DONANIM-YAZILIM BÖLÜŞTÜRMESİ İÇİN YENİ BİR SEZGİSEL ALGORİTMA

Seda ERDEN DERTLİ

Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Dr. Öğr. Üyesi Deniz DAL
2019

ATATÜRK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

DONANIM-YAZILIM BÖLÜŞTÜRMESİ İÇİN YENİ BİR SEZGİSEL ALGORİTMA

Seda ERDEN DERTLİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ERZURUM
2019

T.C.
ATATÜRK ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü Müdürlüğü

TEZ ONAY FORMU

DONANIM-YAZILIM BÖLÜŞTÜRMESİ İÇİN YENİ BİR SEZGİSEL ALGORİTMA

Dr. Öğr. Üyesi Deniz DAL danışmanlığında, Seda ERDEN DERTLİ tarafından hazırlanan bu çalışma, 28/10/2019 tarihinde aşağıdaki jüri tarafından Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Bilim Dalı'nda Yüksek Lisans tezi olarak **oybirliği** / ~~**oy çokluğu**~~ (3./0) ile kabul edilmiştir.

Başkan: Dr. Öğr. Üyesi Deniz DAL İmza :

Üye : Prof. Dr. Abdulsamet HAŞILOĞLU İmza :

Üye : Dr. Öğr. Üyesi Saffet Gökçen ŞEN İmza :

Yukarıdaki sonuç;

Enstitü Yönetim Kurulu'nun 07.11./2019 tarih ve ..43.../13..... nolu kararı ile onaylanmıştır.

Prof. Dr. Mehmet KARAKAN
Enstitü Müdürü

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

ÖZET

Yüksek Lisans Tezi

DONANIM-YAZILIM BÖLÜŞTÜRMESİ İÇİN YENİ BİR SEZGİSEL ALGORİTMA

Seda ERDEN DERTLİ

Atatürk Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr. Ü. Deniz DAL

Donanım ve yazılım bileşenlerini bünyesinde barındıran gömülü sistemlerin kullanım alanlarına son kullanıcının hayatını kolaylaştıran tüketici elektroniği, otomotiv sektörü, savunma sanayi, tıp ve uzay araştırmaları gibi geniş bir yelpazeden örnekler vermek mümkündür. Bu sistemlerin işlevlerini yerine getirirken donanım ve yazılım bileşenlerinden hangisini kullanacaklarının tespit edilmesi sistemin genel performansı açısından önem arz etmektedir. Söz konusu bu tespit işlemi bölüştürme olarak adlandırılmaktadır.

Bir gömülü sistem uygulamasının donanım bileşeni çalışma hızı ve paralel işlem yürütebilme yeteneği nedeniyle yazılıma göre daha yüksek performans sağlar. Öte yandan yazılım, üzerinde kolaylıkla değişiklik yapılabilme esnekliğine sahiptir. Bahsedilen bu avantajlar ve dezavantajlar dikkate alınarak bir sistemi en efektif şekilde çalışır hale getirmek uygulamanın donanım maliyeti ile çalışma süresi arasında bir ödünleşim anlamına gelmektedir.

Bu tez kapsamında donanım-yazılım bölüştürmesi için yeni bir sezgisel algoritma geliştirilmiştir ve performansı Jemai *et al.* (2017) ile verilen sezgisel algoritmayla karşılaştırılmıştır. Bu algoritmaların efektifliğini mukayese edebilmek için her bir deneysel teste ait optimum sonuç bir kaba kuvvet algoritması ile bütün muhtemel bölüştürme kombinasyonları denenerek ayrıca elde edilmiştir. Deneysel bulgular önerilen algoritmanın daha kısa sürede daha iyi sonuçlar ürettiğini göstermektedir.

Deneysel karşılaştırmalar yine bu tezin bir çıktısı olan 5 farklı denektaşı kullanılarak yapılmıştır. Literatürde bu alanda rastlanılan yayınlarda standart bir denektaşı kümesinin olmadığı gerçeğinden hareketle bu denektaşı setinin literatürdeki ilgili boşluğu dolduracağı ve bu nedenle teze özgün bir değer kattığı değerlendirilmektedir.

2019, 58 sayfa

Anahtar Kelimeler: donanım, yazılım, bölüştürme, sezgisel, algoritma, metasezgisel, optimizasyon

ABSTRACT

MS Thesis

A NEW HEURISTIC ALGORITHM FOR HARDWARE-SOFTWARE PARTITIONING

Seda ERDEN DERTLİ

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

Supervisor: Assist. Prof. Dr. Deniz DAL

Embedded systems that incorporate hardware and software components are extensively used in a wide range of applications such as the automotive and defense industry, the medicine and space research and the consumer electronics that make the end-user's life easier. Determining either hardware or software component to be used in performing these functions is important for the overall performance of the system. This detection process is called partitioning.

The hardware component of an embedded system application provides higher performance than its software counterpart due to its operating speed and ability to execute parallel tasks. On the other hand, the software offers more flexibility when it becomes necessary to make a modification. By taking these advantages and disadvantages into consideration, maintaining a system operational in a most effective manner means a trade-off between the hardware cost and the operating time of the application.

Within the scope of this thesis, a new heuristic algorithm for the hardware-software partitioning has been developed and its performance has been compared with by Jemai *et al.* (2017). In order to analyze the effectiveness of these algorithms, the optimum result of each experiment was also obtained by testing all possible partitioning combinations with a brute force algorithm. Experimental findings prove that the proposed algorithm produces better results in less time.

Experimental comparisons were performed by using 5 different benchmarks, which is another output of this thesis. Based on the fact that there is no standard set of benchmarks in the literature in this field, we believe that this benchmark set will fill the relevant gap in the literature and therefore add a unique value to the thesis.

2019, 58 pages

Keywords: hardware, software, partitioning, heuristic, algorithm, metaheuristic, optimization

TEŞEKKÜR

Tezimin konusunun belirlenmesinde, araştırma aşamasında, yönlendirilmesinde ve tamamlanmasında destek olan değerli hocam ve tez danışmanım Sayın Doktor Öğretim Üyesi Deniz DAL'a bana ayırdığı değerli zamanı ve sağladığı destek için teşekkür ederim.

Tezimin başlangıcından bitimine kadar bana inanan, benden yardımlarını esirgemeyen, her zaman yanımda olan başta eşime ve aileme, son olarak da gösterdikleri sabır ve verdikleri her türlü destek için arkadaşlarıma teşekkür ederim.

Seda ERDEN DERTLİ
Ekim, 2019

İÇİNDEKİLER

SİMGELER ve KISALTMALAR DİZİNİ

BENCHMARK	Denektaşı
BSB	Temel Programlama Bloğu (Basic Scheduling Block)
CDFG	Kontrol Veri Akış Şeması (Control Data Flow Graph)
CSP	Kısıt Memnuniyet Problemi (Constraint Satisfaction Problem)
GA	Genetik Algoritma
HW	Donanım (Hardware)
MGPA	Modifiye Genetik Bölümleme Algoritması (Modified Genetic
NGSA	Yeni Genetik Benzetilmiş Tavlama Algoritması (New Genetic Partitioning Algorithm)
PSO	Partikül Sürü Optimizasyonu (Particle Swarm Optimization) Simulated Annealing Algorithm)
SW	Yazılım (Software)

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Gömülü sistem uygulamaları günümüzde hayatın her alanında rastlanır hale gelmiştir. Donanım ve yazılım bileşenlerini bünyesinde barındıran bu tür sistemlerin kullanım alanlarına son kullanıcının hayatını kolaylaştıran tüketici elektroniği, otomotiv sektörü, savunma sanayi, tıp ve uzay araştırmaları gibi geniş bir yelpazeden örnekler vermek mümkündür. Bu sistemlerin işlevlerini yerine getirirken donanım ve yazılım bileşenlerinden hangisini kullanacaklarının tespit edilmesi sistemin genel performansı açısından önem arz etmektedir. Söz konusu bu tespit işlemi bölüştürme olarak adlandırılmaktadır.

Bir gömülü sistem uygulamasının donanım bileşeni çalışma hızı ve paralel işlem yürütebilme yeteneği nedeniyle yazılıma göre daha yüksek performans sağlar. Öte yandan yazılım, üzerinde kolaylıkla değişiklik yapılabilme esnekliğine sahiptir. Bahsedilen bu avantajlar ve dezavantajlar dikkate alınarak bir sistemi en efektif şekilde çalışır hale getirmek uygulamanın donanım maliyeti ile çalışma süresi arasında bir ödünleşim (trade-off) anlamına gelmektedir.

Donanım-yazılım bölüştürmesi, ilgili alt sistem için gerekli işlevselliğin donanım veya yazılımdan hangisinde daha avantajlı bir şekilde uygulanacağına karar verme sürecidir. Burada amaçlanan, toplam sistem gereksinimleri dâhilinde gerekli performansı (maliyet, zaman vb.) sağlayacak bir bölüştürme elde etmektir. Donanım-yazılım bölüştürmesi çok değişkenli ve NP-hard sınıfında kategorize edilen bir optimizasyon problemidir (Abdelhalim 2009). Bu problemin çözümü için ise genellikle iki farklı yöntemden faydalanılmaktadır. Bunlardan birincisi kesin çözümü veren algoritmalardan (genelde çok uzun sürede sonuç üretirler.), ikincisi ise yaklaşık çözüm sunan algoritmalardan (kısa sürede optimumu ya da optimuma yakın bir sonucu bulurlar.) yararlanır. Öte yandan yaklaşık çözüm sunan algoritmalar da kendi içlerinde iki ayrı kategoriye ayrılmaktadır: sezgisel ve metasezgisel yöntemler. Bu tez kapsamında donanım-yazılım bölüştürmesi için yeni bir sezgisel algoritma geliştirilmiştir ve performansı Jemai *et al.* (2017) ile verilen sezgisel algoritma ile karşılaştırılmıştır. Söz konusu bu karşılaştırma

yine bu tezin bir çıktısı olan 5 farklı denektaşı kullanılarak yapılmıştır. Literatürde bu alanda rastlanılan yayınlarda standart bir denektaşı kümesinin olmadığı gerçeğinden hareketle bu denektaşı setinin literatürdeki ilgili boşluğu dolduracağı ve bu nedenle teze özgün bir değer kattığı değerlendirilmektedir.

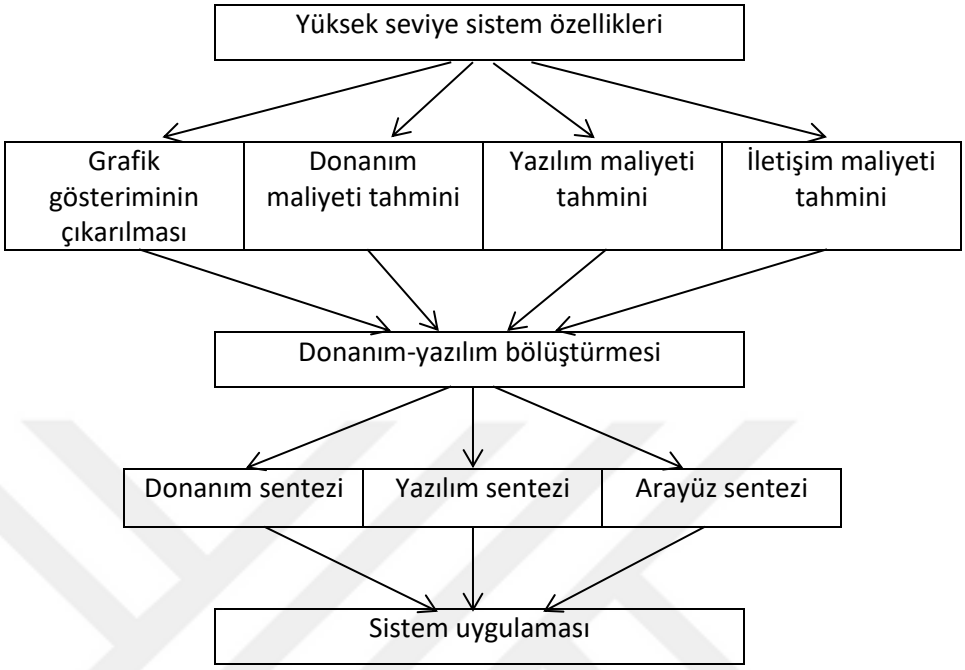
Bu tezin geri kalan bölümü şu şekilde düzenlenmiştir. 1. bölümün alt bölümlerinde bazı temel kavramlar açıklanmıştır. 2. bölümde kaynak özetlerine yer verilmiştir. 3. bölümde materyal ve yöntem incelenmiştir. 4. bölümde araştırma bulgularına yer verilmiştir. 5. bölüm ise tezin sonuç bölümüdür.

1.1. Optimizasyon

En iyileme olarak da bilinen optimizasyon kavramı bir maliyet/amaç fonksiyonunun en büyük veya en küçük değerini genellikle belirli kısıtlar altında bulma işlemini ifade etmektedir. İşletme, tasarım, ekonomi, üretim, mühendislik ve sağlık gibi hemen hemen her bilim dalında kendisine uygulama alanı bulabilmektedir.

1.2. Donanım-Yazılım Bölüştürmesi

Bir sistemin sadece yazılım ile gerçeklendiği durumlarda yüksek ihtimalle zaman kısıtını karşılayamayacağı, benzer şekilde ilgili sistemin sadece donanım ile tasarlandığı durumlarda da maliyetli bir çözüm üreteceği bilinmektedir. Bu nedenle sistem tasarımında donanım ve yazılım bileşenlerinin dengeli ve efektif bir şekilde kullanılmasına yönelik bir bölüştürme gerekliliği ortaya çıkmaktadır.



Şekil 1.1. HW-SW tasarım süreci (Govil 2016)

Şekil 1.1 ile verilen tasarım süreci tipik bir donanım-yazılım bölüştürmesi sürecidir (Govil 2016). Bu bölüştürme, bir uygulamaya ait hesaplamaları iki gruba bölme işlemidir: hangi işlemler mikroişlemci ("yazılım") üzerinde sıralı komutlar olarak çalışacaktır, hangi işlemler ASIC'ler veya FPGA'lar gibi bir donanım üzerinde gerçeklenecektir?

Yıllar boyunca, bölüştürmenin verimli kullanılmasının, yalnızca yazılım çözümlerine kıyasla performansta çok daha üstün tasarımlara yol açacağı gösterilmiştir (Stitt 2004; Lysecky 2005). Bunun temel nedeni, yazılımın daha esnek ve daha ucuz olmasına rağmen, donanımın oldukça hızlı olması ve daha az güç tüketmesidir. Bu nedenle, hesaplama yoğunluğu olan (veya kritik) görevleri donanıma ve kritik olmayanları da yazılıma yerleştirerek bölmek en iyi performansla sonuçlanmaktadır. Bu, mevcut kaynakların verimli kullanılmasını sağlamakta, güç tüketimini azaltmakta ve sistemin genel uygulama süresini hızlandırmaktadır (Govil 2016).

1.3. Bölüştürme Probleminin Çözümü İçin Kullanılan Yöntemler

Bölüştürme probleminin çözümünde kullanılan iki tür yöntem vardır: manuel ve otomatik yöntemler (Koudil 2007).

Manuel yaklaşımlarda, tasarımcı, yeteneklerini ve tasarımdaki sistem bilgisini, varlıkların mimari işlemciler üzerinde en iyi şekilde eşleştirilmesinde etkileşimli bir şekilde kullanır.

Otomatik yaklaşımlarda, tasarımcılara en düşük maliyeti veren bölüştürme çözümlerini bulmalarında yardımcı olan algoritmalar kullanılır. Otomatik bölüştürme yaklaşımları ise iki sınıfa ayrılabilir: ayrıntılı arama yöntemleri ve yaklaşık yöntemler.

1.3.1. Ayrıntılı arama yöntemleri

Ayrıntılı arama yöntemleri, tüm olası çözümlerin belirlenmesi ve değerlendirilmesi prensibine dayanır. Bu tür yöntemlerin gerçeklenmesi kolaydır fakat çözüm uzayı genişledikçe optimum çözüme ulaşmak için gerekli süre artmaktadır ve çoğu zaman çözüm bulunamayacak hale gelebilmektedir. Bu nedenle bu kategorideki yöntemler çok küçük boyutlu problemler için idealdir, fakat büyük boyutlu problemler için kullanılması bir dezavantaj oluşturmaktadır.

1.3.2. Yaklaşık yöntemler

Yaklaşık yöntemler, ‘kabul edilebilir’ bir sürede bir (veya daha fazla) çözüm(ler) elde edebilmeyi sağlar. Bu yöntemleri özel ve genel problemlere uygulanabilirlikleri noktasında iki gruba ayırmak mümkündür.

1.3.2.a. Sezgisel yöntemler

Bu tür yöntemlerin en büyük avantajı belirli bir problem için "uyarlanmış" olmalarıdır. Bununla birlikte, bu yöntemlerin özel olarak geliştirildikleri sistem üzerinde yapılacak küçük bir değişikliğe bile hassas oldukları ve böyle bir durumda tasarım sürecinin yeniden işletilmesi gerektiği dezavantajını taşıdıkları bilinmektedir.

1.3.2.b. Metasezgisel yöntemler

Sezgisel yöntemlerin aksine metasezgisel yöntemler belirli bir problemin çözümüne yönelik geliştirilmemişlerdir ve NP-hard olarak sınıflandırılan farklı türde birçok optimizasyon probleminin çözümünde yaygın olarak kullanılmaktadır. Bu yöntemlerin en büyük avantajı çözüm uzayını sömürü (exploitation) ve keşif (exploration) mekanizmaları ile tarayabilme ve bu sayede lokal optimumdan kaçabilme yeteneğine sahip olmalarıdır.

Sürü Zekâsı

Sürü zekâsı, âdemi merkeziyetçi sistemlerin kolektif davranışını ifade eder ve hem doğal hem de yapay sistemleri tanımlamak için kullanılabilir (Anonymous 2019). Bu sistem sınıfı için spesifik algoritmalar arasında parçacık sürüsü optimizasyon algoritması, karınca kolonisi optimizasyon algoritması ve yapay arı kolonisi algoritması bulunur. Bu algoritmaların her birinin, hayvanların doğal, kendi kendini organize eden davranışlarından esinlendiği bilinmektedir.

Tabu Arama

Bu metasezgisel yöntem, çözüm aramasını optimum çözümlere yönlendirmek için dinamik olarak oluşturulan tabu tablosundan faydalanır. Bu esnada bir problemin potansiyel çözümlerini inceler ve gelişmiş bir çözüm bulmak için yerel komşuları

kontrol eder. Tabu arama algoritması, dinamik olarak kurallar dizisi oluşturur ve kural ihlal edici çözümleri "tabu" veya yasak olarak işaretleyerek sistemin aynı alanda gereksiz arama yapmasını önler. Bu yöntem, arama işlemi en düşük bölgelerde veya çok sayıda eşit çözüm bulunduran alanlarda sıkışıp kaldığında yerel arama yöntemlerini devreye alır.

Benzetilmiş Tavlama

Benzetilmiş tavlama ya da benzetimli tavlama, ilhamını metalürji biliminde malzemelerin mükemmel kristal yapıya ulaşabilmelerini sağlayan soğutma tekniğinden alan ve en iyileme problemleri için tasarlanmış olasılıksal yaklaşımlı bir yöntemdir. Diğer olasılıksal yaklaşımlar gibi en iyi çözümün en kısa zamanda üretimini hedefler. Benzetilmiş tavlama, global optimizasyonda kullanılır ve geniş bir arama alanına sahip bir fonksiyon için global optimumun makul bir yaklaşımını verebilir. Her bir yinelemede, muhtemelen sistemi en düşük enerji durumuna getirirken mevcut durumda kalma veya bir başkasına geçme arasında seçim yapar. Maliyet fonksiyonunu iyileştiren komşu çözümler koşulsuz olarak kabul edilirler. Öte yandan lokal optimumdan kaçabilmek amacıyla kötü komşu çözümlerin de sıcaklığa bağlı olarak olasılıksal kabul edilme durumu söz konusudur.

Genetik Algoritma

Genetik Algoritma (GA) geniş çözüm uzayında stratejik aramalar yapmak için kullanılan bir yöntemdir. GA'da ilgili optimizasyon problemini çözebilmek için başlangıçta rastgele bir çözüm kümesi (popülasyon) oluşturulmaktadır. Bu popülasyonun her bir bireyi, yani her bir çözüm, bir kromozom olarak adlandırılmaktadır ve ilk olarak her bir kromozomun ilgili amaç fonksiyonu kullanılarak bir uygunluk değeri (fitness value) hesaplanmaktadır. Daha sonra çaprazlama işlemi için rastgele iki kromozom şeçilmekte ve yine rastgele seçilen bir noktadan belirli bir orana göre çaprazlanmaktadır. Mutasyon ise yine belirli bir oranda rastgele seçilmiş kromozomlar üzerinde küçük değişiklikler yapılması prensibine dayanmaktadır. Bu

işlem belli bir yürütme süresine veya iterasyon sayısına ulaştığında algoritma sonlandırılmaktadır (Nesmachnow *et al.* 2010).

2. KAYNAK ÖZETLERİ

Bu bölümde literatürde rastlanan ve donanım-yazılım bölüştürmesi problemini ele alan sezgisel ve metasezgisel yöntemlerinden bahsedilmiştir.

Saha *et al.* (1997) tarafından yürütülen çalışmada, donanım-yazılım bölüştürmesi problemi bir Kısıt Memnuniyet Sorunu (CSP) olarak modellenmiştir ve CSP'yi çözmek için genetik algoritma tabanlı bir yaklaşım sunulmuştur. Çözümde kullanılan açgözlü ve metasezgisel yaklaşımlar karşılaştırılmış ve GA'nın daha iyi performans sergilediği belirtilmiştir.

Hidalgo *et al.* (1997) tarafından gerçekleştirilen çalışmada, genetik algoritma kullanılarak donanım-yazılım kod tasarımının işlevsel bölüştürme yönü ele alınmıştır. Bu çalışma üç algoritma ve karşılaştırmalı bir yapı içermektedir: benzetilmiş tavlama (SA), Fiduccia-Matheyses (FM) ve sonuçların iyileştirilmesiyle bu algoritmaların yeniden tasarlanmış yeni bir versiyonu. Sistem bölüştürme için yapısal bölüştürme (SP) ve işlevsel bölüştürme (FP) olmak üzere iki farklı yöntemin varlığından bahsedilmektedir. Yapısal bölüştürmede, sistem ilk önce bloklar halinde bölümlendirilmektedir. Her ne kadar SP çok popüler olsa da, sonuçlarının sadece donanım çözümleri sağlamasının bir dezavantaj olduğu belirtilmektedir. Öte yandan, FP'nin avantajı olarak Donanım-Yazılım Kod Tasarımı için iyi bir tekniğe sahip olması ve donanım uygulamalarının yanı sıra yazılım çözümleri elde edebilmesi gösterilmektedir. Bu bahsedilen avantajlarından dolayı bu araştırmada gömülü sistemler için FP'den faydalanılmaktadır. FM, klasik bir bölüştürme algoritmasıdır. Tamamı yazılım çözümünden başlayarak, minimum Objektif Fonksiyonlarını (OF) ve ilgili çözümü saklamaktadır. OF'yi geliştiren bloklar seçilmekte ve kilitlenmektedir. Bu işlem, tüm bloklar kilitleninceye ve OF'de bir gelişme olmayıncaya kadar tekrarlanmaktadır. Bu yöntem sadece her adımda bir bloğun hareketine izin vermektedir. Bu araştırmada kullanılan diğer bir algoritma (Modified Fiduccia-Matheyses (MFM)) ise şöyle açıklanmaktadır: MFM, FM'nin iletişim maliyetlerine özel bir versiyonudur. Bunun yanında tek bir bloktan daha fazlasının göçüne izin

vermektedir. Böylece diğer algoritmaların sonuçlarını iyileştirmektedir. Ancak birkaç zaman kısıtlaması sorununa çözüm bulma konusunda bazı problemleri vardır. Açıklanan algoritmaların en uygun çözümü elde etmesi için, bu araştırmada SA, FM ve MFM sonuçlarını iyileştiren yeni bir bölüştürme tekniği geliştirilmektedir. Bu teknik, optimal çözümler arayışında genetik bir algoritma kullanılmasına dayanmaktadır. SA'da kullanılan maliyet fonksiyonlarının GA'larla daha iyi sonuçlar üretebileceğinden bahsedilmektedir.

Zou *et al.* (2004) tarafından yürütülen çalışmada, Sistemin Temel Programlama Bloğu (BSB) grafını baz alan bir donanım-yazılım bölüştürmesi modeli oluşturulmuştur ve bir Modifiye Genetik Bölümleme Algoritması (MGPA) önerilmiştir. Bu algoritmanın HW-SW bölüştürmesi problemini çözmede etkin bir rol oynadığı rapor edilmiştir.

Farahani *et al.* (2006) tarafından gerçekleştirilen çalışmada, donanım-yazılım bölüştürmesi probleminde en uygun çözümlere ulaşmak için paralel genetik algoritma (PGA) tabanlı bir yaklaşım sunulmuştur. Önerilen sistemin etkinliğini değerlendirmek için, literatürde yaygın olarak kullanılan Task Graphs For Free (TGFF) aracından faydalanılmıştır. Deneysel sonuçlar, önerilen yaklaşımın en uygun maliyet çözümlerini, kabul edilebilir bir zamanda bulabildiğini göstermiştir.

Farmahini-Farahani *et al.* (2007) tarafından yürütülen çalışmada, belirli bir görev grafının donanım-yazılım bölüştürmesi sürecini minimum maliyetle gerçekleştirmek için Parçacık Sürüsü Optimizasyonunu (PSO) temel alan bir algoritma tasarlanmıştır. Bu çalışma sonucunda en uygun çözüme yaklaşık sonuçlar elde edildiği belirtilmiştir. Farklı graflar üzerinde denenen bu çalışma genetik algoritma ile karşılaştırılmıştır. Önerilen çözümün her durumda genetik algoritmadan daha iyi sonuçlar verdiği gözlemlenmiştir.

Abdelhalim *et al.* (2007) tarafından gerçekleştirilen çalışmada, donanım uygulamalarının alternatiflerinin ve PSO tabanlı bir donanım-yazılım bölüştürmesindeki iletişim maliyetinin modellenmesi araştırılmıştır. İletişim maliyetinin modellemesinde,

alan değeri de maliyet hesabına katıldığı için donanım-yazılım arayüzünün gerçekleştirilmesinde hesaplanan maliyet değeri artmaktadır. Yapılan çalışmada 3 farklı deney gerçekleştirilmiştir. İlk çözüm sadece seri donanım düğümleri kullanılarak, ikinci çözüm sadece paralel donanım düğümleri kullanılarak gerçekleştirilmiştir. Son çözüm ise seri ve paralel düğümler kullanılarak gerçekleştirilmiştir. Önerilen programın, kısıtlamaları karşılayarak, iletişim maliyeti ile alan maliyeti arasında ilk iki deneye göre daha iyi bir denge sağladığı gösterilmiştir.

Zhang *et al.* (2008) tarafından yürütülen çalışmada, donanım-yazılım kod tasarımının ana adımlarından birinin donanım-yazılım bölüştürmesi olduğu ve iyi bir bölüştürmenin güç, boyut, performans vb. gibi bazı kısıtlamalara bağlı olduğu belirtilmiştir. Bu çalışmada donanım-yazılım bölüştürmesi için hem negatif seçim modelinden hem de biyolojik bağışıklık sisteminin evrimsel mekanizmasından esinlenerek, ENSA-HSP olarak isimlendirilen evrimsel bir negatif seçim algoritması önerilmiştir. Deneysel sonuçlar, ENSA-HSP'nin geleneksel evrimsel algoritmadan daha verimli olduğunu göstermiştir.

Wu *et al.* (2008) tarafından gerçekleştirilen çalışmada, donanım alanının minimum değerde olması ile uygulama süresini optimize eden bir donanım-yazılım bölüştürmesi problemine odaklanılmıştır. Önerilen algoritmanın, literatürde yer alan ve karşılaştırmaya konu olan diğer bir algoritmaya kıyasla çalışma süresinde bir artış olmadan HW-SW bölüştürmesi problemini çözdüğü gösterilmiştir.

Abdelhalim *et al.* (2009) tarafından yürütülen çalışmada, gömülü sistemlerin tipik olarak uygulamaya özel donanım parçalarından ve programlanabilir parçalardan oluştuğundan bahsedilmiştir. Bazı tasarım yaklaşımlarının [Marrec *et al.* 1998; Cloute *et al.* 1999] HW-SW bölüştürmesi görevini manuel olarak gerçekleştirdiği, bu manuel yaklaşımın küçük tasarım problemlerine uygun olduğu belirtilmiştir. Ayrıca, otomatik donanım-yazılım bölüştürmesinin büyük ilgi çektiğini, çünkü donanım-yazılım bölüştürmesi probleminin çok karmaşık bir optimizasyon problemi olduğundan bahsedilmiştir. Gerçekleştirilen çalışmada PSO ve GA karşılaştırılmıştır. Yeniden

oluşturulan PSO'nun donanım-yazılım bölüştürmesi problemini çözmede oldukça etkili olduğu araştırmacılar tarafından kanıtlanmıştır. JPEG kodlayıcı sistemi, PSO'nun donanım-yazılım bölüştürmesi problemlerini çözmedeki uygunluğunu test etmek için gerçek hayattan bir vaka çalışması olarak kullanılmıştır. Bu çalışmanın sonuçları literatürde yer alan diğer sonuçlarla karşılaştırılmıştır. Deneysel sonuçlar, algoritmanın daha iyi sonuçlar ürettiğini kanıtlamıştır.

Li *et al.* (2010) tarafından gerçekleştirilen çalışmada, genetik algoritmaların analizine ve benzetilmiş tavlama algoritmasının temel avantajlarına/dezavantajlarına dayanan Yeni Genetik Benzetilmiş Tavlama Algoritması (NGSA) isimli bir metasezgisel önerilmiştir. Bu çalışma kapsamında Genetik Algoritma ve Benzetilmiş Tavlama fikri bütünleştirilmiştir. Deneysel sonuçlar, algoritmanın daha güçlü bir global arama kabiliyetine sahip olduğunu ve genetik algoritma ve benzetilmiş tavlama algoritmasını önemli ölçüde iyileştirdiğini göstermiştir.

Ayadi *et al.* (2012) tarafından yürütülen çalışmada, yeniden yapılandırılabilir bir mimari için tasarım sürecine odaklanılmaktadır. Bu çalışma kapsamında yeni bir geçici bölüştürme algoritması tasarlanmıştır. Geçici bölüştürme problemi graf tabanlı bir problem olarak ön görülmüştür. Bu algoritma, geçici bölüştürme problemini çözmek için tipik matematik akışını temel almaktadır ve iki ana adımdan oluşmaktadır. İlk adım, grafın ilk bölüştürmesini bulmayı amaçlamaktadır. Bu adım iletişim maliyeti açısından en uygun çözümü sunmaktadır. Daha sonra, alan kısıtlaması ilk adımdan sonra sağlanıyorsa, ilk bölüştürme değeri kabul edilmekte, aksi takdirde ikinci adıma geçilmektedir. İkinci adım, alanın kısıtlamasını yerine getirirken aynı zamanda grafın son bölümünü bulmayı amaçlamaktadır. İkinci adım uygun bir çözüm bulamazsa, bölüm sayısını birer birer gevşetmekte ve algoritma ilk adıma geri dönmektedir. Sonrasında yeni bölüştürme ile uygun bir çözüm bulmaya çalışmaktadır. Bu algoritma, tasarım bölümleri ve yeniden yapılandırma ek yükü arasında gereken verilerin transferini optimize etmektedir. Sonuç olarak önerilen algoritmanın diğer iyi bilinen algoritmalara kıyasla iletişim maliyetini ve gecikmeyi önemli ölçüde azalttığı rapor edilmektedir.

Wu *et al.* (2013) tarafından gerçekleştirilen çalışmada, yazılım ve iletişim kısıtlamalarını göz önünde bulundurarak donanım maliyetini en aza indirgemeyi amaçlayan bir HW-SW bölüştürmesi için iki yaklaşım sunulmuştur. Bunlardan birincisi, HW-SW bölüştürmesi problemini 0-1 sırt çantası problemi olarak ele alan bir sezgisel yaklaşımdır. İkinci yaklaşım ise, önerilen sezgisel algoritmadan elde edilen çözümü daha da geliştirmek için tabu araması algoritmasından faydalanmaktadır. Deneysel sonuçlar, önerilen algoritmaların, çalışmada belirtilen bir başka literatür çalışmasını %28'e kadar geride bıraktığına işaret etmektedir.

Lin *et al.* (2014) tarafından yürütülen çalışmada, donanım-yazılım bölüştürmesi problemini çözmek için tabu arama tabanlı bir memetik algoritma önerilmiştir. Bu algoritmanın üç ana bileşenden oluştuğu, bu bileşenlerin yerel bir tabu araması, bir yol bağlantı prosedürü ve popülasyon güncellemesi olduğu belirtilmiştir. Tabu aramasının, çözüm üretme açısından özel bir yeteneğe sahip olduğundan ve tabu görev süresinin güncellenmesi için bir geri bildirim mekanizmasının kullanılmış olduğundan bahsedilmiştir. Son olarak algoritmanın iyi sonuçlar verdiği rapor edilmiştir.

Azari *et al.* (2015) tarafından gerçekleştirilen çalışmada, donanım-yazılım bölüştürmesinin, tüm sistem performansını önemli ölçüde etkilediği için gömülü sistemlerin tasarımında her zaman çok önemli bir adım olduğu belirtilmiştir. Bu çalışmada, bir uygulamayı temsil eden Kontrol Veri Akış Grafı (CDFG) içindeki görevleri bölümlemek için yeni bir yaklaşım önerilmiştir. Performansı arttırmak için, sıcak yol ve kritik yol olarak tanımlanan iki ana yolun kombinasyonu göz önünde bulundurulmuştur. Önerilen yaklaşım, sıcak ve kritik yol belirlendikten sonra, kritik yol üzerinde doğrudan etkisi olan yollardaki görevlere daha yüksek öncelikler vererek, donanım bileşenlerine mümkün olduğunca fazla görev atamaktır. Sonuç olarak, bir uygulamanın toplam yürütme süresi kısalmış olmaktadır. Deneysel değerlendirme, önerilen yola dayalı bölümleme yönteminin, performansı önemli ölçüde iyileştirdiğini göstermiştir.

Ouni *et al.* (2015) tarafından yürütülen çalışmada, SOPC için yeni bir donanım-yazılım bölüştürmesi algoritması önerilmektedir. Önerilen algoritmanın mevcut yaklaşımlara kıyasla mümkün olan en düşük maliyetle daha iyi bir performans gösterdiği belirtilmektedir. Algoritma, bölümleme nesnesinin oluşturulması olarak tanımlanan bir fonksiyona dayanmaktadır ve bu fonksiyon ile donanım-yazılım gecikmesi ve donanım alanı hesaplanmaktadır. Bahsedilen fonksiyon birçok bölüştürme oluşturmaktadır (örneğin: %1 donanım, %99 yazılım, %25 donanım, %75 yazılım vb.). Her bölüm nesnesi için gecikme değeri hesaplanmakta ve bu farklı değerler kullanılarak bir eğri çizilmektedir. Ardından, alan maliyeti de hesaplanmakta ve ikinci bir eğri üretilmektedir. Bu iki eğri üst üste geldiğinde, benzersiz bir kesişme noktası elde edilmektedir. Bu nokta, kullanılması gereken en iyi bölüştürme nesnesini ifade etmektedir.

Dimassi *et al.* (2015) tarafından gerçekleştirilen çalışmada, metasezgisel bir yöntem sunulmaktadır. Bu çalışmada, donanımın kullandığı görev sayısını en aza indirmek, yazılımın kullandığı görev sayısını arttırmak ve sistem alanını en aza indirmek ve böylece verimli bir donanım-yazılım bölüştürmesi bulmak için tabu arama ve genetik algoritma birlikte kullanılmıştır. Genetik algoritmanın tabu algoritmaya göre %18,69'a ulaşan bir kazanç sağladığı belirtilmiştir. Algoritmalar arası karşılaştırma, tanımlanan $\beta = \frac{L(G)}{A_{max} - A_L}$ değerine göre yapılmıştır.

Jemai *et al.* (2017) tarafından yürütülen çalışmada, donanım-yazılım bölüştürmesi problemi Tabu Arama, Genetik Algoritma ve ikili arama ağacı temelli bir metasezgisel ile çözülmeye çalışılmıştır. Geliştirilen algoritma, bir zaman kısıtlaması gözetirken, programlanabilir bir çip üzerindeki sistem mantık alanını en aza indiren etkin bir donanım-yazılım bölüştürme optimizasyonunu hedeflemektedir. Öncelikle Zamansal sınırlama (T_{con}) değeri belirlenmektedir ve graf üzerindeki modüller alan değerleri kullanılarak ikiye ayrılmaktadır. Hesaplanan değerden küçük olan modüller sol alt ağaca (LST), büyük olan modüller sağ alt ağaca (RST) atanmaktadır. LST mimarinin donanım, RST ise mimarinin yazılım parçaları olarak belirtilmektedir. Sonrasında çalışma zamanı (T_{ex}) hesaplanmaktadır. T_{con} ve T_{ex} karşılaştırılarak sağ alt ağaç mı

yoksa sol alt ağaç mı kullanılacağı tespit edilmektedir. Eğer T_{con} değeri büyük ise sol alt ağaçta (LST) genetik algoritma ve tabu arama, küçük ise sağ alt ağaçta (RST) genetik algoritma ve tabu arama yapılmaktadır. Eğer durdurma şartı sağlanıyorsa donanım-yazılım bölüştürmesi sonuca ulaşmış olmaktadır. Şartlar sağlanmadı ise belirlenen ağaçlarda genetik algoritma ve tabu arama işlemleri tekrarlanmaktadır. İkili arama ağacı ve genetik algoritmanın beraber kullanıldığı algoritmanın genetik algoritmaya göre %69 kazanıma, ikili arama ağacı ve tabu aramanın beraber kullanıldığı algoritmanın ise tabu aramaya göre %78 kazanıma sahip olduğu vurgulanmaktadır. Kullanılan metodun verimliliğini ölçmek için $A_V = \frac{L}{A_{max} - A_L}$ formülü kullanılmaktadır. Bu formüle göre genetik algoritma ve ikili arama ağacı yönteminin genetik algoritmadan, tabu arama ve ikili arama ağacı yönteminin de tabu aramadan daha performanslı olduğu rapor edilmektedir.

Bu tez kapsamında geliştirilen sezgisel yöntem, bizim bildiğimiz kadarıyla literatürde donanım-yazılım bölüştürmesi için en son geliştirilen sezgisel algoritmadır ve bu algoritmanın etkinliği bir önceki paragrafta detayları verilen sezgisel algoritma ile karşılaştırılarak elde edilmiştir.

3. MATERYAL ve YÖNTEM

Tezin bu bölümünde Jemai *et al.* (2017) ile verilen sezgisel yöntem ve bu tez kapsamında geliştirilen sezgisel algoritma detaylı olarak incelenmiştir. Bu bölümde ayrıca her iki algoritmanın da bilgisayar ortamında gerçeklenmesi için kullanılan yazılımsal bileşenlere yer verilmiştir.

3.1. Donanım-Yazılım Bölüştürmesinde Kullanılan Formüller ve Maliyet Fonksiyonunun Hesaplanması

Donanım-yazılım bölüştürmesi problemi en nihayetinde bir maliyet fonksiyonunu minimize etmeye çalışmaktadır. Bu alt bölümde söz konusu bu maliyet fonksiyonunun hangi bileşenlerden oluştuğu ve kendisine girdi olarak verilen bir çözüm vektörünü hangi adımlar sonrasında bir sonuca dönüştürdüğü detaylı olarak incelenecektir.

Donanım-yazılım bölüştürmesi algoritmaları bir gömülü sistem tarafından gerçeklenecek uygulamaları temsil eden bir Veri Akış Grafını (DFG, Data Flow Graph) girdi olarak alırlar. Çıktı olarak da bu grafın her bir düğümünün donanım veya yazılım ünitelerinin hangisi tarafından işletileceğine karar verirler.

Bir önceki paragrafta kendisinden söz edilen Kontrol Veri Akış Grafının $\boldsymbol{G(V,E)}$ ile temsil edildiğini kabul edelim. Burada $\boldsymbol{V}$ grafın düğümlerinin, $\boldsymbol{E}$ ise düğümler arası sıralı (ordered) kenarların kümesini işaret eder. Bir çözüm vektörüne ait gecikmeyi hesaplamak için kullanılacak notasyon aşağıda yer almaktadır:

V_i	:	Grafın **i.** düğümü
$S_L(V_i)$	:	Grafın V_i düğümünün yazılım gecikmesi
$H_L(V_i)$	:	Grafın V_i düğümünün donanım gecikmesi
$A(V_i)$	:	Grafın V_i düğümünün donanım maliyeti (alan)
$S_L(G)$	:	Grafın yazılım gecikmesi
$H_L(G)$	:	Grafın donanım gecikmesi
$L(G)$	:	Grafın gecikmesi
T_{con}	:	Zamansal kısıt (temporal constraint)
S	:	Çözüm vektörü $S(i) = 0$ ise grafın **i.** düğümü yazılım tarafından gerçeklenir $S(i) = 1$ ise grafın **i.** düğümü donanım tarafından gerçeklenir
A_L	:	Çözümün donanım alanının büyüklüğü
A_{max}	:	Tüm düğümlerin donanım alanlarının toplamı
β	:	Çözümün kalitesinin ölçütü ($L(G)$ ile A_L arasındaki ödünleşim)
T_{ex}	:	Çözümün yazılım ve donanımda gerçekleşen gecikmeleri toplamı

Şekil 3.1. Çözüm vektörüne ait gecikmeyi hesaplamak için kullanılan ifadeler

Bir S çözüm vektörü için ilgili G grafına ait $L(G)$ gecikme değeri aşağıdaki formüller kullanılarak elde edilir:

$$L(G) = S_L(G) + H_L(G)$$	(1)
$$S_L(G) = \sum_{V_i \epsilon G} (1 - S(i)) \times S_L(V_i)$$	(2)
$$H_L(G) = max_{V_i \in G}\{S(i) \times H_L(V_i) + \sum_{V_j \epsilon G} (S(j) \times \alpha_{ij}) \times H_L(V_j)\}$$ $$\alpha_{ij} = \begin{cases} 1, & V_j \ V_i' ye \ bağımlı \ ise \\ 0, & V_j \ V_i' ye \ bağımlı \ değilse \end{cases}$$	(3)
$$A_L = \sum_{V_i \epsilon G} S(i) \times A(V_i)$$	(4)
$$\beta = \frac{L(G)}{A_{max} - A_L}$$	(5)

Şekil 3.2. Çözüm için kullanılan formüller

Yukarıda verilen 1. Formül grafın gecikmesinin grafın yazılım gecikmesi ile grafın donanım gecikmesinin toplamına eşit olduğunu ifade eder.

Yukarıda verilen 2. Formül grafın yazılım gecikmesinin çözüm vektöründe yazılıma atanan düğümlerin yazılım gecikmelerinin toplamı ile hesaplandığını ifade eder.

Yukarıda verilen 3. Formül grafın donanım gecikmesinin iki adımda hesaplandığını ifade eder. Birinci adımda grafın her bir $\boldsymbol{V_i}$ düğümü için eğer bu düğüm çözümde donanıma atanmış ise ilgili düğümün donanım gecikmesi ile grafın $\boldsymbol{V_i}$'ye bağımlı ve çözümde donanıma atanmış diğer tüm düğümlerinin donanım gecikmelerinin toplamı hesaplanır. İkinci adımda ise tüm $\boldsymbol{V_i}$'ler için birinci adımda hesaplanan değerler kontrol edilerek bu değerlerin en büyüğü bulunur ve grafın donanım gecikmesinin değeri olarak kullanılır.

Yukarıda verilen 4. Formül çözümün donanım alanının büyüklüğünün çözümde donanıma atanmış düğümlerin alanları toplanarak bulunduğunu ifade eder.

Yukarıda verilen 5. Formül çözümün $\boldsymbol{\beta}$ değerini ifade eder.

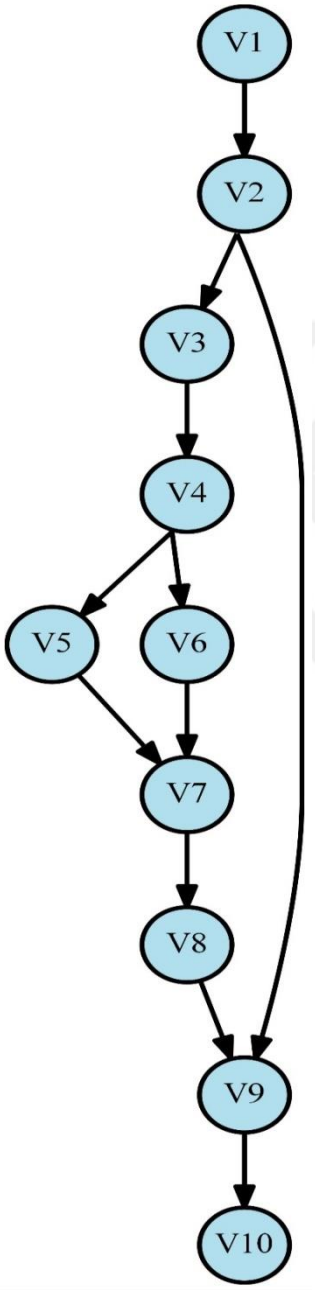
Yukarıdaki bilgiler ışığında donanım-yazılım bölüştürmesi problemi aşağıdaki gibi bir optimizasyon problemi olarak modellenebilir:

$$\boldsymbol{L(G) \leq T_{con}}\ kısıtı\ altında\ \boldsymbol{\beta}\ değerinin\ \boldsymbol{minimize}\ edilmesi$$

3.2. Örnek Bir Graf Üzerinde Bir Rastgele Çözümün $\boldsymbol{\beta}$ Değerinin Hesaplanması

Jemai *et al.* (2017) tarafından yürütülen çalışmada kullanılan 10 düğümlü bir graf Şekil 3.3 ve bu grafın her bir düğümünün donanım gecikmesi, yazılım gecikmesi ve donanım alanı değerlerini içeren tablo ise Çizelge 3.1 ile verilmiştir. Öte yandan Çizelge 3.2 bu graftaki düğümlerin birbirlerine olan bağımlılıklarını bir matris formatında listeleyen bir tablodur ve bu matris graf kullanılarak elde edilmiştir. Bu tabloda yer alan herhangi bir 1 değeri ilgili satırdaki düğümün ilgili sütundaki düğüme bağımlı olduğuna işaret etmektedir. Çizelge 3.2 dikkatle incelendiğinde 10 numaralı düğümün ($\boldsymbol{V_{10}}$) kendisi

hariç tüm düğümlere bağımlı olduğu, benzer şekilde 1 numaralı düğümün ($\boldsymbol{V_1}$) hiçbir düğüme bağımlı olmadığı anlaşılmaktadır.



Şekil 3.3. 10 düğümlü graf

Çizelge 3.1. 10 düğümlü grafın düğüm değerlerini içeren tablo

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
$H_L(V_i)$	4	3	3	4	4	2	3	2	3	4
$S_L(V_i)$	8	7	8	9	10	5	8	7	10	9
$A(V_i)$	4	11	5	7	9	6	13	10	3	12

Çizelge 3.2. 10 düğümlü grafın düğümlerine ait bağımlılık matrisi

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
V_1	0	0	0	0	0	0	0	0	0	0
V_2	1	0	0	0	0	0	0	0	0	0
V_3	1	1	0	0	0	0	0	0	0	0
V_4	1	1	1	0	0	0	0	0	0	0
V_5	1	1	1	1	0	0	0	0	0	0
V_6	1	1	1	1	0	0	0	0	0	0
V_7	1	1	1	1	1	1	0	0	0	0
V_8	1	1	1	1	1	1	1	0	0	0
V_9	1	1	1	1	1	1	1	1	0	0
V_{10}	1	1	1	1	1	1	1	1	1	0

Şimdi rastgele bir çözüm vektörünün $\boldsymbol{S} = \{\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{0},\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{0}\}$ şeklinde üretildiğini kabul edelim. ($\boldsymbol{V_1}$, $\boldsymbol{V_4}$, $\boldsymbol{V_6}$ ve $\boldsymbol{V_9}$ düğümleri donanıma, benzer şekilde $\boldsymbol{V_2}$, $\boldsymbol{V_3}$, $\boldsymbol{V_5}$, $\boldsymbol{V_7}$, $\boldsymbol{V_8}$ ve $\boldsymbol{V_{10}}$ düğümleri de yazılıma atanmıştır.) Bu çözüme ait $\boldsymbol{\beta}$ değeri aşağıdaki adımlar sırası ile takip edilerek hesaplanabilir:

1) Grafın yazılım gecikmesinin hesaplanması

$\boldsymbol{S_L(G)}$=(1-1)x8+(1-0)x7+(1-0)x8+(1-1)x9+(1-0)x10+(1-1)x5+(1-0)x8+ (1-0)x7+(1-1)x10+(1-0)x9 = $\boldsymbol{49}$

2) Grafın donanım gecikmesinin hesaplanması

(1x4)+{(1x0)x4+(0x0)x3+(0x0)x3+(1x0)x4+(0x0)x4+(1x0)x2+(0x0)x3+(0x0)x2+(1x0)x3+(0x0)x4} (**V_1 için**)	**4**
(0*3)+{(1x1)x4+(0x0)x3+(0x0)x3+(1x0)x4+(0x0)x4+(1x0)x2+(0x0)x3+(0x0)x2+(1x0)x3+(0x0)x4} (**V_2 için**)	**4**
(0x3)+{(1x1)x4+(0x1)x3+(0x0)x3+(1x0)x4+(0x0)x4+(1x0)x2+(0x0)x3+(0x0)x2+(1x0)x3+(0x0)x4} (**V_3 için**)	**4**
(1x4)+{(1x1)x4+(0x1)x3+(0x1)x3+(1x0)x4+(0x0)x4+(1x0)x2+(0x0)x3+(0x0)x2+(1x0)x3+(0x0)x4} (**V_4 için**)	**8**
(0x4)+{(1x1)x4+(0x1)x3+(0x1)x3+(1x1)x4+(0x0)x4+(1x0)x2+(0x0)x3+(0x0)x2+(1x0)x3+(0x0)x4} (**V_5 için**)	**8**
(1x2)+{(1x1)x4+(0x1)x3+(0x1)x3+(1x1)x4+(0x0)x4+(1x0)x2+(0x0)x3+(0x0)x2+(1x0)x3+(0x0)x4} (**V_6 için**)	**10**
(0x3)+{(1x1)x4+(0x1)x3+(0x1)x3+(1x1)x4+(0x1)x4+(1x1)x2+(0x0)x3+(0x0)x2+(1x0)x3+(0x0)x4} (**V_7 için**)	**10**
(0x2)+{(1x1)x4+(0x1)x3+(0x1)x3+(1x1)x4+(0x1)x4+(1x1)x2+(0x1)x3+(0x0)x2+(1x0)x3+(0x0)x4} (**V_8 için**)	**10**
(1x3)+{(1x1)x4+(0x1)x3+(0x1)x3+(1x1)x4+(0x1)x4+(1x1)x2+(0x1)x3+(0x1)x2+(1x0)x3+(0x0)x4} (**V_9 için**)	**13**
(0x4)+{(1x1)x4+(0x1)x3+(0x1)x3+(1x1)x4+(0x1)x4+(1x1)x2+(0x1)x3+(0x1)x2+(1x1)x3+(0x0)x4} (**V_{10} için**)	**13**
Maksimum	**13**

$$\boldsymbol{H_L(G) = 13}$$

3) Grafın gecikmesinin hesaplanması

$$\boldsymbol{L(G) = S_L(G) + H_L(G)}$$

$$\boldsymbol{L(G) = 49+13 = 62}$$

4) Çözümün donanım alanının büyüklüğünün hesaplanması

$$\boldsymbol{A_L =} (1x4)+(0x11)+(0x5)+(1x7)+(0x9)+(1x6)+(0x13)+(0x10)+(1x3)+(0x12)$$

$$\boldsymbol{A_L = 20}$$

5) Tüm düğümlerin donanım alanlarının toplamının hesaplanması

$$\boldsymbol{A_{max}} = 4+11+5+7+9+6+13+10+3+12=\boldsymbol{80}$$

6) Çözüme ait $\boldsymbol{\beta}$ değerinin hesaplanması

$$\boldsymbol{\beta} = \frac{L(G)}{A_{max} - A_L} = \frac{62}{80 - 20} = \mathbf{1,03}$$

3.3. β Değeri ve Önemi

Çizelge 3.3'te Bölüm 3.2'de kullanılan örnek için 2 farklı çözüm vektörü ve bu çözümlere ait $\boldsymbol{L(G)}$, $\boldsymbol{A_L}$ ve $\boldsymbol{\beta}$ değerleri verilmiştir. Görüldüğü üzere her iki çözümün gecikme değeri de zamansal kısıtı ihlal etmemektedir. Dolayısıyla her iki çözüm de olası (feasible) bir çözümdür. Öyleyse bu iki çözümden hangisi daha kabul edilebilir bir çözümdür? Böyle bir durumda $\boldsymbol{\beta}$ değerleri dikkate alınmakta ve $\boldsymbol{\beta}$ değeri küçük olan çözüm diğerine tercih edilmektedir.

Çizelge 3.3. Bölüm 3.2'de kullanılan örnek için iki farklı çözüme ait değerler tablosu

Çözüm Vektörü	$L(G)$	A_L	β	**Zamansal Kısıt (T_{con})**
1010000010	65	12	0,955882	70
1000000010	70	7	0,958904	

3.4. Kaba Kuvvet Yaklaşımı ile Optimum Sonucun Bulunması

Donanım-yazılım bölüştürmesi probleminin çözüm yöntemlerinden biri olan kaba kuvvet yaklaşımı, muhtemel bütün kombinasyonlar test edilerek optimum çözümün bulunmasıdır. Bu yaklaşım genellikle problem boyutunun küçük olduğu durumlarda tercih edilmektedir. Örneğin 10 düğüm içeren bir grafın muhtemel bütün çözüm vektörlerinin sayısı 1024'tür ve arama uzayı küçüktür. Öte yandan 32 düğümlü bir graf için 4 294 967 296 farklı çözüm vardır ve bu çözümlerin her birinin test edilmesi ile optimumun bulunması pratikte uygulanabilir ve kabul edilebilir değildir. Bu tez kapsamında optimum çözümler kaba kuvvet yaklaşımı ile bulunmuştur ve sezgisel algoritmaların kalitesini değerlendirmek amacıyla kullanılmıştır.

3.5. Literatürdeki (Jemai *et al.* 2017) Sezgisel Yöntem

Jemai *et al.* (2017) tarafından yürütülen araştırmada, tabu arama algoritmasına ve ikili arama ağacına dayalı bir donanım-yazılım bölüştürmesi algoritması önerilmektedir. Bu algoritma, bir zaman kısıtı altında, donanım alanını en aza indirgemeyi hedefleyen bir algoritmadır. Söz konusu algoritmanın sözde kodu aşağıda verilmiştir.

Literatürdeki (Jemai *et al.* 2017) Sezgisel Yönteme Ait Sözde Kod

Giriş: Graf, Düğümlerin Donanım Gecikmeleri, Düğümlerin Yazılım Gecikmeleri, Düğümlerin Donanım Alanları, Zamansal Kısıt (T_{con}), İterasyon Sayısı, FIFO Listesinin Boyutu

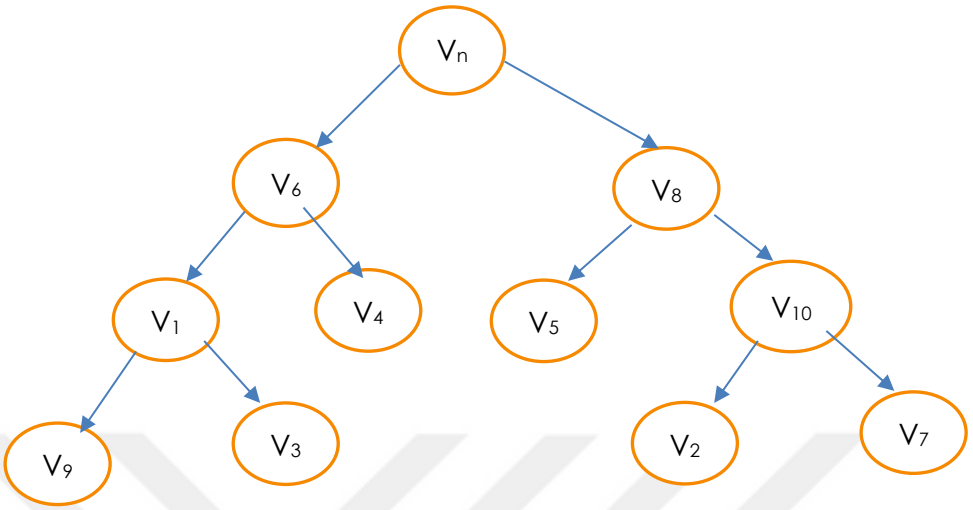
Çıkış: Bölüştürülmüş Graf, A_L, β

1: binarySearchTree ← İkiliAramaAğacınaDönüştür(Graf)
2: LST ←1
3: RST← 0
4: T_{ex} ← ÇözümVektörününÇalışmaZamanınıHesapla(S)
5: **if** $T_{ex} \leq T_{con}$
6: S_0 ← LST_Rastgele(LST)
7: S^*_0 ← RST
8: **else**
9: S_0 ← RST_Rastgele(RST)
10: S^*_0 ← LST
11: A_L ← AlanHesapla(S_0)
12: **for** i←1 **to** iterasyonSayisi **by** 1
13: S_1 ← KomşularArasıEnİyiSonucuBelirle(S_0)
14: FIFOList ← ListeyiGüncelle(S_1)
15: minAlan← FIFOListesiMinimumDeğeri(FIFOList)

Yukarıda sözde kodu verilen algoritmanın uygulama adımları, 10 düğümlü graf ve bu grafa ait tablo değerleri kullanılarak aşağıda detaylandırılmıştır. (T_{con} değeri olarak 70, iterasyon sayısı olarak 7 ve FIFO listesinin boyutu olarak 3 kullanılmıştır.)

Öncelikle graf ikili arama ağacına dönüştürülür. İkili arama ağacında kök, en büyük ve en küçük donanım alan değerlerinin ortalamasıdır.

Kök değeri = (A(v_7) + A(v_9))/2 = (13+3)/2 =8



İkili arama ağacının köke göre solunda yer alan düğümlerin donanıma, sağında yer alan düğümlerin yazılıma atandığı bir çözüm vektörü **S** aşağıdaki gibi oluşturulur.

S=(1, 0, 1, 1, 0, 1, 0, 0, 1, 0)

Bu çözüm vektörünün gecikme değeri olan T_{ex} yazılıma atanan düğümlerin yazılım gecikmesi değerleri toplamları ile donanıma atanan düğümlerin donanım gecikmesi değerleri toplamı olarak hesaplanır.

$$T_{ex}=H_L(V_1)+S_L(V_2)+H_L(V_3)+H_L(V_4)+S_L(V_5)+H_L(V_6)+S_L(V_7)+S_L(V_8)+H_L(V_9)+S_L(V_{10})$$

T_{ex} = 4+7+3+4+10+2+8+7+3+9 = 57

$T_{ex} \leq T_{con}$ $(57 \leq 70)$ şartı sağlandığından dolayı LST bireyleri arasından rastgele bir düğüm yazılıma atanarak S_0 çözümü elde edilir. RST bireyleri ise S_0^* çözümünü oluşturur. (Örneğin aşağıda rastgele LST düğümü olarak V_3 seçilmiştir ve bu düğüm yazılıma atanmıştır.)

S_0=(10111) S_0^*=(00000)

Bu yeni durumda S=(1001010010) şekline dönüşmüştür ve sonrasında L(G) değeri 62 ve A_L değeri de 20 olarak hesaplanmıştır.

Daha sonra S_0 vektörünün bütün komşuluklarının toplam gecikme değerleri ile A_L değerleri hesaplanmıştır. $L(G) \leq T_{con}$ şartını sağlayan ve donanım alanları toplamını minimum yapan komşu S_0 vektörüne atanmıştır. Bahsedilen komşuluklar aşağıdaki gibi elde edilmiştir ve burada N komşulukları temsil etmektedir.

S_0=(1 0 1 1 1) iken

N(0)=0 0 1 1 1 N(1)=1 1 1 1 1 N(2)=1 0 0 1 1 N(3)=1 0 1 0 1 N(4)=1 0 1 1 0

Çizelge 3.4. S_0 Çözümüne ait komşulukların A_L ve L(G) değerleri

Komşuluk	N(0)	N(1)	N(2)	N(3)	N(4)
A_L	16	25	13	14	17
L(G)	66	57	67	65	69

S_0 çözümüne ait komşulukların gecikme ve A_L değerleri hesaplandığında aranan komşunun ve dolayısıyla yeni S_0 çözümünün 10011 (N(2)) olduğu anlaşılır. Bu esnada FIFO listesine de elde edilen en iyi sonucun A_L değeri atanır. Bu işlem sonucunda iterasyon sayısı 1 azalmış olur.

FIFO listesi güncellenirken, liste dolana kadar algoritmanın bulduğu minimum değerler listeye eklenir. Liste algoritmaya girdi olarak verilen boyuta ulaştığında listedeki değerler ile bulunan A_L değeri karşılaştırılır. Bulunan A_L değeri listedeki değerlerden küçükse yeni değer listeye eklenir. A_L değeri listede var ise yeni S_0 çözümü belirlenir.

Elde edilen yeni çözümle, iterasyon sayısı 7 olana kadar aynı işlemler tekrar edilir. Tüm iterasyonlar tamamlandığında aşağıda verilen tablo tamamlanmış olur ve en iyi sonuç olarak S=(1000000010) vektörü kullanıcıya algoritmanın çıktısı olarak sunulur.

Çizelge 3.5. Literatürde kullanılan yöntemle elde edilen örnek sonuç tablosu

10 Düğümlü Graf	**S0**	**A_L(S0)**	**L(G)**	**S1**	**A_L(S1)**	**L(G)**	**FIFO Listesi**
İterasyon 1	10111	20	62	10011	13	67	{13}
İterasyon 2	10011	13	67	10001	7	70	{13,7}
İterasyon 3	10001	7	70	11001	12	65	{13,7,12}
İterasyon 4	10111	20	62	10011	13	67	{13,12,7}
İterasyon 5	11110	22	64	11010	15	69	{13,12,7}
İterasyon 6	11010	15	69	11011	18	62	{13,12,7}
İterasyon 7	11100	16	67	11101	19	60	{13,12,7}

3.6. Bu Tez Kapsamında Önerilen Sezgisel Algoritma

Bu tez kapsamında donanım-yazılım bölüştürmesi problemini çözmek için yeni bir sezgisel algoritma geliştirilmiştir.

Donanım-yazılım bölüştürme problemi için geliştirilen bu yeni sezgisel algoritmada ilk kullanılan ölçüt donanım alanı değerleridir. Amaç alanın minimum olmasını sağlamak olduğu için, geliştirilen yöntemin ilk çözüm değerinin tamamı yazılıma atanmıştır. Dolayısıyla önerilen sezgisel yöntemde kullanılmak üzere tanımlanmış ilk çözüm aşağıdaki gibidir:

S={ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

Önerilen Sezgisel Algoritmaya Ait Sözde Kod

Giriş: Graf, Düğümlerin Donanım Gecikmeleri, Düğümlerin Yazılım Gecikmeleri, Düğümlerin Donanım Alanları, Zamansal Kısıt (T_{con})

Çıkış: Bölüştürülmüş Graf, A_L, β

1: ilkCozum ← İlkÇözümüSadeceYazılımdanOluştur()

2: ilkCozumMaliyeti ← MaliyetHesapla(ilkCozum)

3: siraliDonanimAlani[] ← DüğümAlanlarınıİndisliArtanSırala(Düğüm Alanları)

4: **for** i ← 0 **to** grafBoyutu **by** 1

5: yeniCozum[siraliDonanimAlani [i]] ← 1

6: yeniCozumMaliyeti ← MaliyetHesapla(yeniCozum)

7: enIyiCozum ← MaliyetKarsilastirEnİyiÇözümüGüncelle()

8: **End for**

9: yeniCozum← enIyiCozum

10: bağımlılıkMatrisi[] ← DüğümBağımlılıklarınıİndisliArtanSırala(Düğüm Bağımlılıkları)

11: **for** k←0 **to** grafBoyutu **by** 1

12: yeniCozum = enIyiCozum;

13: **if**(yeniCozum[bağımlılıkMatrisi[k]]==0)

14: yeniCozum[bağımlılıkMatrisi[k]]=1

15: yeniCozumMaliyeti ← MaliyetHesapla(yeniCozum)

16: if (yeniCozumMaliyet < enIyiCozumMaliyet)

17: enIyiCozum = yeniCozum

18: enIyiCozumMaliyet = yeniCozumMaliyet;

19: eniCozum[bağımlılıkMatris[k]] ← 0

20: **End for**

Yukarıdaki sözde kodda, düğümlerin donanım alanı değerleri indisleri korunarak ve küçükten büyüğe doğru sıralanarak bir diziye atanmıştır. İlk çözüm olarak kullanılan,

tamamı yazılımdan oluşan çözüm vektörünün değerleri, alan indisleri değerlerine göre sırasıyla yazılımdan donanıma çevrilerek maliyet hesabı yapılmıştır.

Hesaplanan maliyet değeri bir önceki maliyet değeri ile karşılaştırılmıştır. $L(G) \leq T_{con}$ şartı göz önünde bulundurularak β değerleri açısından bir karşılaştırma yapılmıştır. Karşılaştırma sonucu hesaplanan maliyet bir önceki maliyetten küçük ise minimum sonuç güncellenmiştir. Döngü devam ederken bir sonraki indis değeri yazılımdan donanıma çevrilerek işlem tekrarlanmıştır.

Algoritmanın çalışma adımları aşağıda sırayla verilmiştir.

Donanım alanlarının küçükten büyüğe sıralanmış değerleri Çizelge 3.6 ile verilmiştir:

Çizelge 3.6. Donanım alanına göre artan sıralanmış indis tablosu

Düğüm	V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
$A(V_i)$	3	4	5	6	7	9	10	11	12	13

1. Adım: İlk çözümde tüm düğümler yazılıma atanmıştır.

Çizelge 3.7. İlk çözümün maliyet değeri

İlk çözüm	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
$\beta = 1{,}01250$	0	0	0	0	0	0	0	0	0	0

2. Adım: Alan değeri en küçük olan V_9 düğümü donanıma atanmıştır.

Çizelge 3.8. V9 düğümünün seçilmesi

1									
V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
3	4	5	6	7	9	10	11	12	13

Çizelge 3.9. 2. çözümün maliyet değeri

									1	
$\beta = 0{,}96103$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	0	0	0	0	0	0	0	0	1	0

3. Adım: Alan değeri ikinci küçük olan V_1 düğümü donanıma atanmıştır.

Çizelge 3.10. V2 düğümünün seçilmesi

1	2								
V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
3	4	5	6	7	9	10	11	12	13

Çizelge 3.11. 3. çözümün maliyet değeri

	2								1	
$\beta = 0{,}9589$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	0	0	0	0	0	0	1	0

4. Adım: Alan değeri üçüncü küçük olan V_3 düğümü donanıma atanmıştır.

Çizelge 3.12. V3 düğümünün seçilmesi

1	2	3							
V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
3	4	5	6	7	9	10	11	12	13

Çizelge 3.13. 4. çözümün maliyet değeri

	2		3						1	
$\beta = 0{,}9558$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	0	0	0	0	0	1	0

4. adıma kadar tüm maliyetler bir önceki maliyete göre daha küçüktür. Bu aşamalar gerçekleşirken $L(G) \leq T_{con}$ şartı dikkate alınarak β değerleri arasında karşılaştırma yapılmaktadır.

5. Adım: Alan değeri dördüncü küçük olan V_6 düğümü donanıma atanmıştır.

Çizelge 3.14. V6 düğümünün seçilmesi

1	2	3	4						
V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
3	4	5	6	7	9	10	11	12	13

Çizelge 3.15. 5. çözümün maliyet değeri

	2		3			4			1	
$\beta = 1$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	0	0	1	0	0	1	0

6. Adım: Alan değeri beşinci küçük olan V_4 düğümü donanıma atanmıştır.

Çizelge 3.16. V4 düğümünün seçilmesi

1	2	3	4	5					
V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
3	4	5	6	7	9	10	11	12	13

Çizelge 3.17. 6. çözümün maliyet değeri

	2		3	5		4			1	
$\beta = 1{,}03636$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	1	0	1	0	0	1	0

7. Adım: Alan değeri altıncı küçük olan V_5 düğümü donanıma atanmıştır.

Çizelge 3.18. V5 düğümünün seçilmesi

1	2	3	4	5	6				
V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
3	4	5	6	7	9	10	11	12	13

Çizelge 3.19. 7. çözümün maliyet değeri

	2		3	5	6	4			1	
$\beta = 1{,}10869$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	1	1	1	0	0	1	0

8. Adım: Alan değeri yedinci küçük olan V_8 düğümü donanıma atanmıştır.

Çizelge 3.20. V8 düğümünün seçilmesi

1	2	3	4	5	6	7			
V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
3	4	5	6	7	9	10	11	12	13

Çizelge 3.21. 8. çözümün maliyet değeri

	2		3	5	6	4		7	1	
$\beta = 1{,}27777$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	1	1	1	0	1	1	0

9. Adım: Alan değeri sekizinci küçük olan V2 düğümü donanıma atanmıştır.

Çizelge 3.22. V2 düğümünün seçilmesi

1	2	3	4	5	6	7	8		
V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
3	4	5	6	7	9	10	11	12	13

Çizelge 3.23. 9. çözümün maliyet değeri

	2	8	3	5	6	4		7	1	
$\beta = 1{,}67999$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	1	1	1	1	1	0	1	1	0

10. Adım: Alan değeri dokuzuncu küçük olan V_{10} düğümü donanıma atanmıştır.

Çizelge 3.24. V10 düğümünün seçilmesi

1	2	3	4	5	6	7	8	9	
V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
3	4	5	6	7	9	10	11	12	13

Çizelge 3.25. 10. çözümün maliyet değeri

	2	8	3	5	6	4		7	1	9
$\beta = 2{,}53846$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	1	1	1	1	1	0	1	1	1

11. Adım: Alan değeri en büyük olan V_7 düğümü donanıma atanmıştır.

Çizelge 3.26. V7 düğümünün seçilmesi

1	2	3	4	5	6	7	8	9	10
V_9	V_1	V_3	V_6	V_4	V_5	V_8	V_2	V_{10}	V_7
3	4	5	6	7	9	10	11	12	13

Çizelge 3.27. 11. çözümün maliyet değeri

	2	8	3	5	6	4	10	7	1	9
$\beta = sonsuz$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	1	1	1	1	1	1	1	1	1

Yürütülen işlem adımları sonucunda en iyi çözüm {1, 0, 1, 0, 0, 0, 0, 0, 1, 0} vektörüne aittir.

İlk adımda yapılan işlemlerin sonucunda aşağıdaki çizelgede verilen değerler en iyi çözümü göstermektedir.

Çizelge 3.28. İlk döngü sonucunda elde edilen en iyi çözüm

	2		3						1	
$\beta = 0{,}9558$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	0	0	0	0	0	1	0

Bir sonraki adımın gerçekleşme aşamaları ise şu şekildedir.

Bu adımda, bir önceki adımda en iyi çözüm olarak bulunan çözüm içerisinde yazılıma atanmış düğümler sırasıyla (bağımlılık sayısı en az düğümden en fazla düğüme kadar) donanıma atanmakta ve en iyi çözümü iyileştirip iyileştirmedikleri kontrol edilmektedir.

Çizelge 3.29. Bağımlılık değerine göre artan sıralanmış indis tablosu

Düğüm	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
$A(v_i)$	0	1	2	3	4	4	6	7	8	9

Çizelge 3.30. V2 düğümünün seçilmesi ve yazılımdan donanıma aktarılması

	2	1.	3						1	
$\beta = 1{,}07017$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	1	1	0	0	0	0	0	1	0

Çizelge 3.31. V4 düğümünün seçilmesi ve yazılımdan donanıma aktarılması

	2		3	2.					1	
$\beta = 0{,}98360$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	1	0	0	0	0	1	0

Çizelge 3.32. V5 düğümünün seçilmesi ve yazılımdan donanıma aktarılması

	2		3		3.				1	
$\beta = 1$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	0	1	0	0	0	1	0

Bu aşamaya kadar yapılan işlemler ilk döngü sonucu elde edilen çözümü değiştirmemiştir.

Çizelge 3.33. V6 düğümünün seçilmesi ve yazılımdan donanıma aktarılması

	2		3			4.			1	
$\beta = 1$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	0	0	1	0	0	1	0

Çizelge 3.34. V7 düğümünün seçilmesi ve yazılımdan donanıma aktarılması

	2		3				5.		1	
$\beta = 1{,}090909$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	0	0	0	1	0	1	0

Çizelge 3.35. V8 düğümünün seçilmesi ve yazılımdan donanıma aktarılması

	2		3					6.	1	
$\beta = 1{,}034482$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	0	0	0	0	1	1	0

Çizelge 3.36. V10 düğümünün seçilmesi ve yazılımdan donanıma aktarılması

	2		3						1	7.
$\beta = 1$	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
	1	0	1	0	0	0	0	0	1	1

Yukarıdaki adımların uygulanması sonucunda $L(G) \leq T_{con}$ kısıtını sağlayan en küçük β değerine ait çözüm, yani en iyi çözüm S=(1, 0, 1, 0, 0, 0, 0, 0, 1, 0) vektörüdür. Bu çözümün β değeri 0,9558'dir ve bu değer aynı zamanda kaba kuvvet algoritması ile bulunan optimum değere eşittir.

4. ARAŞTIRMA BULGULARI ve TARTIŞMA

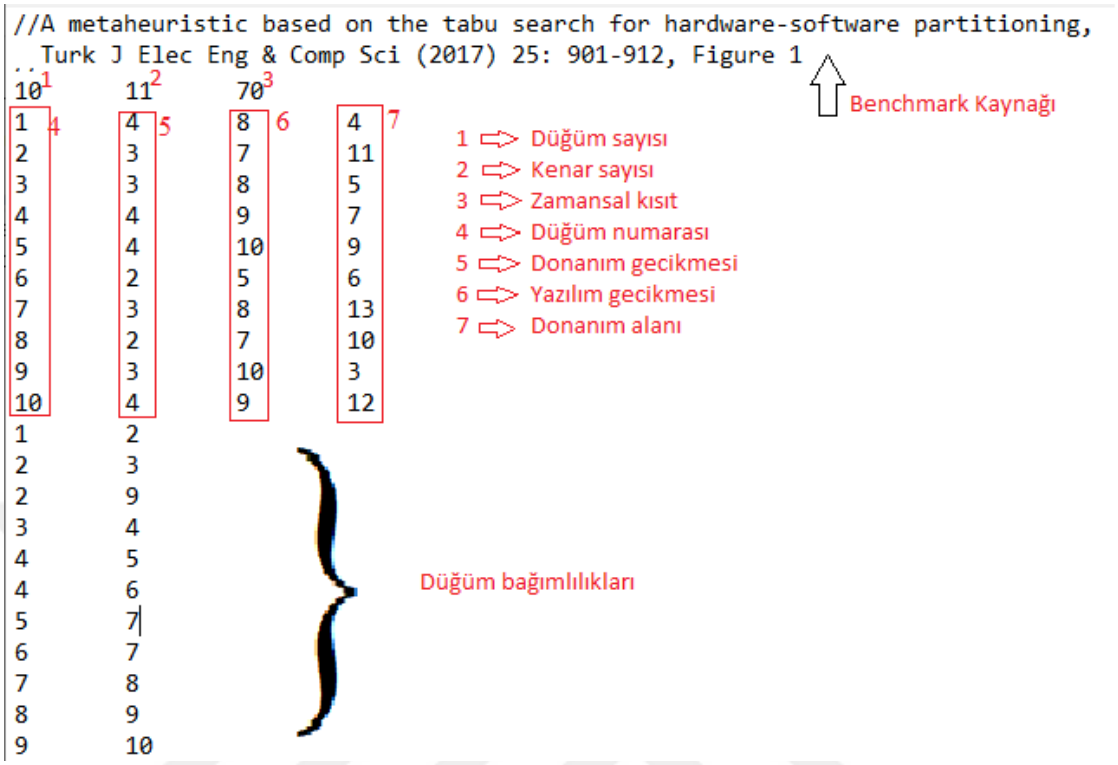
Bu tez kapsamında gerçeklenen iki algoritma da C++ programlama dili kullanılarak kodlanmıştır. Deneysel sonuçlar ise 2.50 GHz CPU hızına sahip Intel(R) Core (TM) i5-4200M işlemcili ve 8 GB ana bellek kapasitesine bir bilgisayar kullanılarak elde edilmiştir.

Donanım-yazılım bölüştürmesi problemi ile ilgili literatür taraması yapıldığında bu alanda tasarlanan algoritmaların efektifliğini ölçebilmek ve karşılaştırma yapabilmek için kullanılan denektaşlarının yetersiz olduğu gözlemlenmiştir. Bundan dolayı sıklıkla kullanılan graflardan yola çıkılarak 5 denektaşı içeren yeni bir denektaşı seti bildiğimiz kadarıyla bu problem için ilk defa oluşturulmuştur.

4.1. Denektaşı Seti

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

Jemai *et al.* (2017) tarafından yürütülen çalışmada önerilen algoritmanın veriminin tek bir denektaşı ile değerlendirilmesinin büyük bir eksiklik olduğu gerçeğinden hareketle bu tez kapsamında 5 denektaşı içeren yeni bir denektaşı seti literatüre kazandırılmıştır. Bir denektaşı dosyası içerisinde modellenen ilgili grafa ait bilgiler, yani denektaşı dosyasının formatı aşağıdaki gibidir:



Şekil 4.1. Denektaşı dosyasının formatı

Benchmark kaynağı, grafın alındığı yayın bilgilerinden oluşturulmuştur.

Denektaşı oluştururken donanım gecikmesi, yazılım gecikmesi ve donanım alan verileri Jemai *et al.* (2017) çalışmasında kullanılan değerler baz alınarak rastgele belirlenmiştir. Değer aralıkları şu şekilde verilmiştir:

$\boldsymbol{H_L(V_i)}$: 3-8 arasında rastgele değerler

$\boldsymbol{S_L(V_i)}$: $H_L(vi)x2$, $H_L(vi)x2+1$, $H_L(vi)x2+2$, $H_L(vi)x3+1$ veya $H_L(vi)x3+2$ rastgele değerinden birisi

$\boldsymbol{A(V_i)}$: 5-15 arası rastgele değerler

4.2. Karşılaştırma Tablosu

Bu tez kapsamında geliştirilen algoritma 10'lu grafın optimum çözümünü 20 adımda bulabilmiştir. Jemai *et al.* (2017) çalışmasında kullanılan algoritma ise 35 adımda minimuma yakın sonuca ulaşmıştır, minimumu bulamamıştır.

Tüm denektaşları ile gerçekleştirilen deneylerin sonuçları Çizelge 36'da özetlenmiştir.

Çizelge 4.1'de donanım-yazılım bölüştürmesi probleminin çözümü için 5 farklı denektaşına uygulanan 3 ayrı yöntem ile elde edilmiş sonuç değerleri gösterilmiştir. Denektaşı 1 ve denektaşı 5'te, baz alınan zamansal kısıt değerlerine göre, tez kapsamında geliştirilen sezgisel algoritma ile optimum sonuca ulaşılmıştır. Denektaşı 2, denektaşı 3 ve denektaşı 4'te ise geliştirilen sezgisel yöntem optimum çözüme yakın sonuçlar vermiştir. Bu sonuçlar, önerilen algoritmanın literatürdeki sezgisel (Jemai *et al.* 2017) algoritmaya göre optimuma daha yakın değerler ürettiğini kanıtlamıştır.

Çizelge 4.1. Sonuç tablosu

Denektaşı No	**Düğüm Sayısı**	**Zamansal Kısıt**	**Literatürdeki Sezgisel (Jemai *et al.* 2017)**	**Geliştirilen Sezgisel Yöntem**	**Optimum Çözüm**
1	10	70	β = 0,958904 Yazılım Gecikmesi =63 Donanım Gecikmesi =7 Toplam Gecikme = 70 1000000010	β = 0,955882 Yazılım Gecikmesi = 55 Donanım Gecikmesi=10 Toplam Gecikme =65 1010000010	β =0,955882 Yazılım Gecikmesi = 55 Donanım Gecikmesi=10 Toplam Gecikme =65 1010000010

5	4	3	2
32	26	23	15
475	300	170	115
$\beta = 1{,}579125$ Yazılım Gecikmesi =415 Donanım Gecikmesi =18 Toplam Gecikme =469 00000000000000000000000000011011	$\beta = 1{,}492462$ Yazılım Gecikmesi =262 Donanım Gecikmesi =42 Toplam Gecikme =304 00000000010101100001011101	$\beta = 1{,}16153$ Yazılım Gecikmesi =121 Donanım Gecikmesi =30 Toplam Gecikme = 151 011101100110111100 0110	$\beta = 2{,}837838$ Yazılım Gecikmesi =37 Donanım Gecikmesi =68 Toplam Gecikme = 105 111111111100101
$\beta = 1{,}47751$ Yazılım Gecikmesi =389 Donanım Gecikmesi =38 Toplam Gecikme =427 10000000000001000000000110000010	$\beta = 1{,}41294$ Yazılım Gecikmesi =222 Donanım Gecikmesi =62 Toplam Gecikme = 284 00011000010001100001010101	$\beta = 1{,}13559$ Yazılım Gecikmesi =104 Donanım Gecikmesi =30 Toplam Gecikme = 134 01110111011011110000110	$\beta = 1{,}98246$ Yazılım Gecikmesi =59 Donanım Gecikmesi =54 Toplam Gecikme = 113 110011101111101
$\beta = 1{,}47751$ Yazılım Gecikmesi =389 Donanım Gecikmesi =38 Toplam Gecikme =427 10000000000001000000000110000010	$\beta = 1{,}33603$ Yazılım Gecikmesi =310 Donanım Gecikmesi =20 Toplam Gecikme = 330 00010000000001000000000001	$\beta = 1{,}09559$ Yazılım Gecikmesi =119 Donanım Gecikmesi =30 Toplam Gecikme = 149 01100111011010110000011	$\beta = 1{,}24832$ Yazılım Gecikmesi =180 Donanım Gecikmesi =6 Toplam Gecikme =186 010000000000000

5. SONUÇ

Bu tez kapsamında öncelikle donanım-yazılım bölüştürmesi problemi, sezgisel ve metasezgisel yaklaşımlar hakkında bilgiler verilmiştir. Ayrıca detaylı bir literatür araştırması yapılarak donanım-yazılım bölüştürmesi problemi konusunda yayınlanmış farklı kaynaklardaki araştırmalar özetlenmiştir.

Literatür araştırması sonucunda donanım-yazılım bölüştürmesi algoritmalarının performans karşılaştırmalarında kullanılacak standart bir denektaşı kümesinin olmadığı fark edilmiştir. Bu motivasyonla literatürde farklı problemler için yaygın olarak kullanılan graflar özelleştirilerek 5 denektaşından oluşan bir set hayata geçirilmiştir.

Son olarak donanım-yazılım bölüştürmesi probleminin çözümüne yönelik yeni bir sezgisel algoritma geliştirilmiştir ve performansı Jemai *et al.* (2017) ile verilen sezgisel algoritma ile karşılaştırılmıştır. Önerilen algoritmanın diğer algoritmadan daha iyi sonuçlar ürettiği (optimumu bulduğu veya bulamadığı durumlarda optimuma daha yakın sonuçlar ürettiği) deneysel bulgular ile kanıtlanmıştır. Her bir denektaşına ait optimum sonuç bir kaba kuvvet algoritması ile bütün muhtemel bölüştürme kombinasyonları denenerek ayrıca hesaplandığı için algoritmaların efektifliği hakkında daha sağlıklı yorumlar yapılabilmesi mümkün olmuştur.

KAYNAKLAR

Abdelhalim, M. B., & Habib, S. D. (2007, December). Modeling communication cost and hardware alternatives in PSO based HW/SW partitioning. In 2007 Internatonal Conference on Microelectronics (pp. 111-114). IEEE.

Abdelhalim, M. B., & Habib, S. E. D. (2009). Particle swarm optimization for HW/SW partitioning. Particle Swarm Optimization, 49-76.

Anonymous, 2019. https://en.wikipedia.org/wiki/Swarm_intelligence (1.11.2019)

Ayadi, R., Ouni, B., & Mtibaa, A. (2012). A partitioning methodology that optimizes the communication cost for reconfigurable computing systems. International Journal of Automation and Computing, 9(3), 280-287.

Azari, E., & Koc, H. (2015, October). Improving performance through path-based hardware/software partitioning. In 2015 Fifth International conference on digital information processing and communications (ICDIPC) (pp. 54-59). IEEE.

Ben Haj Hassine, S., Jemai, M., & Ouni, B. (2017). Power and execution time optimization through hardware software partitioning algorithm for core based embedded system. Journal of Optimization, 2017.

Clouté, F., Contensou, J. N., Esteve, D., Pampagnin, P., Pons, P., & Favard, Y. (1999, March). Hardware/software co-design of an avionics communication protocol interface system: an industrial case study. In Proceedings of the Seventh International Workshop on Hardware/Software Codesign (CODES'99)(IEEE Cat. No. 99TH8450) (pp. 48-52). IEEE.

Dimassi, S., Jemai, M., Ouni, B., & Mtibaa, A. (2014). Hardware-Software Partitioning Algorithm Based On Binary Search Trees And Genetic Algorithm To Optimize Logic Area For SOPC. Journal of Theoretical & Applied Information Technology, 66(3).

Dimassi, S., Jemai, M., Ouni, B., & Mtibaa, A. (2015). Optimization Algorithms for Hardware/Software Partitioning.

Farahani, A. F., Kamal, M., & Salmani-Jelodar, M. (2006, September). Parallel-genetic-algorithm-based HW/SW partitioning. In International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06) (pp. 337-342). IEEE.

Farmahini-Farahani, A., Kamal, M., Fakhraie, S. M., & Safari, S. (2007, March). HW/SW partitioning using discrete particle swarm. In Proceedings of the 17th ACM Great Lakes symposium on VLSI (pp. 359-364). ACM.

Govil, N., 2016. Algorithms for High Performance Hardware Software Partitioning. Y.Lisans, International Institute of Information Technology, INDIA.

Hidalgo, J. I., & Lanchares, J. (1997, September). Functional partitioning for hardware-software codesign using genetic algorithms. In EUROMICRO 97. Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology (Cat. No. 97TB100167) (pp. 631-638). IEEE.

Jemai, M., & Ouni, B. (2015). Hardware software partitioning of control data flow graph on system on programmable chip. Microprocessors and Microsystems, 39(4-5), 259-270.

Jemai, M., Dimassi, S., Ouni, B., & Mtibaa, A. (2015). Combined partitioning hardware-software algorithms. International Journal of Computer Applications, 975, 8887.

Jemai, M., Dimassi, S., Ouni, B., & Mtibaa, A. (2017). A metaheuristic based on the tabu search for hardware-software partitioning. Turkish Journal of Electrical Engineering & Computer Sciences, 25(2), 901-912.

Koudil, M., Benatchba, K., Tarabet, A., & Sahraoui, E. B. (2007). Using artificial bees to solve partitioning and scheduling problems in codesign. Applied Mathematics and Computation, 186(2), 1710-1722.

Le Marrec, P., Valderrama, C. A., Hessel, F., Jerraya, A. A., Attia, M., & Cayrol, O. (1998, June). Hardware, software and mechanical cosimulation for automotive applications. In Proceedings. Ninth International Workshop on Rapid System Prototyping (Cat. No. 98TB100237) (pp. 202-206). IEEE.

Li, G., Feng, J., Wang, C., & Wang, J. (2014). Hardware/software partitioning algorithm based on the combination of genetic algorithm and tabu search. Engineering Review: Međunarodni časopis namijenjen publiciranju originalnih istraživanja s aspekta analize konstrukcija, materijala i novih tehnologija u području strojarstva, brodogradnje, temeljnih tehničkih znanosti, elektrotehnike, računarstva i građevinarstva, 34(2), 151-160.

Li, L., Song, Y., & Gao, M. (2010, December). A new genetic simulated annealing algorithm for hardware-software partitioning. In The 2nd International Conference on Information Science and Engineering (pp. 1-4). IEEE.

Lin, G., Zhu, W., & Ali, M. M. (2014). A tabu search-based memetic algorithm for hardware/software partitioning. Mathematical Problems in Engineering, 2014.

Lysecky, R., & Vahid, F. (2005, March). A study of the speedups and competitiveness of FPGA soft processor cores using dynamic hardware/software partitioning. In Design, Automation and Test in Europe (pp. 18-23). IEEE.

Nath, P. K., & Datta, D. (2014). Multi-objective hardware–software partitioning of embedded systems: A case study of JPEG encoder. Applied Soft Computing, 15, 30-41.

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

Saha, D., Mitra, R. S., & Basu, A. (1997, January). Hardware software partitioning using genetic algorithm. In Proceedings Tenth International Conference on VLSI Design (pp. 155-160). IEEE.

Stitt, G., Vahid, F., & Nematbakhsh, S. (2004). Energy savings and speedups from partitioning critical software loops to hardware in embedded systems. ACM Transactions on Embedded Computing Systems (TECS), 3(1), 218-232.

Wu, J. G., Srikanthan, T., & Zou, G. W. (2008). New model and algorithm for hardware/software partitioning. Journal of Computer Science and Technology, 23(4), 644-651.

Wu, J., Srikanthan, T., & Lei, T. (2010). Efficient heuristic algorithms for path-based hardware/software partitioning. Mathematical and Computer Modelling, 51(7-8), 974-984.

Wu, J., Wang, P., Lam, S. K., & Srikanthan, T. (2013). Efficient heuristic and tabu search for hardware/software partitioning. The Journal of Supercomputing, 66(1), 118-134.

Zhang, Y., Luo, W., Zhang, Z., Li, B., & Wang, X. (2008). A hardware/software partitioning algorithm based on artificial immune principles. Applied Soft Computing, 8(1), 383-391.

Zou, Y., Zhuang, Z., & Chen, H. (2004, June). HW-SW partitioning based on genetic algorithm. In Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753) (Vol. 1, pp. 628-633). IEEE.

ÖZGEÇMİŞ

Seda ERDEN DERTLİ 1991 yılında Tokat'ın Zile ilçesinde doğdu. İlk, orta ve lise eğitimini Zile'de tamamladı. 2009 yılında Kırıkkale Üniversitesi Bilgisayar Mühendisliği bölümünde öğrenime başlayıp 2013 yılında Bilgisayar Mühendisi olarak mezun oldu. 2015-2017 yılında Atatürk Üniversitesi Bilgisayar Bilimleri Araştırma ve Uygulama Merkezi'nde Bilgisayar Mühendisi ve Üniversite Bilgi Yönetim Sistemi'nde analist-yazılımcı olarak çalıştı. 2015 yılında Atatürk Üniversitesi Fen Bilimleri Enstitüsü'nde Yüksek Lisans öğrenimine başladı. 2017 yılında Atatürk Üniversite'ne Öğretim Görevlisi olarak atandı. 2017 yılından beri Bilimsel Araştırma Projeleri Koordinasyon Birimi'nde sistem yöneticisi olarak çalışmaktadır.