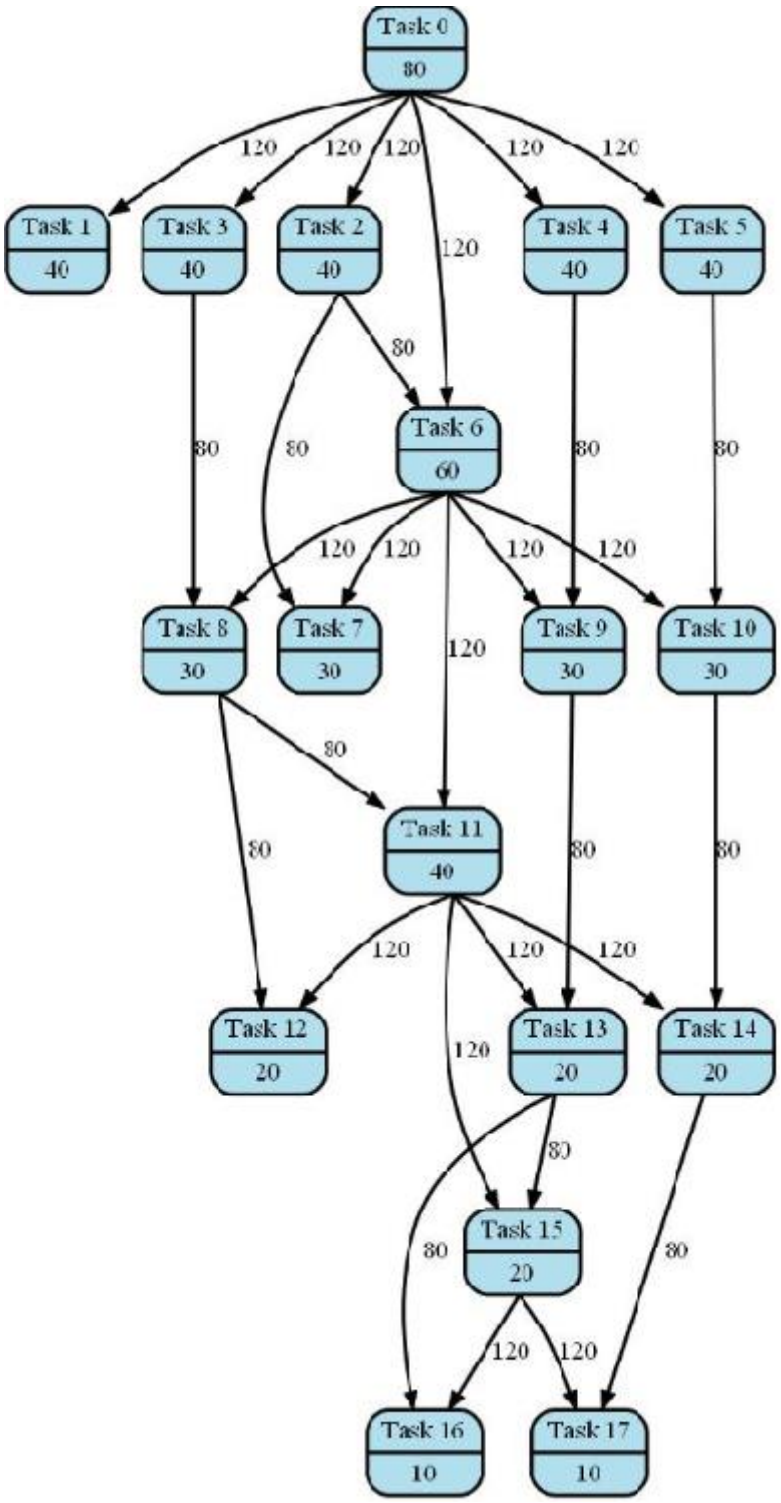
PROCESSOR1| 0| 0| 1| 1| 1| 4| 4| 4| 4| 4| | 6| 6| 6| 6| 7| 7| 7| 7| | 8|

Şekil 5.7. 9 görevden oluşan görev çizgesine (Şekil 5.6) ait programın çıktısı



Şekil 5.8. 9 görevden oluşan görev çizgesine (Şekil 5.6) ait Gantt grafiği

Bu bölümde Şekil 5.9'daki görev çizgesi için DSC, MCP, DCP, GA ve MD algoritmaları ile önerilen algoritma kıyaslanır ve analiz edilir.



Şekil 5.9. 18 görevden oluşan görev çizgesi (DAG) (Wang *et al.* 2011)

Çizelge 5.5'te DSC, MD, MCP, DCP, GA ve MHA algoritmaların sonuçları gösterilmiştir.

Çizelge 5.5. DAG Şekil 5.9 için kıyaslamalar

DAG Şekil 5.9						
Algoritmanın adı	MHA	GA	DCP	MD	DSC	MCP
Algoritmanın numarası	1	2	3	4	5	6
İşlemci sayısı	2	2	3	3	6	3
Bitiş zamanı	440	440	440	460	460	520

Çizelge 5.6. DAG Şekil 5.9 için önerilen algoritmanın özellikleri

Önerilen algoritmanın özellikleri	**İlk Popülasyon**	**İterasyon Sayısı**	**Çaprazlama Olasılığı**	**Mutasyon Olasılığı**
	50	**100**	**0.7**	**0.3**

Bu DAG'ın programın çıktışı aşagıda gibi olacak:

Name of the Task Graph:? TG18Tasks.txt

Number of Processors for the Schedule:? 2

THE BEST LATENCY IS -> 440

ELAPSED TIME IS -> 243.305 seconds

THE BEST CHROMOSOME IS -> 5 10 2 11 1 17 12 8 13 7 6 3 0 9 16 15 4 14 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1

SCHEDULING TIMES OF TASKS:

TASK0 is scheduled at time interval (0,80)

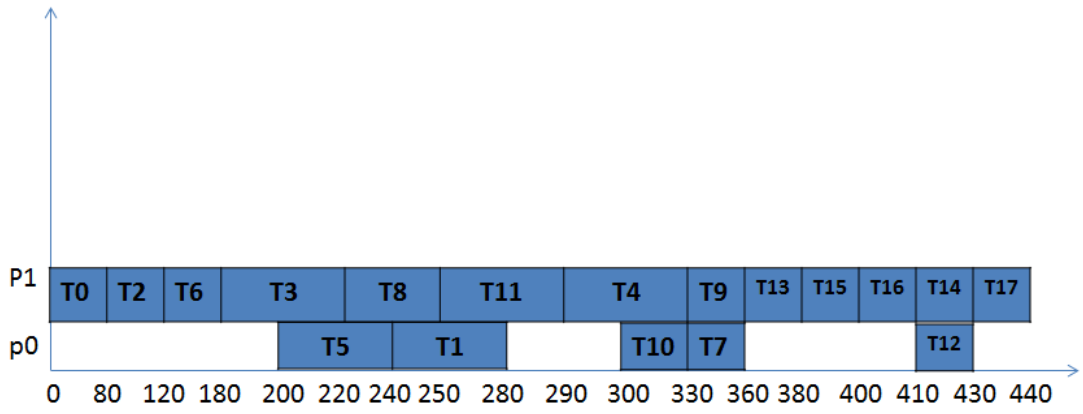
TASK1 is scheduled at time interval (240,280)

TASK2 is scheduled at time interval (80,120)

TASK3 is scheduled at time interval (180,220)

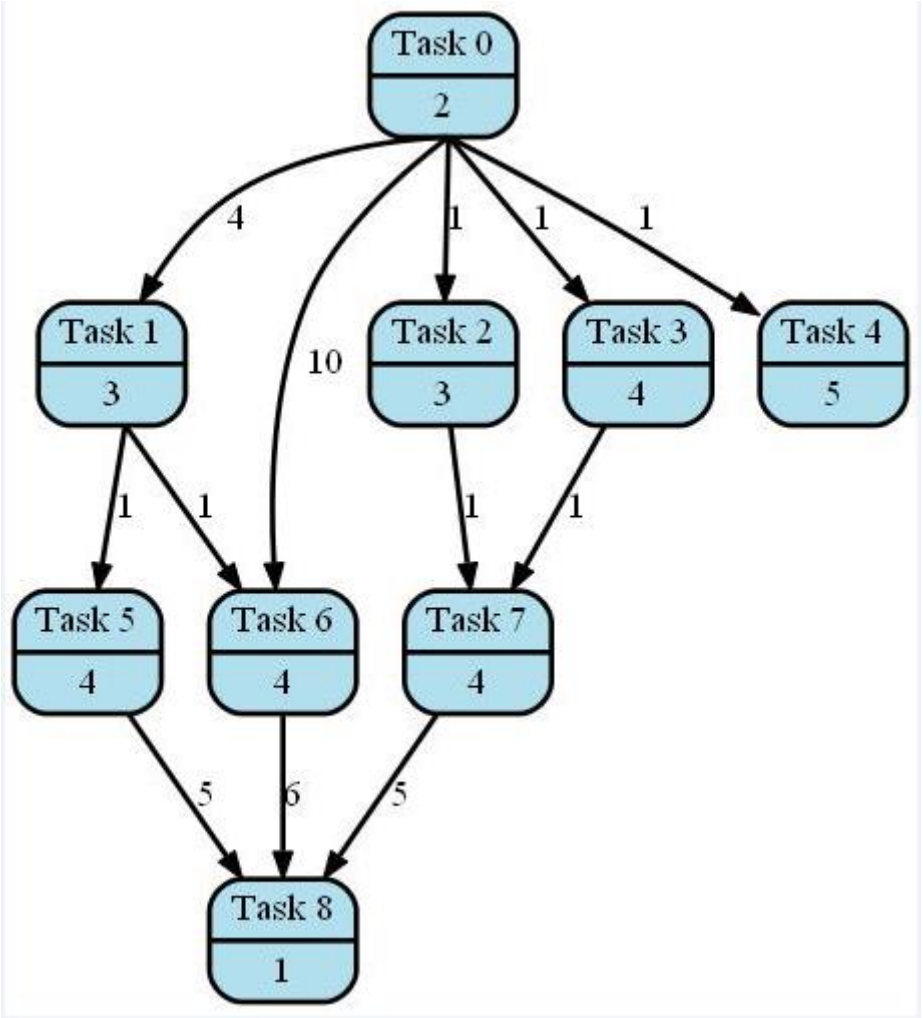
```
TASK4 is scheduled at time interval (290,330)
TASK5 is scheduled at time interval (200,240)
TASK6 is scheduled at time interval (120,180)
TASK7 is scheduled at time interval (330,360)
TASK8 is scheduled at time interval (220,250)
TASK9 is scheduled at time interval (330,360)
TASK10 is scheduled at time interval (300,330)
TASK11 is scheduled at time interval (250,290)
TASK12 is scheduled at time interval (410,430)
TASK13 is scheduled at time interval (360,380)
TASK14 is scheduled at time interval (410,430)
TASK15 is scheduled at time interval (380,400)
TASK16 is scheduled at time interval (400,410)
TASK17 is scheduled at time interval (430,440)
TASKS-PROCESSORS BINDINGS:
TASKS BOUND TO PROCESSOR0 -> 5 1 10 7 12
TASKS BOUND TO PROCESSOR1 -> 0 2 6 3 8 11 4 9 13 15 16 14 17
```

Şekil 5.10. 18 görevden oluşan görev çizgesine (Şekil 5.9) ait programın çıktısı



Şekil 5.11. 18 görevden oluşan görev çizgesine (Şekil 5.9) ait Gantt grafiği

Şimdi diğer DAG'ları önerilen algoritmaya uyguluyoruz ve sonuçları yazıyoruz:



Şekil 5.12. 9 görevden oluşan görev çizgesi (DAG) (Yin *et al.* 2006)

Programın çıktısı:

Name of the Task Graph:? TG9Tasks-2.txt

Number of Processors for the Schedule:? 2

THE BEST LATENCY IS -> 17

ELAPSED TIME IS -> 288.455 seconds

THE BEST CHROMOSOME IS -> 5 6 8 3 2 1 7 4 0 0 0 0 1 0 0 1 1 0

SCHEDULING TIMES OF TASKS:

TASK0 is scheduled at time interval (0,2)

TASK1 is scheduled at time interval (5,8)

TASK2 is scheduled at time interval (2,5)

TASK3 is scheduled at time interval (3,7)

TASK4 is scheduled at time interval (11,16)

TASK5 is scheduled at time interval (8,12)

TASK6 is scheduled at time interval (12,16)

TASK7 is scheduled at time interval (7,11)

TASK8 is scheduled at time interval (16,17)

TASKS-PROCESSORS BINDINGS:

TASKS BOUND TO PROCESSOR0 -> 0 2 1 5 6 8

TASKS BOUND TO PROCESSOR1 -> 3 7 4

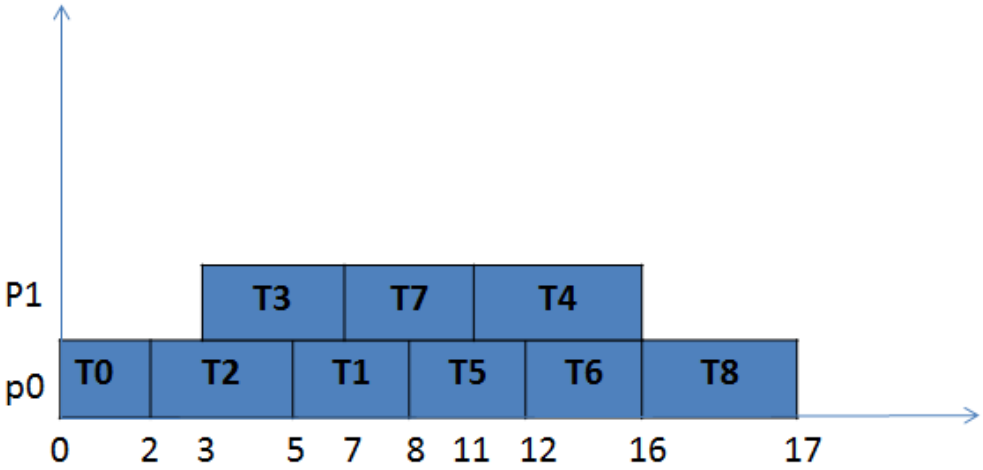
THE GANTT CHART OF THE SCHEDULE:

TIME | 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|16|17|

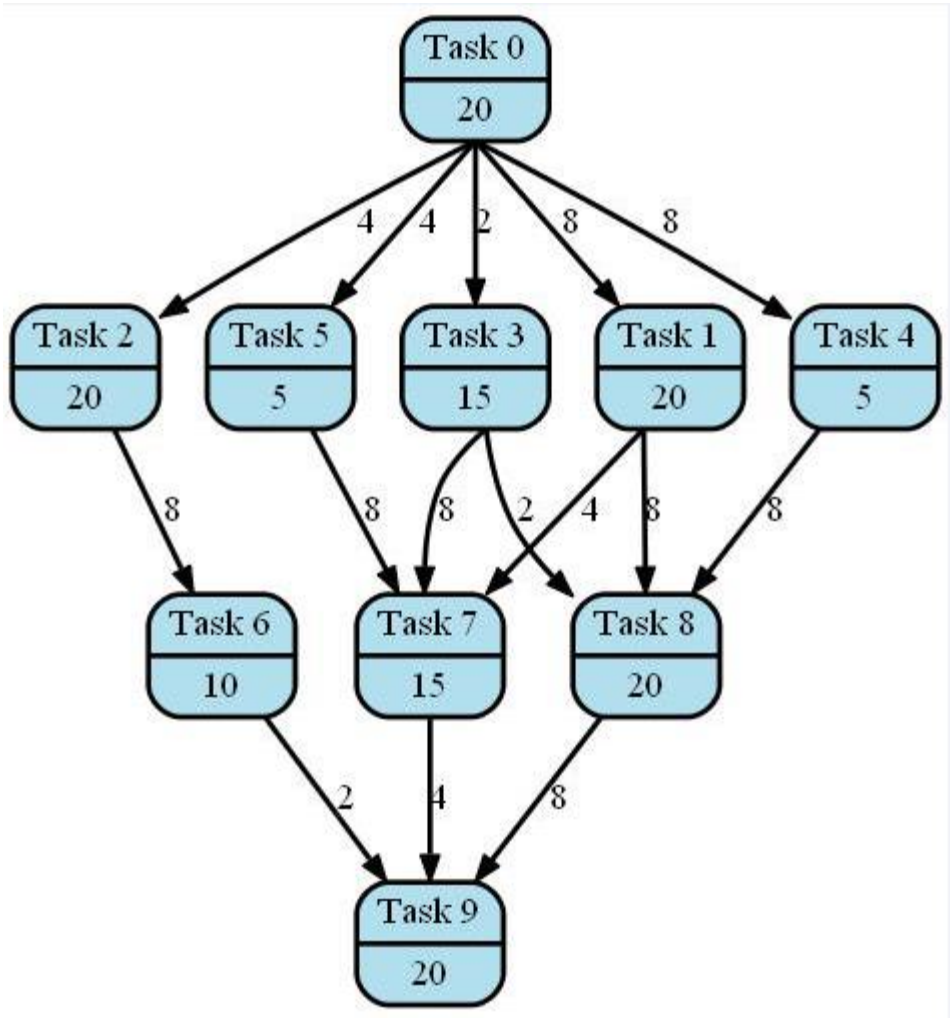
PROCESSOR0| 0| 0| 2| 2| 2| 1| 1| 1| 5| 5| 5| 5| 6| 6| 6| 6| 8|

PROCESSOR1| | | | 3| 3| 3| 3| 7| 7| 7| 7| 4| 4| 4| 4| 4|

Şekil 5.13. 9 görevden oluşan görev çizgesine (Şekil 5.12) ait programın çıktısı



Şekil 5.14. 9 görevden oluşan görev çizgesine (Şekil 5.12) ait Gantt grafiği



Şekil 5.15. 10 görevden oluşan görev çizgesi (DAG) (Hagras and Janeček 2003)

Program çıktısı:

Name of the Task Graph:? TG10Tasks.txt

Number of Processors for the Schedule:? 3

THE BEST LATENCY IS -> 85

ELAPSED TIME IS -> 306.069 seconds

THE BEST CHROMOSOME IS -> 0 6 4 3 5 8 2 9 1 7 2 0 0 1 1 2 0 2 2 1

SCHEDULING TIMES OF TASKS:

TASK0 is scheduled at time interval (0,20)

TASK1 is scheduled at time interval (20,40)

TASK2 is scheduled at time interval (33,53)

TASK3 is scheduled at time interval (22,37)

TASK4 is scheduled at time interval (28,33)

TASK5 is scheduled at time interval (37,42)

TASK6 is scheduled at time interval (53,63)

TASK7 is scheduled at time interval (44,59)

TASK8 is scheduled at time interval (41,61)

TASK9 is scheduled at time interval (65,85)

TASKS-PROCESSORS BINDINGS:

TASKS BOUND TO PROCESSOR0 -> 4 2 6

TASKS BOUND TO PROCESSOR1 -> 3 5 7

TASKS BOUND TO PROCESSOR2 -> 0 1 8 9

THE GANTT CHART OF THE SCHEDULE:

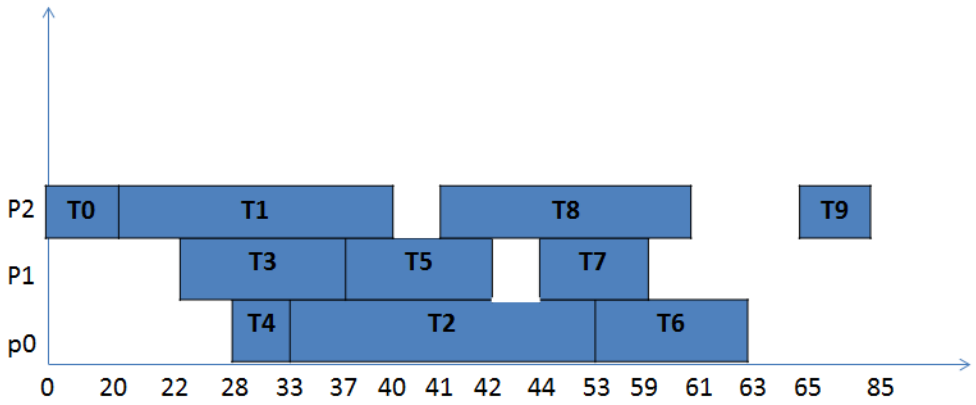
TIME | 0| 1| 2| 3| 4| 5| 6| 7| 8|
9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|36|37|38
|39|40|41|42|43|44|45|46|47|48|49|50|51|52|53|54|55|56|57|58|59|60|61|62|63|64|65|66|67|
68|69|70|71|72|73|74|75|76|77|78|79|80|81|82|83|84|85|

PROCESSOR0| | | | | | | | | | | | | | | | | | | | | | | | | | | | | 4| 4| 4| 4| 4| 2| 2| 2| 2|
2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 2| 6| 6| 6| 6| 6| 6| 6| 6| 6| 6|

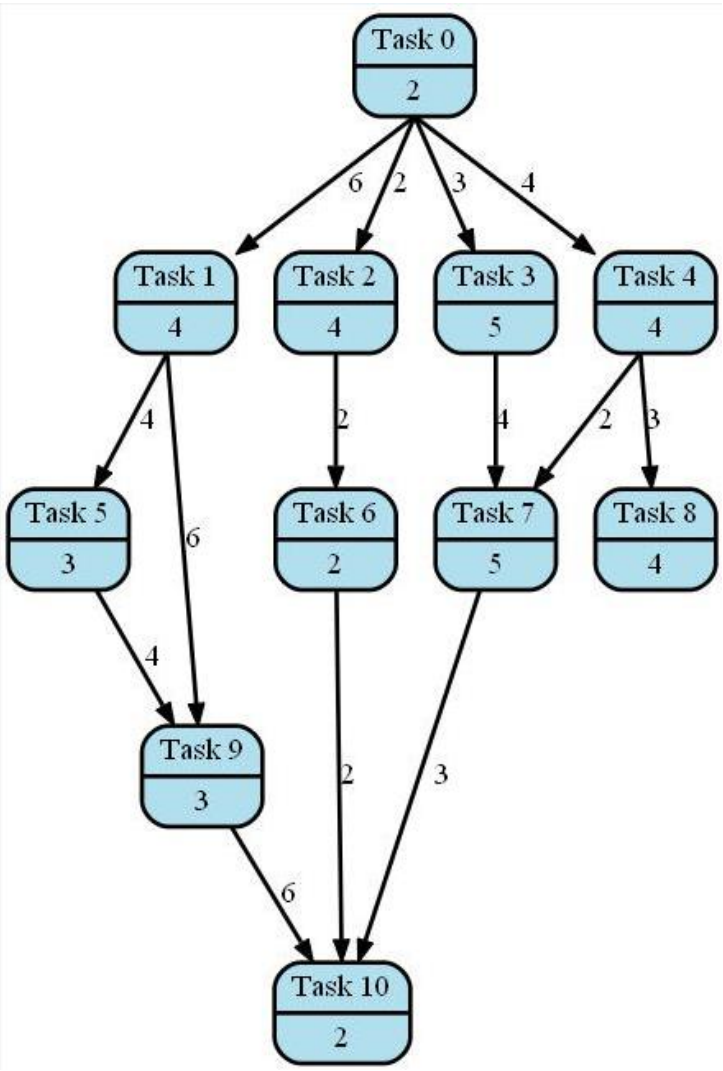
PROCESSOR1| | | | | | | | | | | | | | | | | | | | | | | 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3|
3| 5| 5| 5| 5| 5| | | 7| 7| 7| 7| 7| 7| 7| 7| 7| 7| 7| 7| 7| 7| 7|

PROCESSOR2| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
1| 1| 1| 1| 1| 1| 1| 1| 1| 1| | 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| | | | | 9| 9| 9|
9| 9| 9| 9| 9| 9| 9| 9| 9| 9| 9| 9| 9| 9| 9| 9| 9|

Şekil 5.16. 10 görevden oluşan görev çizgesine (Şekil 5.15) ait programın çıktısı



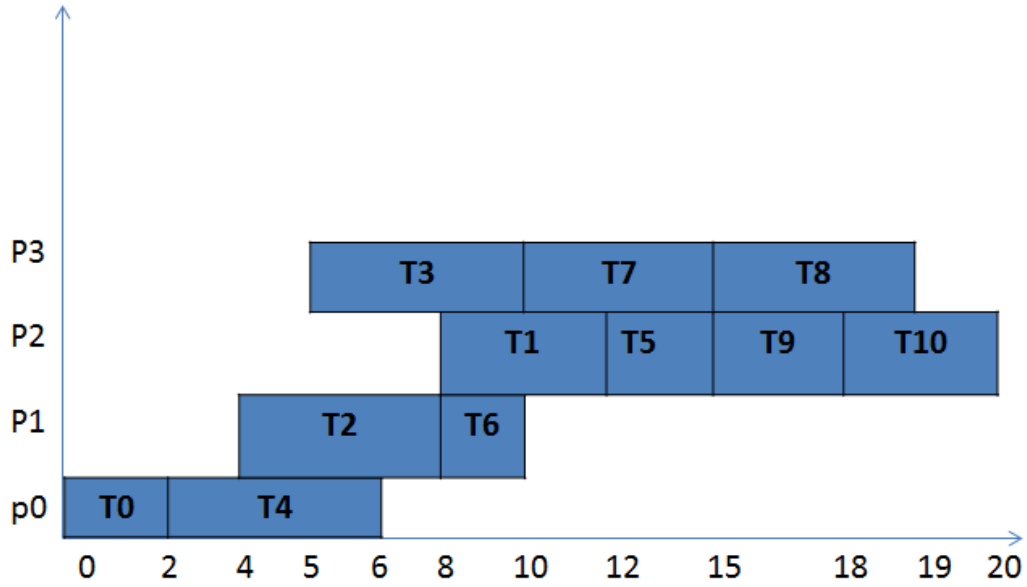
Şekil 5.17. 10 görevden oluşan görev çizgesine (Şekil 5.15) ait Gantt grafiği



Şekil 5.18. 11 görevden oluşan görev çizgesi (DAG) (Rajak 2012)

```
Program çıktısı:
Name of the Task Graph:? TG11Tasks-1.txt
Number of Processors for the Schedule:? 4
THE BEST LATENCY IS -> 20
ELAPSED TIME IS -> 325.897 seconds
THE BEST CHROMOSOME IS -> 0 5 3 1 2 6 4 9 7 10 8 0 2 3 2 1 1 0 2 3 2 3
SCHEDULING TIMES OF TASKS:
TASK0 is scheduled at time interval (0,2)
TASK1 is scheduled at time interval (8,12)
TASK2 is scheduled at time interval (4,8)
TASK3 is scheduled at time interval (5,10)
TASK4 is scheduled at time interval (2,6)
TASK5 is scheduled at time interval (12,15)
TASK6 is scheduled at time interval (8,10)
TASK7 is scheduled at time interval (10,15)
TASK8 is scheduled at time interval (15,19)
TASK9 is scheduled at time interval (15,18)
TASK10 is scheduled at time interval (18,20)
TASKS-PROCESSORS BINDINGS:
TASKS BOUND TO PROCESSOR0 -> 0 4
TASKS BOUND TO PROCESSOR1 -> 2 6
TASKS BOUND TO PROCESSOR2 -> 1 5 9 10
TASKS BOUND TO PROCESSOR3 -> 3 7 8
THE GANTT CHART OF THE SCHEDULE:
TIME      | 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|16|17|18|19|20|
PROCESSOR0| 0| 0| 4| 4| 4| 4|
PROCESSOR1|  |  |  | 2| 2| 2| 2| 6| 6|
PROCESSOR2|  |  |  |  |  |  |  | 1| 1| 1| 1| 5| 5| 5| 9| 9| 9|10|10|
PROCESSOR3|  |  |  |  | 3| 3| 3| 3| 3| 7| 7| 7| 7| 7| 8| 8| 8| 8|
```

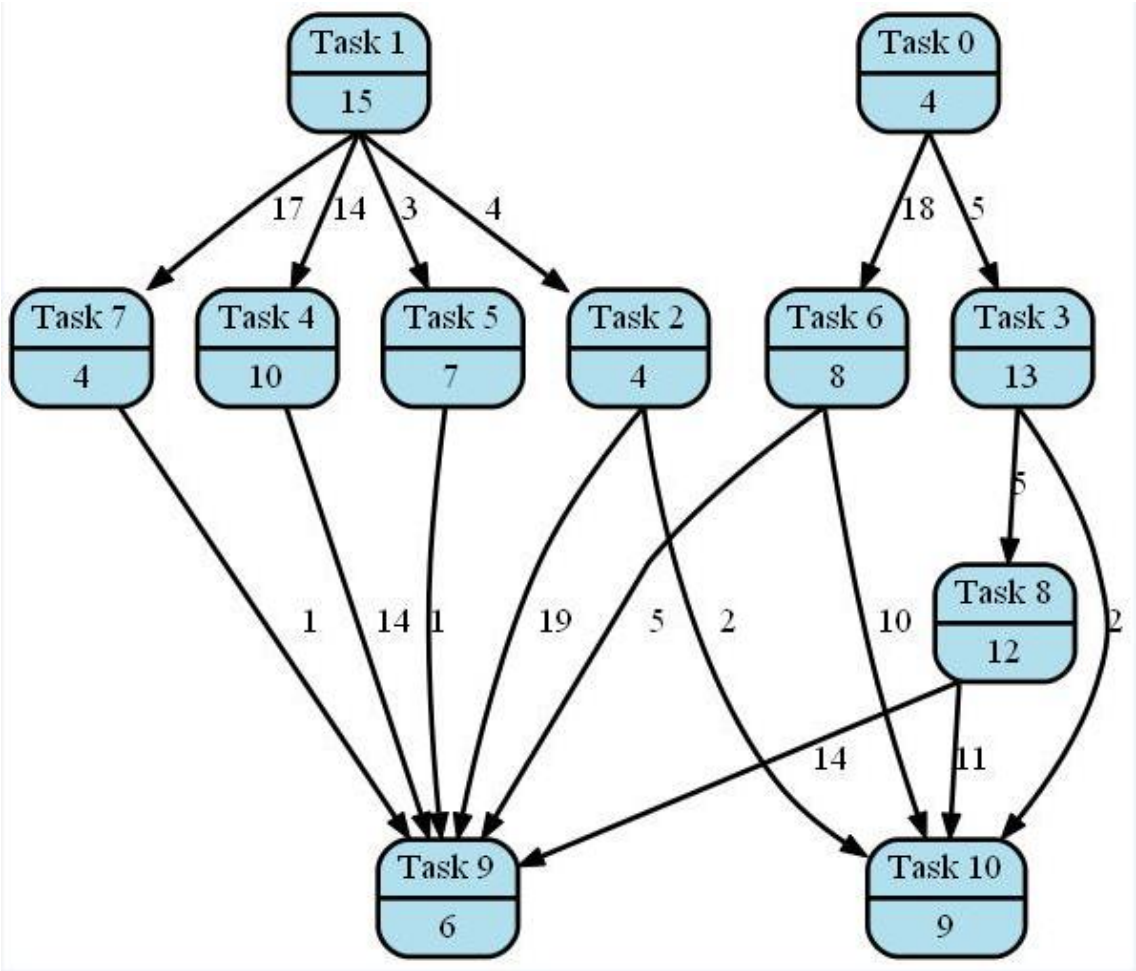
Şekil 5.19. 11 görevden oluşan görev çizgesine (Şekil 5.18) ait programın çıktısı



Şekil 5.20. 11 görevden oluşan görev çizgesine (Şekil 5.18) ait Gantt grafiği

Çizelge 5.7. DAG Şekil 5.18 için kıyaslamalar

DAG Şekil 5.18					
Algoritmanın adı	MHA	HLFET	MCP	ETF	DLS
Algoritmanın numarası	1	2	3	4	5
İşlemci sayısı	4	4	4	4	4
Bitiş zamanı	20	22	24	25	24



Şekil 5.21. 10 görevden oluşan görev çizgesinden (DAG) bir örnek (Bansal and Kaur 2012)

Program çıktısı:

Name of the Task Graph:? TG11Tasks-2.txt

Number of Processors for the Schedule:? 2

THE BEST LATENCY IS -> 49

ELAPSED TIME IS -> 322.602 seconds

THE BEST CHROMOSOME IS -> 5 2 10 1 4 7 3 8 0 6 9 1 1 0 1 1 1 0 0 0 0 1

SCHEDULING TIMES OF TASKS:

TASK0 is scheduled at time interval (0,4)

TASK1 is scheduled at time interval (0,15)

TASK2 is scheduled at time interval (22,26)

TASK3 is scheduled at time interval (4,17)

TASK4 is scheduled at time interval (26,36)

TASK5 is scheduled at time interval (15,22)

TASK6 is scheduled at time interval (29,37)

TASK7 is scheduled at time interval (36,40)

TASK8 is scheduled at time interval (17,29)

TASK9 is scheduled at time interval (43,49)

TASK10 is scheduled at time interval (37,46)

TASKS-PROCESSORS BINDINGS:

TASKS BOUND TO PROCESSOR0 -> 0 3 8 6 10

TASKS BOUND TO PROCESSOR1 -> 1 5 2 4 7 9

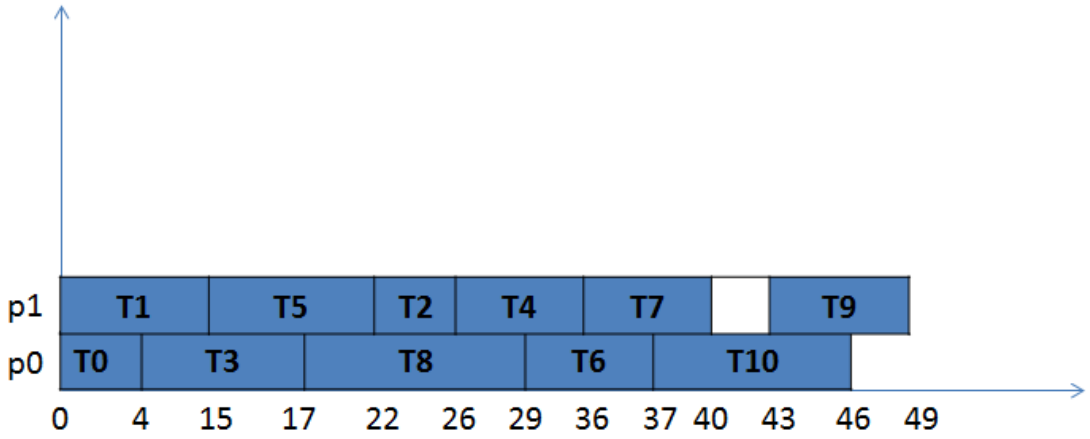
THE GANTT CHART OF THE SCHEDULE:

TIME | 0| 1| 2| 3| 4| 5| 6| 7| 8|
9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|36|37|38
|39|40|41|42|43|44|45|46|47|48|49|

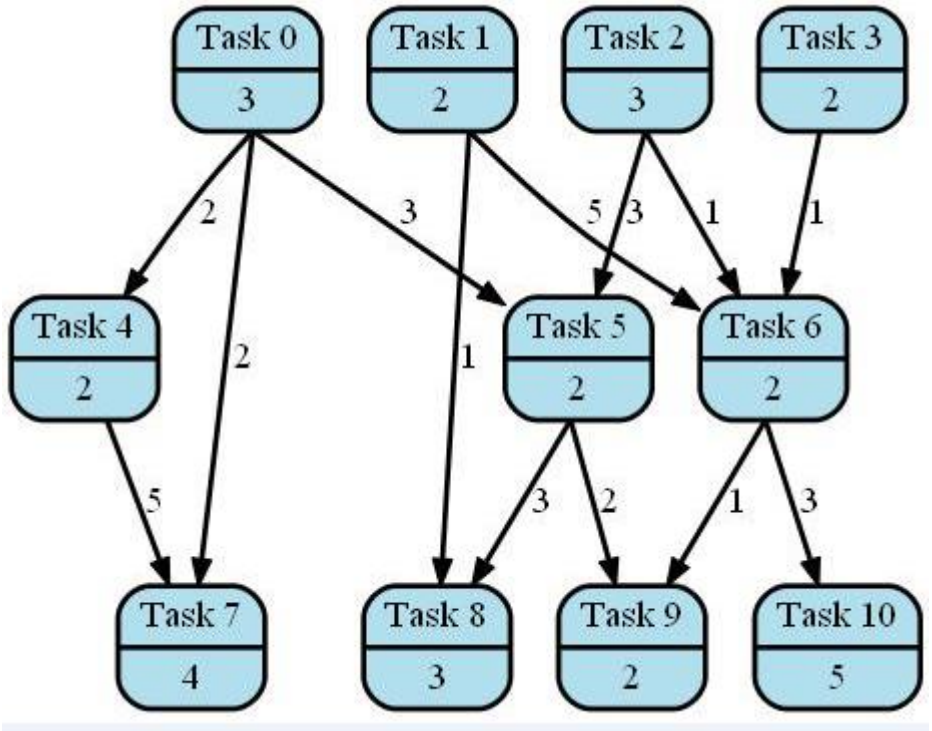
PROCESSOR0| 0| 0| 0| 0| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 8| 8| 8| 8| 8| 8| 8| 8| 8| 8| 6|
6| 6| 6| 6| 6| 6|10|10|10|10|10|10|10|10|

PROCESSOR1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 5| 5| 5| 5| 5| 5| 5| 2| 2| 2| 2| 4| 4| 4| 4|
4| 4| 4| 4| 4| 4| 7| 7| 7| 7| | | | 9| 9| 9| 9| 9| 9|

Şekil 5.22. 10 görevden oluşan görev çizgesine (Şekil 5.21) ait programın çıktısı



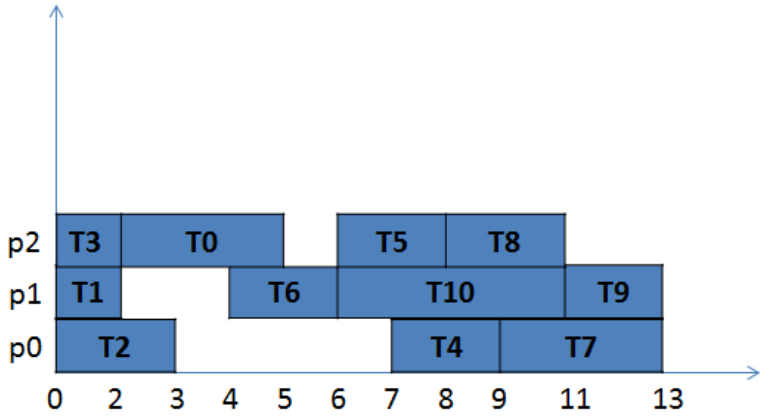
Şekil 5.23. 10 görevden oluşan görev çizgesine (Şekil 5.21) ait Gantt grafiği



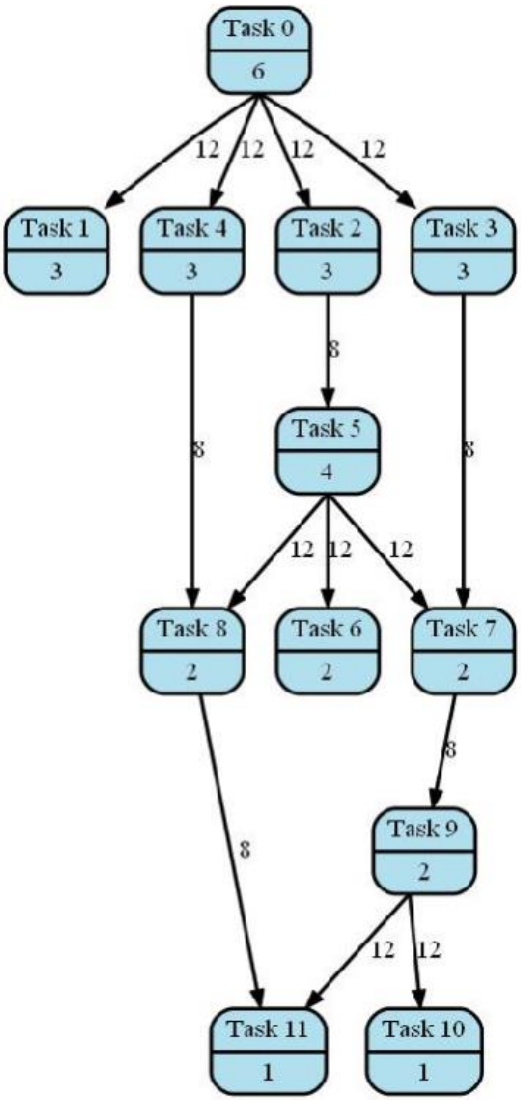
Şekil 5.24. 11 görevden oluşan görev çizgesinden (DAG) bir örnek (Kaur *et al.* 2010)

```
Program çıktısı:
Name of the Task Graph:? TG11Tasks-3.txt
Number of Processors for the Schedule:? 3
THE BEST LATENCY IS -> 13
ELAPSED TIME IS -> 323.705 seconds
THE BEST CHROMOSOME IS -> 10 8 9 7 2 3 5 1 6 0 4 1 2 1 0 0 2 2 1 1 2 0
SCHEDULING TIMES OF TASKS:
TASK0 is scheduled at time interval (2,5)
TASK1 is scheduled at time interval (0,2)
TASK2 is scheduled at time interval (0,3)
TASK3 is scheduled at time interval (0,2)
TASK4 is scheduled at time interval (7,9)
TASK5 is scheduled at time interval (6,8)
TASK6 is scheduled at time interval (4,6)
TASK7 is scheduled at time interval (9,13)
TASK8 is scheduled at time interval (8,11)
TASK9 is scheduled at time interval (11,13)
TASK10 is scheduled at time interval (6,11)
TASKS-PROCESSORS BINDINGS:
TASKS BOUND TO PROCESSOR0 -> 2 4 7
TASKS BOUND TO PROCESSOR1 -> 1 6 10 9
TASKS BOUND TO PROCESSOR2 -> 3 0 5 8
THE GANTT CHART OF THE SCHEDULE:
TIME      | 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|
PROCESSOR0| 2| 2| 2|  |  |  | 4| 4| 7| 7| 7| 7|
PROCESSOR1| 1| 1|  |  | 6| 6|10|10|10|10|10| 9| 9|
PROCESSOR2| 3| 3| 0| 0| 0|  | 5| 5| 8| 8| 8|
```

Şekil 5.25. 11 görevden oluşan görev çizgesine (Şekil 5.24) ait programın çıktısı



Şekil 5.26. 11 görevden oluşan görev çizgesine (Şekil 5.24) ait Gantt grafiği



Şekil 5.27. 12 görevden oluşan görev çizgesi (DAG) (Hashimoto *et al.* 2000)

Program çıktısı:

Name of the Task Graph:? TG12Tasks.txt

Number of Processors for the Schedule:? 4

THE BEST LATENCY IS -> 31

ELAPSED TIME IS -> 342.162 seconds

THE BEST CHROMOSOME IS -> 5 0 7 2 11 1 10 6 4 9 8 3 0 0 0 0 1 2 1 2 1 1 1 0

SCHEDULING TIMES OF TASKS:

TASK0 is scheduled at time interval (0,6)

TASK1 is scheduled at time interval (18,21)

TASK2 is scheduled at time interval (6,9)

TASK3 is scheduled at time interval (13,16)

TASK4 is scheduled at time interval (18,21)

TASK5 is scheduled at time interval (9,13)

TASK6 is scheduled at time interval (25,27)

TASK7 is scheduled at time interval (16,18)

TASK8 is scheduled at time interval (25,27)

TASK9 is scheduled at time interval (27,29)

TASK10 is scheduled at time interval (30,31)

TASK11 is scheduled at time interval (29,30)

TASKS-PROCESSORS BINDINGS:

TASKS BOUND TO PROCESSOR0 -> 0 2 5 3 7

TASKS BOUND TO PROCESSOR1 -> 4 8 9 11 10

TASKS BOUND TO PROCESSOR2 -> 1 6

TASKS BOUND TO PROCESSOR3 ->

THE GANTT CHART OF THE SCHEDULE:

TIME | 0| 1| 2| 3| 4| 5| 6| 7| 8|
9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|

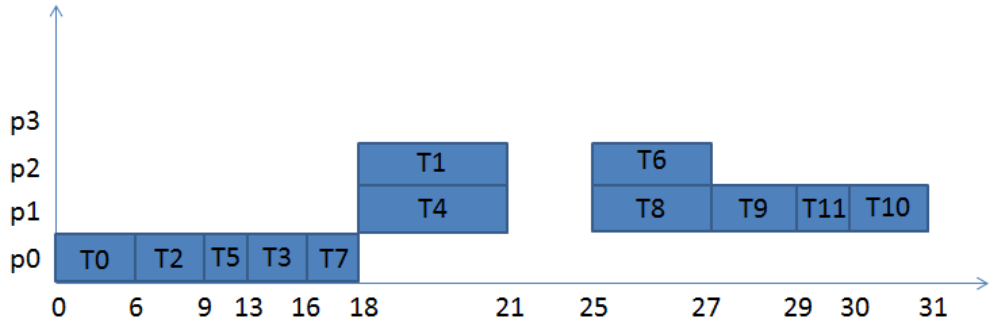
PROCESSOR0| 0| 0| 0| 0| 0| 0| 2| 2| 2| 5| 5| 5| 5| 3| 3| 3| 7| 7|

PROCESSOR1| | | | | | | | | | | | | | | | | | | 4| 4| 4| | | | | 8| 8| 9| 9|11|10|

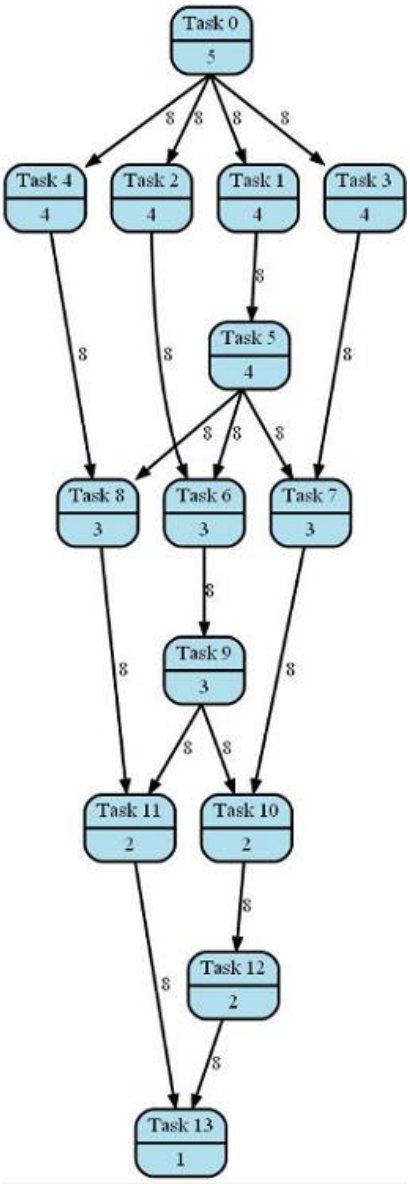
PROCESSOR2| | | | | | | | | | | | | | | | | | | 1| 1| 1| | | | | 6| 6|

PROCESSOR3|

Şekil 5.28. 12 görevden oluşan görev çizgesine (Şekil 5.27) ait programın çıktısı



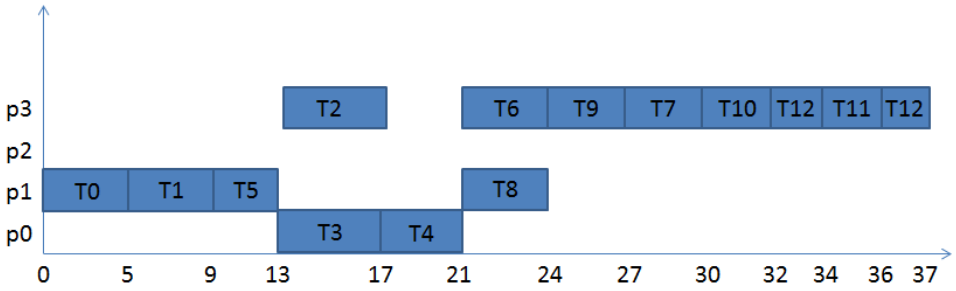
Şekil 5.29. 12 görevden oluşan görev çizgesine (Şekil 5.27) ait Gantt grafiği



Şekil 5.30. 14 görevden oluşan görev çizgesi (DAG) (Hashimoto *et al.* 2000)

```
Program çıktısı:
Name of the Task Graph:? TG14Tasks.txt
Number of Processors for the Schedule:? 4
 THE BEST LATENCY IS -> 37
ELAPSED TIME IS -> 380.033 seconds
THE BEST CHROMOSOME IS -> 10 5 11 12 9 0 3 1 6 2 4 7 8 13 3 1 3 3 3 1 0 1 3 3 0
3 0 3
SCHEDULING TIMES OF TASKS:
TASK0 is scheduled at time interval (0,5)
TASK1 is scheduled at time interval (5,9)
TASK2 is scheduled at time interval (13,17)
TASK3 is scheduled at time interval (13,17)
TASK4 is scheduled at time interval (17,21)
TASK5 is scheduled at time interval (9,13)
TASK6 is scheduled at time interval (21,24)
TASK7 is scheduled at time interval (27,30)
TASK8 is scheduled at time interval (21,24)
TASK9 is scheduled at time interval (24,27)
TASK10 is scheduled at time interval (30,32)
TASK11 is scheduled at time interval (34,36)
TASK12 is scheduled at time interval (32,34)
TASK13 is scheduled at time interval (36,37)
TASKS-PROCESSORS BINDINGS:
TASKS BOUND TO PROCESSOR0 -> 3 4 8
TASKS BOUND TO PROCESSOR1 -> 0 1 5
TASKS BOUND TO PROCESSOR2 ->
TASKS BOUND TO PROCESSOR3 -> 2 6 9 7 10 12 11 13
THE GANTT CHART OF THE SCHEDULE:
TIME      | 0| 1| 2| 3| 4| 5| 6| 7| 8|
9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|36|37|
PROCESSOR0| | | | | | | | | | | | | 3| 3| 3| 3| 4| 4| 4| 4| 8| 8| 8|
PROCESSOR1| 0| 0| 0| 0| 0| 1| 1| 1| 1| 5| 5| 5| 5|
PROCESSOR2|
PROCESSOR3| | | | | | | | | | | | | 2| 2| 2| 2| | | | 6| 6| 6| 9| 9| 9| 7| 7|
7|10|10|12|12|11|11|13|
```

Şekil 5.31. 14 görevden oluşan görev çizgesine (Şekil 5.30) ait programın çıktısı



Şekil 5.32. 14 görevden oluşan görev çizgesine (Şekil 5.30) ait Gantt grafiği

KAYNAKLAR

Anonim 2013a. http://www.baskent.edu.tr/~ayyuce/END407Ders1.pdf.

Anonim 2013b. http://yonerge.blogspot.com/2012/04/genetik-algoritmalar-secim-yontemleri.html/

Anonymous 2012. http://xgrid.blogfa.com/

Adar, N., Kuvat, G., 2012. "Paralel Genetik Algoritmalarda Farklılık ve Geçirgenlik". DPÜ Fen Bilimler Enstitüsü Dergisi, Sayı (27), Issn1302-3055

Bansal, A., ve Kaur, R., 2012. "Task Graph Scheduling on Multiprocessor System using Genetic Algorithm". International Journal of Engineering Research & Technology (IJERT), Vol. 1 Issue 5, ISSN: 2278-0181.

Blelloch, E., ve Maggs, M., 1996. "Parallel Algorithms". Acm Computing Surveys, Volume 28 Issue 1, Pages 51-54.

Cantú-Paz, E., 1998. "A Survey of Parallel Genetic Algorithms". Calculateurs Paralleles, Reseaux et Systems Reparts 10(2), 141—171.

Colouris, G., Doilimoro, J., Kindborg, T., 2011. "Consept and Design Distributed Systems". Cambridge University, 128, London.

Dönmez, F., 2013. "Paralel Programlama". Ahmet Yesevi Üniversitesi.

Daoud, M., Kharma, N., 2008. "A High Performance Algorithm for Static Task Scheduling in Heterogeneous distributed computing systems". Parallel Distrib, Comput(68), 399- 409.

Ercan, U., Akar, H., Koçer, A., 2009. "Paralel Programlamada Kullanılan Temel Algoritmalar". Akdeniz Üniversitesi.

Eraslan, G., 2007. "Paralel Programlama ve MPI". İstanbul Bilgi Üniversitesi.

Ghader, H., Fakhr, K. ve Arzil, S., 2010. "A Hybrid Method for Task Scheduling". Education Technology and Computer, Volume(1), 91-95.

Gülsün, B., Tuzkaya, G., Duman, C., 2009. "Genetik Algoritmalar ile Tesis Yerleşimi Tasarımı ve Bir Uygulama". Doğuş Üniversitesi Dergisi, 10 (1), 73-87.

Gözütok, S., Özdemir, O., 2004. "Genetik Algoritma Yöntemi ile Su Şebekelerinde Hidrolik Kalibrasyonun Geliştirmesi". Gazi Üniv. Müh. Mim. Fak. Der, 19(2), 125-130.

Gökay, E., Çağatan, T.,2002. "Genetik algoritmalar ve uygulama alanlari". İktisadi ve İdari Bilimler Fakültesi Dergisi, XXI (1), 129-152.

Ghader, H., Lotfi, S., Esfahlan, M., 2008. "An Overview of Some Intelligent Optimization Methods". Machine Intelligence Research Labs, 210, Iran.

Garey, M., Johnson, D., 1979. "Computers and Instractability: A Guid to Theory of NP\Completeness". Freeman, 140, USA,

Hagras, T., ve Janeček, J., 2003. "Static vs. Dynamic List-Scheduling Performance Comparison". Acta Polytechnica, 43 (6), 34-42.

Hwang, R., Gen, M., ve Katayama, H., 2008. "A comparison of multiprocessor task scheduling algorithms with communication costs". Computers & Operations Research, 35(1), 976 – 993.

Hashimoto, K., Tsuchiya, T., ve Kikuno, T., 2000. "Fault-Secure Scheduling of Arbitrary Task Graphs to Multiprocessor Systems". Dependable Systems and Networks, 529 (1), 203-212.

Hwang, R., Gen, M., ve Katayama, H., 2006. "A Performance Evaluation of Multiprocessor Scheduling with Genetic Algorithm". Asia Pacific Management Review, 11(2), 67-72.

Kwok. Y., ve Ahmad, I., 1999. "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors". ACM Computing Surveys, Volume 31 (4), 406–471.

Kaur, k., Chhabra, A., ve Singh, G., 2010. "Heuristics Based Genetic Algorithm for Scheduling Static Tasks in Homogeneous Parallel System". International Journal of Computer Science and Security (IJCSS), Volume (4), Issue (2).

Kim, J., Jeong, H., Lee, H., ve Park, J., 2007. "PC Cluster based Parallel PSO Algorithm for Optimal Power Flow". Intelligent Systems Applications to Power Systems, 8(5), 1-6.

Liu, H., li, P., ve wen, Y., 2006. "Parallel Ant Colony Optimization Algorithm". Proceedings of the 6th World Congress on Intelligent Control and Automation, Volume(1) 3222 – 3226.

Lozano, M., Herrera, F., ve Cano, J., 2005. "Replacement Strategies to Maintain Useful Diversity in Steady-state Genetic Algorithms". Springer verlag Berlin Heidelberg, Advances in Soft Computing (1), 85-96.

Mostafa, R., Medhat, H., ve awadalla, A., 2011. "Hybrid Algorithm for Multiprocessor Task Scheduling". IJCSI International Journal of Computer Science Issues, 8(3), 1694-0814.

Mccreary, CL., Khan, AA., Thompson, J., Mcardle, ME ., 1994. "A Comparison of Heuristics for Scheduling DAGs on Multiprocessors". Department of Computer Science and Engineering AL 36849, 446-451.

Nedjah, N., Mourelle, L., ve Alba, E., 2006. "Parallel Evolutionary Computations". Springer Berlin Heidelberg, 208, New York.

Nezhad, V., Gader, H., Efimov, E., 2011. "A Novel Hybrid Algorithm for Task Graph Scheduling". IJCSI International Journal of Computer Science Issues, 8(2), 1694-0814.

Polratanasuk, P., mesacharoenwong, P., anantasate, S., ve leeprechanon, N., 2010. "Solving Optimal Power Flow Problem Using Parallel Bee Algorithm". Selected Topics in Power Systems and Remote Sensing,12(2), 60-64.

Parpinelli, R., Benitez, C., ve Lopes, H., 2010. "Parallel Approaches for the Artificial Bee Colony Algorithm". Handbook of Swarm Intelligence Adaptation, Learning, and Optimization Volume 8, 329-345.

Rajak, R., 2012. "Comparison of Bounded Number of Processors (BNP) Class of Scheduling Algorithms Based on Matrices". GESJ: Computer Science and Telecommunications, 2(34), 35-44.

Sarkar, V., 1987. "Partitionning and Scheduling Parallel Programs for Execution on Multiprocessors". Stanford University Stanford, 213, USA.

Tang, J., Lim, M., Ong, Y., ve Er, M., 2006. "Parallel Memetic Algorithm With Selective Local Search for Large Scale Quadratic Assignment Problems". International Journal of InnovativeComputing, Information and Control, 2 (6), 1349-4198.

Temirci, G., 2009. "Cluster computing" . Dokuz Eylul University

Ütün, O., 2009. "Genetik Algoritma Kullanılarak ileri Beslemeli bir Sinir Agenda Etkinlik Fonksiyonlarının Belirlenmesi". Mühendislik Bilimleri Dergisi, 15 (3),

395-403.

Vanneeschi, L., codecasa, D.,ve mauri, G., 2011. "A Comparative Study of Four Parallel and Distributed PSO Methods". New Generation Computing, 29(2), 129-161.

Wille, E., Yabcznski, E., ve Lopes, H., 2011. " Discrete Capacity Assignment in IP Networks Using Particle Swarm Optimization". Applied Mathematics and Computation, 217(12), 5338–5346.

Wang, J., Cai, Y., Zhou, Y., Wang, R., ve Li, C., 2011. "Discrete Particle Swarm Optimization Based on Estimation of Distribution for Terminal Assignment Problems". Computers & Industrial Engineering, 60(4), 566–575.

Yin, p., Yu, S., Wang, P., ve Wang, Y., 2006. "A Hybrid Particle Swarm Optimization Algorithm for Optimal Task Assignment in Distributed Systems". Computer Standards & Interfaces, 28(4), 441–450.

Yang, Z., ve Yu, B., 2007. "A Parallel Ant Colony Algorithm for Bus Network Optimization". Computer-Aided Civil and Infrastructure Engineering, 22(1), 44-55.

Yaeger, L., 2013. "Artificial life as an Approach to Artificial Intelligence". Intro to Genetic Algorithms, Lecture 3. Professor of informatics indiana university, Indiana, I400-I590.

Yalçın, N., 2011. "Genetik Algoritmalar". Bilecik üniversitesi.

Yavuz, G., Aytekin, S., Akçay, M., 2012. "Apache Hadoop ve Dağıtık Sistemler Üzerindeki Rolü". Fen Bilimleri Enstitüsü, 12 (27), 24-43.

ÖZGEÇMİŞ

Raşid Moradi 1983 yılında İran' da doğdu. İlk, orta ve lise tahsilini Tebriz' de tamamladı. 2007 yılında Tebriz Üniversitesi' nden mezun oldu. 2010 yılında Atatürk Üniversitesi Fen Bilimleri Ensitüsünde yüksek lisans öğrenimine başladı.