SUDOKU PROBLEMİNİN MATEMATİKSEL PROGRAMLAMA İLE ÇÖZÜMÜ VE BULMACA OLUŞTURMA

Tuğçe ATEŞ

T.C.
BURSA ULUDAĞ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

SUDOKU PROBLEMİNİN MATEMATİKSEL PROGRAMLAMA İLE ÇÖZÜMÜ VE BULMACA OLUŞTURMA

Tuğçe Ateş
0000-0002-5361-2735

Prof. Dr. Fatih ÇAVDUR
(Danışman)

YÜKSEK LİSANS
ENDÜSTRİ MÜHENDİSLİĞİ ANABİLİM DALI

BURSA – 2023

ÖZET

Yüksek Lisans Tezi

SUDOKU PROBLEMİNİN MATEMATİKSEL PROGRAMLAMA İLE ÇÖZÜMÜ VE BULMACA OLUŞTURMA

Tuğçe ATEŞ

Bursa Uludağ Üniversitesi
Fen Bilimleri Enstitüsü
Endüstri Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Fatih ÇAVDUR

Sudoku dünya çapında popülerlik kazanmış bir mantık bulmacasıdır. Bu tez çalışmasında, öncelikle matematiksel programlama kullanılarak sudoku probleminin çözümü incelenmiş ve sonrasında ise bir sudoku bulmacası oluşturma algoritması geliştirilerek oyun tasarımları gerçekleştirilmiştir. Çalışma kapsamında standart sudoku bulmacası dikkate alınmış olmakla birlikte, önerilen yaklaşımların geliştirilip genelleştirilerek diğer sudoku çeşitleri üzerinde de uygulanabileceği belirtilebilir.

Tez çalışması kapsamında öncelikle sudoku probleminin çözümü için bir matematiksel programlama formülasyonu dikkate alınmış, problem bir sağlanabilirlik veya uygunluk problemi olarak modellenerek, Visual Studio ortamında C# programlama dili ve Gurobi çözücü ile çözülmüştür. Buna ek olarak, yine Visual Studio ortamında bir oyun tasarımı gerçekleştirilmiştir. Tez çalışmasının ikinci kısmında, herhangi bir çözücü kullanılmadan kaba-kuvvet algoritmasıyla Sudoku problemini çözmenin yanı sıra, ayrıca bir sudoku bulmacası oluşturma algoritması geliştirilip Unity ortamında C# programlama dili ile kodlanıp bir oyun tasarımı gerçekleştirilmiştir. Çalışmanın her iki aşamasında da önerilen yaklaşımlar test edilmiş, örnek problemler ve bulmacalar oluşturulmuştur.

Anahtar Kelimeler: Bulmaca oyunları, sudoku, matematiksel programlama, 0-1 tamsayılı programlama
2023, x + 30 sayfa.

ABSTRACT

MSc Thesis

SOLVING THE SUDOKU PROBLEM WITH MATHEMATICAL PROGRAMMING AND PUZZLE CREATION

Tuğçe ATEŞ

Bursa Uludağ University
Graduate School of Natural and Applied Sciences
Department of Industrial Engineering

Supervisor: Prof. Dr. Fatih ÇAVDUR

Sudoku is a popular logic puzzle that has gained worldwide popularity. In this thesis, the solution of the sudoku problem was first studied using mathematical programming, and then a sudoku puzzle creation algorithm was developed and game designs were implemented. While the standard sudoku puzzle was considered in the scope of the study, it can be mentioned that the proposed approaches can be developed and generalized to be applied on other sudoku types.

In the scope of the thesis, a mathematical programming formulation was first considered for solving the sudoku problem, and the problem was modeled as a feasibility or suitability problem and solved using the C# programming language and the Gurobi solver in the Visual Studio environment. In addition, a game design was also implemented in the Visual Studio environment. In the second part of the thesis, in addition to solving the Sudoku problem with the brute-force algorithm without using any solver, a sudoku puzzle creation algorithm was developed and coded in the Unity environment using the C# programming language and a game design was implemented. Both stages of the study have tested the proposed approaches, and sample problems and puzzles have been created.

Key words: Puzzle games, sudoku, mathematical programming, 0-1 integer programming
2023, x + 30 pages.

ÖNSÖZ ve TEŞEKKÜR

Bu tez, sudoku problemini matematiksel programlama ile çözme ve sudoku bulmacası oluşturma algoritmasının geliştirilmesi çalışmalarımın araştırma ve bulgularını sunmaktadır. Bu konu benim için ilgi çekici çünkü sudoku dünya çapında popüler bir bulmaca oyunudur ve matematiksel programlama teknikleri kullanılarak çözülmesi daha etkin çözüm yöntemlerine ilişkin ipuçları sağlayabilir.

Umarım bu tez, sudoku, bulmaca oyunları ve matematiksel programlama ile ilgilenenler için yararlı olur.

Bu tezi yazma sürecinde beni yönlendiren ve destekleyen tez danışmanım Fatih Çavdur'a derin saygı ve minnettarlığımı ifade etmek istiyorum.

Yüksek lisans eğitimi sürecinde hem çalışma arkadaşım olan hem de değerli görüşleri ve desteği ile her zaman yanımda olan Aslı Sebatlı-Sağlam'a teşekkür ederim.

Yüksek lisans eğitimi için beni cesaretlendiren ve tüm süreç boyunca beni motive eden arkadaşım Barış Gündüz'e ve aileme ayrıca teşekkür ederim.

Yüksek lisans eğitimim TÜBİTAK 2210-A Yurt İçi Yüksek Lisans Burs Programı ile desteklenmiştir. Desteklerinden dolayı TÜBİTAK'a teşekkür ederim.

Tuğçe ATEŞ
02/01/2023

İÇİNDEKİLER

Sayfa

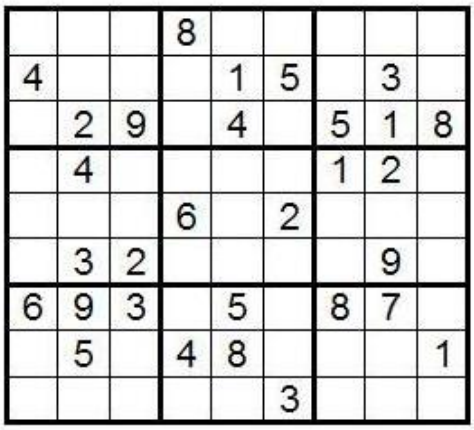
ŞEKİLLER DİZİNİ

1. GİRİŞ

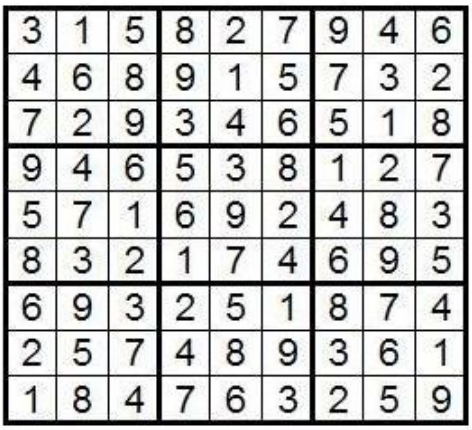
Sudoku dünya çapında popülerlik kazanmış bir oyundur. Bir sudoku oyununu çözmek için, mantık ve deneme-yanılma kombinasyonunu kullanmak gerekir. Arka planında ise daha fazla matematik yer alır: problemi çözmek için var olan çeşitli kısıtlar vardır. Oyun şu anki haliyle 1979 yılında Howard Garns tarafından geliştirilmiştir ve "Numbers in Place" adıyla Dell Magazines tarafından yayınlanmıştır. 1984 yılında Japon Maki Kaji, bulmaca şirketi Nikoli'nin dergisinde yayınlatmıştır. Maki Kaji oyuna "Tek sayılar" anlamına gelen modern Sudoku adını vermiştir. Oyun Japonya'da popüler olmuştur ve daha sonra Sudoku üretecek bir bilgisayar programı yazacak olan Yeni Zelandalı Wayne Gould tarafından keşfedilmiştir. Wayne Gould 2004'te Londra gazetesi The Times'ta bazı sudokular yayınlatmayı başarmıştır. Kısa bir süre sonra Sudoku İngiltere'de popüler olup, 2005 yılında ABD'de de popüler olmayı başarmıştır. Birçok gazete ve dergide düzenli olarak yayınlanmaya başlamıştır ve dünyanın hemen her yerindeki insanlar tarafından beğenilmiştir (https://en.wikipedia.org/wiki/Sudoku).

Sudokunun standart versiyonu 81 hücre içeren 9x9 kare matristen oluşmaktadır. Matris 9 tane 3x3'lük alt-matrislere bölünmüştür. Oyunun başlangıcında, 81 hücreden bazıları {1,2,3,4,5,6,7,8,9} kümesindeki rakamlarla doludur. Amaç; her satır, sütun ve alt matriste her rakamı sadece bir kez kullanacak şekilde tüm matrisi doldurmaktır.

Sudoku bulmacasını çözerken en temel strateji, önce her boş hücreye, verilen rakamlara göre kurallarla çelişmeyecek tüm olası seçenekleri yazmaktır. Bir hücrede yalnızca bir olası seçenek varsa bu rakam yazılıp aynı mantıkla devam edilir. Diğer bir strateji ise bir rakam ve bir satır, sütun veya alt matris seçmektir. Kuralları ihlal etmeden rakamın yerleştirilebileceği satır, sütun veya alt matristeki tüm hücreler belirlenerek, rakam o bölgede sadece bir hücreye yerleşebiliyorsa, o hücre doldurulur. Sonrasında seçilen rakam herhangi bir hücre için olasılık olmaktan çıkarılır. Bu iki strateji genellikle bir sudoku tablosunu tamamen doldurmak için yeterli değildir. İlerlemek için genellikle daha karmaşık analizlere ihtiyaç duyulur ve bazen de bir tahminde bulunulup devam edilir ve tahminin bir çelişki ile sonuçlanması durumunda geri adım atılır. Şekil 1.1'de bir Sudoku örneği ve onun çözümü verilmiştir (http://www.dailysudoku.com/sudoku/play.shtml).



(a)

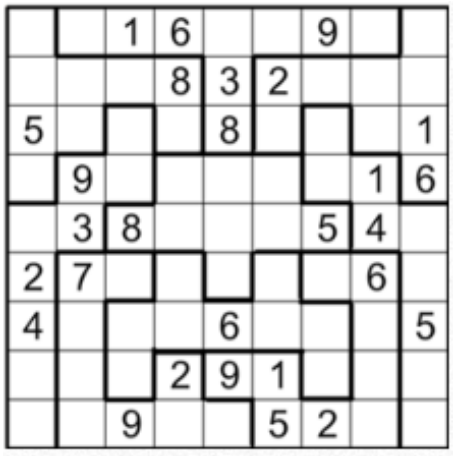


(b)

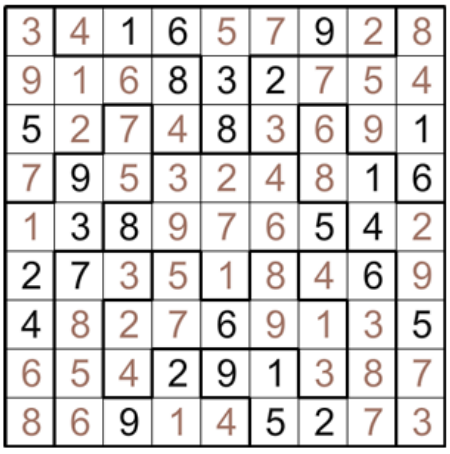
Şekil 1.1. Sudoku örneği (a) ve çözümü (b)

Sudoku standart versiyonu dışında, birçok farklı çeşitte oyuncuların karşısına çıkmaktadır. Bunlardan bazıları bölgesel sudoku, ardışık sudoku, ardışıksız sudoku, tek/çift sudoku ve çoklu sudokudur (https://en.wikipedia.org/wiki/Glossary_of_Sudoku).

Bölgesel sudokunun standart sudokudan farkı, sudoku bölgelerinin kare matris değil de birçok farklı geometrik şekilde olabilmesidir. Şekil 1.2'de bölgesel sudoku örneği ve onun çözümü verilmiştir (https://en.wikipedia.org/wiki/Glossary_of_Sudoku).



(a)

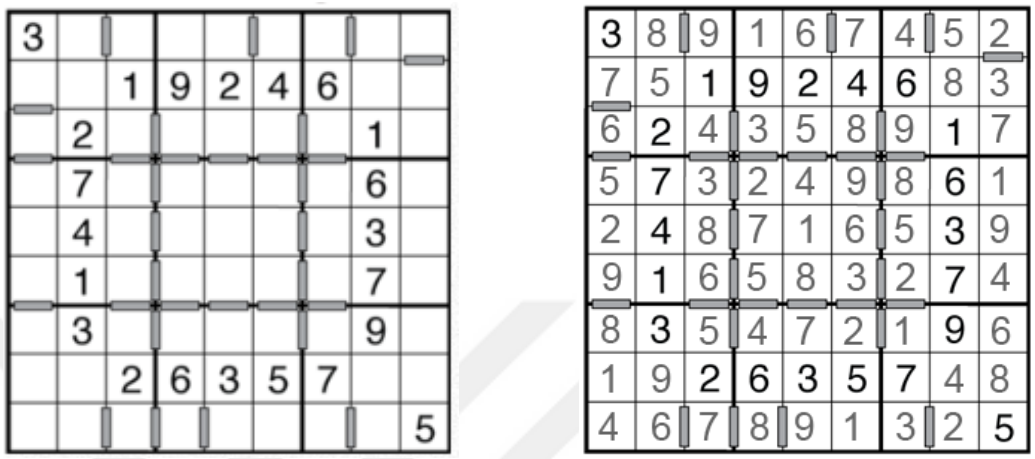


(b)

Şekil 1.2. Bölgesel sudoku örneği (a) ve çözümü (b)

Ardışık sudoku, standart sudoku kuralları haricinde bazı hücreler arasında yer alan kalın, kısa, düz çizgilerin her iki yanına yazılacak olan rakamların birbirinin ardışığı olması

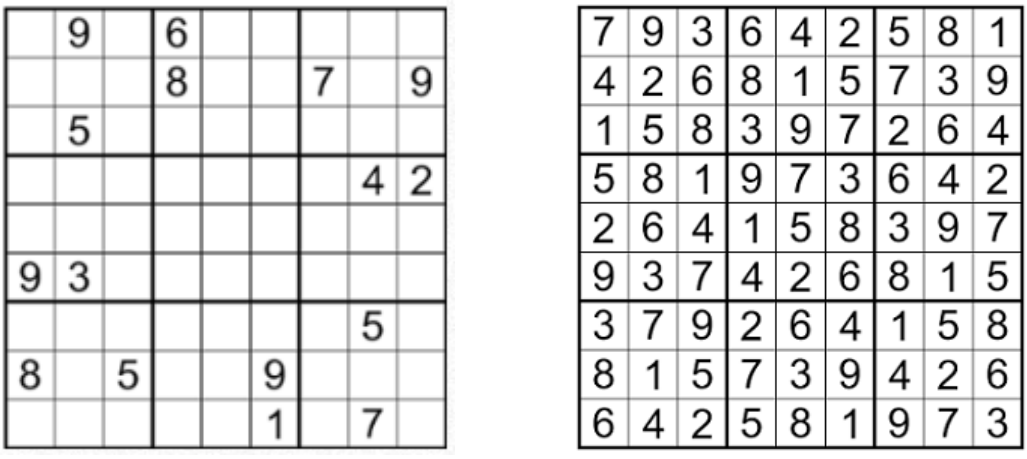
gereken bir sudoku çeşididir. Ayrıca hücreler arasında kısa çizgi bulunmayan yerlere yerleştirilecek rakamlar hiçbir zaman birbirinin ardışığı olmamalıdır. Şekil 1.3'te ardışık sudoku örneği ve onun çözümü verilmiştir (https://en.wikipedia.org/wiki/Glossary_of_Sudoku).



(a) (b)

Şekil 1.3. Ardışık sudoku örneği (a) ve çözümü (b)

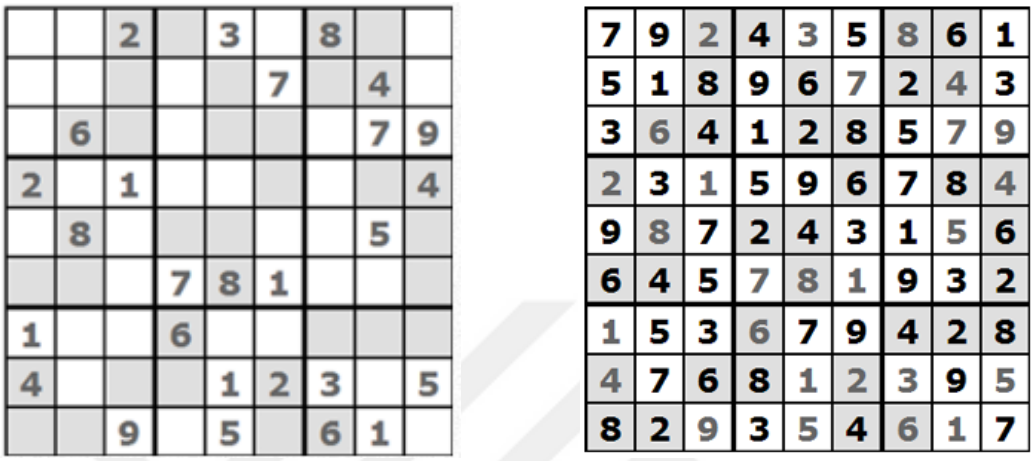
Ardışıksız sudoku, standart sudoku kuralları haricinde matristeki herhangi bir rakamın dört tarafına gelecek rakamların o rakamın ardışığı olmaması gereken bir sudoku çeşididir. Şekil 1.4'te ardışıksız sudoku örneği ve onun çözümü verilmiştir (https://en.wikipedia.org/wiki/Glossary_of_Sudoku).



(a) (b)

Şekil 1.4. Ardışıksız sudoku örneği (a) ve çözümü (b)

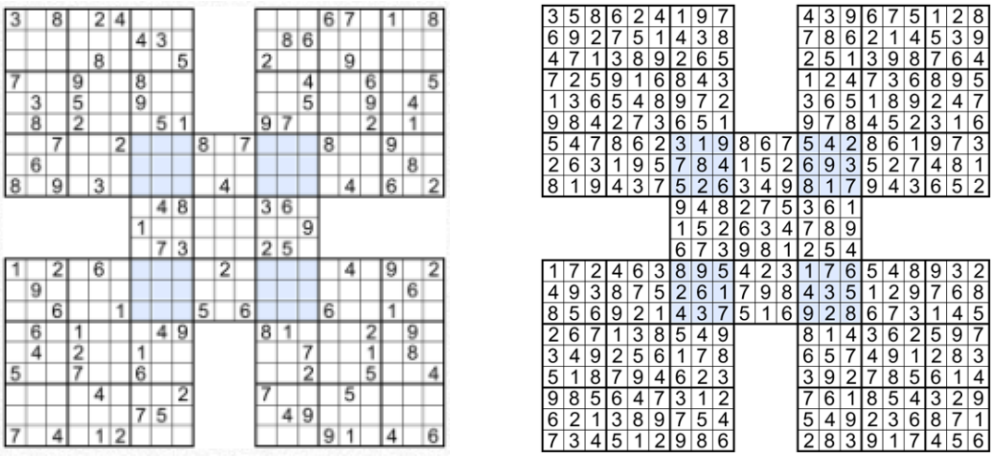
Tek/çift sudoku, standart sudoku kuralları haricinde matriste koyu renkle boyanmış hücrelere bazı sudoku bulmacalarında tek rakamların, bazı sudoku bulmacalarında ise çift rakamların gelmesi gereken bir sudoku çeşididir. Şekil 1.5'te tek/çift sudoku örneği ve onun çözümü verilmiştir (https://en.wikipedia.org/wiki/Glossary_of_Sudoku).



(a) (b)

Şekil 1.5. Tek/çift sudoku örneği (a) ve çözümü (b)

Çoklu sudoku, birden fazla sudokunun bir arada bulunduğu sudoku çeşididir. Çoklu sudoku içindeki herhangi bir sudoku standart sudoku kurallarına göre çözülür. Şekil 1.6'da çoklu sudoku örneği ve onun çözümü verilmiştir (https://en.wikipedia.org/wiki/Glossary_of_Sudoku).



(a) (b)

Şekil 1.6. Çoklu sudoku örneği (a) ve çözümü (b)

Bu tez çalışması kapsamında standart sudoku bulmacası dikkate alınmış olmakla birlikte, önerilen yaklaşımların geliştirilip genelleştirilerek diğer sudoku çeşitleri üzerinde de uygulanabileceği belirtilebilir. Tez çalışmasının amacı öncelikle standart sudoku probleminin matematiksel programlama kullanılarak formülize edilip çözülmesidir. Sonrasında ise ve ayrıca bir sudoku bulmacası oluşturma algoritması geliştirilerek oyun tasarımı gerçekleştirilmesidir.

Çalışmanın ilerleyen bölümlerinin organizasyonunda, "Kaynak Araştırması", bölümünde literatürde yer alan ilgili çalışmalar özetlenmektedir. İzleyen "Materyal ve Yöntem" bölümünde tez kapsamında önerilen matematiksel programlama modeli ve bulmaca oluşturma algoritması detayları sunulmakta, onu takip eden "Bulgular" bölümünde ise önerilen yaklaşımların uygulanmasıyla elde edilen sonuçlar sunulmaktadır. Çalışmanın son sayfalarında yer alan "Tartışma ve Sonuç" bölümünde tez çalışmasının genel bir değerlendirmesi sunulmaktadır.

2. KAYNAK ARAŞTIRMASI

Literatürdeki mevcut çalışmalar incelendiğinde, sudoku problemi için farklı çözüm yaklaşımlarının kullanıldığı görülmektedir. Bu bölümde bu çalışmalara kısaca değinilmeye çalışılmıştır.

Barlett ve diğerleri (2008), sudoku problemini iki açıdan incelemişlerdir. Bunlardan ilki sudoku probleminin matematiksel olarak nasıl çözülebileceği, ikincisi ise sudoku problemi oluşturmak için hangi matematiksel tekniklerin kullanılabileceği olmuştur. Sudoku problemini matematiksel olarak çözmek için 0-1 tamsayılı programlama modeli önermişlerdir ve bu yaklaşımı sudokunun farklı varyasyonlarını çözmek için genişletmişlerdir. Sudoku problemi oluşturmak için ise kaba-kuvvet (brute-force) ve eski bulmacalardan yeni bulmacalar üretme yaklaşımlarını kullanmışlardır. Coelco ve Laporte (2014), çalışmalarında Sudoku ve bazı yöneylem araştırması problemleri arasındaki ilişkiyi sunmuşlardır. Sudoku problemini, lineer-tamsayılı programlama modeli ve lineer olmayan-tamsayılı programlama modeli olmak üzere iki yaklaşımla modellemişlerdir. Sonrasında problemi çözmek için dört tane sayısal algoritma önererek, bu algoritmaların etkinliklerini ve verimliliklerini kıyaslamışlardır. Önerdikleri algoritmalardan geri-izleme (backtracking) algoritmasının daha etkin olduğunu görmüşlerdir.

Simonis (2005), çalışmasında sudokuyu kısıt sağlanabilirlik problemi olarak ele almıştır. Sonrasında benzer bir çalışma ile Lynce ve Ouaknine (2006), bireysel sudoku bulmacalarının nasıl birleştirici normal forma dönüştürülüp sağlanabilirlik teknikleri ile çözülebileceğini göstermişlerdir. Her iki çalışmada da çözümler, yazarların önerdikleri algoritmalar tarafından herhangi bir arama gerektirmeden hızlı bir şekilde elde edilmiştir. Chadwick ve diğerleri (2007), ise Sudoku problemlerinin üretilmesi üzerine bir algoritma geliştirmişlerdir. Algoritmayı geliştirirken üç ana hedef gözetmişlerdir. Bunlardan ilki çelişkilerin olmaması ve sadece tek bir çözümün geçerli olmasıdır. İkincisi bilinen dört farklı zorluk seviyesine sahip olmasıdır. Üçüncüsü ise bunun kabul edilebilir bir süre içerisinde yapılmasıdır. Chang ve diğerleri (2007), ise Sudoku probleminin üretilmesine odaklanan bir diğer çalışmaya imza atmışlardır. Sudokuyu bir arama problemi olarak tanımlamış ve bulmacanın üretilmesini bulmacanın zorluk seviyesiyle iç içe ele

almışlardır. Geliştirdikleri zorluk seviyesi ölçüm stratejisi için 800 bulmacayı analiz etmişler, kolay ve orta zorluk seviyesinde bulmacalar için bir adet, zor ve süper zorlar için bir adet olmak üzere iki adet bulmaca üreticisi geliştirmişlerdir.

Koch (2006), ise doktora tezinde, lineer ve karışık-tamsayılı programlamada yeni bir yazılımın uygulanmasını ele almıştır. Çalışmasında yeni bir cebirsel modelleme dilinin (ZIMPL) nasıl uygulanacağını, Sudoku problemini tamsayılı programlama ile modelleyerek adım adım anlatmıştır. Tang ve Zong (2016), çalışmalarında tek/çift sudoku probleminin çözümü için 0-1 tamsayılı programlama modeli geliştirmişlerdir. Sonrasında geliştirdikleri modeli MATLAB yazılımını kullanarak çözmüşlerdir. Çalışmaları tek/çift sudoku probleminin çözümü için yapılan ilk çalışma olması açısından önemlidir.

Oleinik ve Rogulin (2018), sudoku problemini kesikli kaynak atama probleminin bir türü olarak ele almışlardır ve lineer programlama ile modellemişlerdir. Sudoku problemindeki her bir matris elemanını üretim hattındaki benzersiz bir işçi grubu olarak düşünmüşlerdir. Problemi çözmek için benzer şekilde MATLAB yazılımını kullanmışlardır. Gabor ve Woeginger (2010), ise Sudokuyu bir tutarlılık kontrol problemi olarak ele almış ve Sudokunun nasıl çözülmemesi gerektiğini incelemişlerdir. Sıklıkla yapılan hataların rakam elimine etme stratejisindeki belirli yanlış hamlelerden kaynaklandığını ortaya koymuş, iki-bloklu eliminasyon stratejisi için daha gerçekçi bir varyant önermişlerdir.

Garcia ve Palomino (2007), çeşitli kuralları uygulayarak sudoku probleminin çözülebileceğini göstermişlerdir. Çözüm için kullandıkları teknik; tarama, işaretleme ve analiz olarak bilinen üç işlemdir. Yazarlar ana strateji olarak eliminasyon kullanmışlar, ve ayrıca eğer-ise (what-if) ve birkaç olasılık stratejisi kullanmışlardır. Bir diğer çalışmada, Mahdian ve diğerleri (2015), Latin Kare tamamlama yöntemini sudoku karelerine uygulamış ve buradan çıkan klasik sonuçların doğal uzantılarını araştırmışlardır.

Lewis (2007), metasezgisel tekniği ilk defa Sudoku probleminin çözümüne uyarlamıştır. Problemin çözümünde tavlama benzetimi algoritmasını kullanarak stokastik arama tabanlı bir algoritma geliştirmiştir. Birleşik Krallık gazetelerindeki sudoku problemlerine

algoritmayı uyguladığında, algoritmanın optimal sonuçlara yaklaşamadığını vurgulamakla birlikte, çözümü sıklıkla kabul edilebilir bir zaman dilimi içerisinde verdiğini belirtmiştir. Soto ve diğerleri (2013), sudoku problemini çözmek için yeni bir hibrit AC3-tabu arama algoritması önermişlerdir. Klasik tabu arama algoritmasını bağlantı-tutarlılığı (arc-consistency) üç (AC3) algoritmasıyla birleştirip ihtimal sayısını düşürmüşlerdir. Tabu aramasında AC3'ün tek bir ön işlem olarak değil her iterasyonunda geçerli olan tamamen entegre bir prosedür olarak hareket etmesini sağlamışlardır. Bu entegrasyon daha hızlı bir çözüm süreci sağlamıştır. Arama yöntemleri kullanılarak çözülen sonuçlardan daha iyi bir performans gösterdiğini deneysel olarak kanıtlamışlardır. Polnik ve diğerleri (2013), sudoku problemini Evrimsel Çoklu Ajan Sistemleri (EMAS) kullanarak ele almışlardır ve Paralel Evrimsel Algoritma (PEA) ile elde edilen sonuçları karşılaştırdıklarında, EMAS kullanılarak elde edilen stratejilerin daha başarılı olduklarını ortaya koymuşlardır.

Maji ve diğerleri (2013), ise sudoku problemini çözmek için kullanılan mini-karelerden (minigrid) sadece istenen permütasyonları üretmişlerdir. Yazarlar geri-izleme (backtracking) algoritması kullanmışlardır, fakat her bir hücre yerine mini karelere uygulamış ve bu şekilde çözüm zamanından kazanç sağlamışlardır. Bir diğer çalışmada, Geem (2007), harmonik aramayı farklı optimizasyon teknikleri kullanarak problemi 285 fonksiyon değerlendirmesinden sonra standart bir kişisel bilgisayar işlemcisi ile dokuz saniyede başarıyla çözebilmiştir. Chae ve Regan (2021), basit harmonik arama algoritmasının daha önceki araştırmalarda incelenen bir Sudoku bulmacasına uygulandığındaki süreci analiz etmişlerdir. Harmonik aramaya getirilen olumsuz genel geçer eleştirileri haklı bulmakla birlikte, birkaç ufak değişiklik ile birlikte harmonik aramanın performansının oldukça arttırılabileceğini ortaya koymuşlardır. İlk olarak arama yöntemini kolaylaştıran yeni bir amaç fonksiyonu önermişlerdir. İkinci olarak lokal araştırma ile birleştirerek geliştirdikleri doğaçlama (improvisation) tekniği ile problem çözüm süresini oldukça kısaltmışlardır. Sonrasında bu tekniği genişleterek çeşitli Sudoku problemleri için uygulamışlardır. Weyland (2015), ise metasezgisel bir çerçevede harmonik arama algoritmasının eleştirel bir analizini sunmuştur. Harmonik aramanın mevcut probleme yeni bir terminoloji getirmek dışında, diğer evrimsel algoritmalardan

farklı bir yenilik katmadığını ve harmonik arama ile elde edilen en verimli sonuçların dahi standart bir evrimsel algoritma ile kolayca elde edilebileceğini iddia etmiştir.

Berggren ve Nilsson (2012), üç farklı algoritma ile sudoku problemini çözmüştür. Araştırmalarında sadece çözüme değil, aynı zamanda zorluk derecesi, bulmaca üretebilme yeterliliği ve paralelleştirmeye uygunluğu da incelemişlerdir. Değerlendirilen algoritmalar geri-izleme, kural tabanlı bir yaklaşım ve Boltzmann makineler kullanarak sudoku probleminin çözümünde bu üç algoritmayı kıyaslamışlar ve en verimli algoritmanın kural tabanlı olduğuna karar vermişlerdir. Paralelleştirme, tüm algoritmalara farklı ölçüde uygulanabilmiştir. Boltzmann makinelerinin daha iyi şekilde paralelleştirebileceğini fakat düğüm durumlarının senkron güncelleştirmeleri nedeniyle biraz sınırlı olduğu sonucuna varmışlardır.

Bonde ve Agrawal (2015), sudoku probleminin çözümü için kaba kuvvet (brute-force) ve geri-izleme algoritmalarını kullanarak performans değerlendirmesi yapmışlardır. Değerlendirme yaparken dört farklı seviyedeki (kolay, orta, zor, çok zor) problemleri kullanmışlar ve sonuç olarak her seviyede kaba-kuvvet algoritmasının daha hızlı çalıştığını görmüşlerdir.

Mantere ve Koljonen (2007), çalışmalarında sudoku problemi ile ilgili üç farklı araştırma yapmışlardır. Birincisi, genetik algoritmanın yaptığı optimizasyonun sudoku problemi için etkili bir çözüm olup olmayacağıdır. İkincisi, genetik algoritmanın yeni sudoku problemi üretmede kullanılabilirliğidir. Üçüncüsü ise sudoku probleminin zorluğunu genetik algoritma ile değerlendiren bir derecelendirme makinesi olarak kullanılabilirliğidir. Araştırmalarının sonucu olarak sudoku probleminin kombinasyonel bir genetik algoritma ile etkili bir çözüme ulaştığını ispatlamışlardır. Deng ve Li (2013), sudoku probleminin çözümü için yeni bir hibrit genetik algoritma önermişlerdir. Önerdikleri algoritmada seçim, çaprazlama ve mutasyon operatörlerini sudoku probleminin özelliklerine göre uyarlamışlardır. Hibrit genetik algoritma kullanarak ulaştıkları sonuçları, gelişmiş genetik algoritma ve genetik algoritma ile kıyaslamışlardır. Önerilen hibrit algoritmanın kolay ve orta seviyedeki Sudoku problemlerinin çözümü için

kusursuza yakın sonuç verdiğinin altını çizmekle birlikte, zor ve süper zor seviyedeki bulmacalar için ideal olmadığını vurgulamışlardır.

Benzer şekilde, metasezgisel bir yaklaşımın değerlendirildiği başka bir çalışmada ise Schiff (2015), incelenen sudoku problemlerinin hepsi için optimal çözüm bulmayı sağlayan ilk karınca algoritmasını geliştirmiştir. Çalışmada önce genel karınca algoritmasını etraflıca tanımlamış, daha sonra bu algoritmayı sudoku problemine uyarlamıştır. Algoritma pek çok sudoku problemini milisaniyeler içerisinde çözerken, zorluk seviyesi maksimumlara çıkartıldığında çözüm süresinin 20-25 dakikalara çıktığını gözlemlemiştir. Asif ve Baig (2009), standart bir karınca kolonisi optimizasyon (Ant Colony Optimization-ACO) algoritmasının modifikasyonu ile sudoku problemini çözmüş ve diğer çözüm teknikleri ile kalitesini ve zaman performansını kıyaslamışlardır. Lloyd ve Amos (2019), sudoku problemi için karınca kolonisi algoritması tabanlı yeni bir algoritma geliştirmişlerdir. Geleneksel geri-izleme (backtracking) metotlarıyla karşılaştırıldığında önerilen algoritmanın çok daha etkili çözüm uzayları ürettiğini ortaya koymuşlardır.

Chlond (2005), ise sunduğu lineer tamsayılı programlama modeli üzerinden hem Sudoku hem de Log Pile bulmacalarının çözümü için matematiksel bir formülasyon önermiştir. Çalışmadaki bulmacalar diğerlerinden farklı olarak Excel yerine Xpress-Mosel programlama dili kullanılarak oluşturulmuştur. Weiss ve Rasmussen (2007), Chlond'un çalışmasının aksine sudokuyu Microsoft Excel programında çözülebilecek bir problem olarak ele almış ve VBA dilini kullanarak geliştirdikleri kısıtlar ile birlikte 3 boyutlu bir formülasyon önermişlerdir. Doğru bir formülasyon kullanmanın iyi bir çözüm algoritması geliştirmekten çok daha önemli olduğunu iddia etmişlerdir. Friesen ve diğerleri (2013), tamsayılı programlama modelini Excel'in Premium Solver'ı ile çözdükleri benzer bir çalışma sunmuşlardır.

Ernstberger ve Venkataramanan (2016), ise problemi Excel ortamında ele alan bir diğer çalışmaya imza atmışlardır. Nispeten az karar değişkeni, basitleştirilmiş kısıtlar ve Excel'de bulunan Evrimsel Çözücü kullanılarak çözülen alternatif bir Excel modeli formülasyonu geliştirmişlerdir.

3. MATERYAL ve YÖNTEM

3.1. Matematiksel Programlama Modeli

Bu tez çalışmasında sudokunun standart versiyonu olan 9x9 kare matris ele alınmıştır. Öncelikle aşağıda verilen matematiksel programlama modeli Visual Studio ortamında C# program dili ile kodlanıp Gurobi çözücüsü kullanılarak çözülmüştür. Probleme ait lineer-tamsayılı programlama modeli aşağıda sunulmuştur.

İndisler:

i:	Satır indisi, $i = 1, \dots, 9$
j:	Sütun indisi, $j = 1, \dots, 9$
k:	Değer indisi, $k = 1, \dots, 9$
i_0:	Alt matris satır indisi, $i_0 = 1, \dots, 3$
j_0:	Alt matris sütun indisi, $j_0 = 1, \dots, 3$
p_{ij}:	Başlangıçta verilen değerleri gösteren indis, $p_{ij} = 1, \dots, 9$

Değişkenler:

x_{ijk}: $\begin{cases} 1, & k \text{ değeri } i.\text{satır } j.\text{sütunda kullanılıyorsa} \\ 0, & \text{aksi durumda} \end{cases}$

Amaç fonksiyonu:

$$\min z = 0^T \mathrm{x} \tag{3.1}$$

Kısıtlar:

$$\sum_{k=1}^{9} x_{ijk} = 1, \qquad \forall i, j \tag{3.2}$$

$$\sum_{i=1}^{9} x_{ijk} = 1, \qquad \forall j, k \tag{3.3}$$

$$\sum_{j=1}^{9} x_{ijk} = 1, \qquad \forall i,k \tag{3.4}$$

$$\sum_{i=3i_0-2}^{3i_0} \sum_{j=3j_0-2}^{3j_0} x_{ijk} = 1, \qquad \forall i_0, j_0 \tag{3.5}$$

$$x_{ijp_{ij}} = 1, \qquad \forall i,j,p_{ij} \tag{3.6}$$

$$x_{ijk} \in \{0,1\}, \qquad \forall i,j,k \tag{3.7}$$

Yukarıda formülasyonu verilen sudoku bulmacasının çözümü problemi aslında bir sağlanabilirlik (satisfiability) veya uygunluk (feasibility) problemi olup, problemin çözümü için yukarıda verilen kısıtların sağlanması yeterlidir. Bu nedenle problem için amaç fonksiyonu tanımlamasına ihtiyaç olmayıp, Denklem (3.1) ile verilen amaç fonksiyonu kullanılabilir. Denklem (3.2) ile her hücrede bir adet rakam kullanılması sağlanmaktadır. Denklem (3.3) ile her rakamın her satırda yalnızca 1 kez kullanılması sağlanmaktadır. Denklem (3.4) ile her rakamın her sütunda yalnızca 1 kez kullanılması sağlanmaktadır. Denklem (3.5) ile her rakamın her alt matriste yalnızca 1 kez kullanılması sağlanmaktadır. Denklem (3.6) ile oyunun başlangıcında verilen değerlerin değiştirilemeyeceği ifade edilmektedir. Denklem (3.7) ile genel işaret kısıtı verilmiştir.

Giriş bölümünde bahsedilen ardışık sudoku için, oluşturulan matematiksel programlama modeline aşağıdaki gibi kısıtlar eklenebilir;

- Ardışık rakamların yer aldığı hücreler arasındaki denklemler: Bu denklemler, ardışık rakamların yer alacağı hücreler arasındaki farkın 1 olmasını garanti eder. Örneğin; x_{12k} ve x_{13k} hücreleri arasında ardışık olma koşulu olsun, bu iki hücre arasındaki kısıt Denklem (3.8) ile verilmiştir. Benzer şekilde diğer hücreler için de denklemler oluşturulur.

$$x_{12k} - x_{13k} = 1 \tag{3.8}$$

- Ardışık rakamların yer aldığı hücreler dışındaki hücreler için denklemler: Bu denklemler, ardışık rakamların yer aldığı hücreler dışındaki hücrelerin değerlerinin ardışık olmamasını garanti eder. Örneğin; x_{12k} ve x_{13k} hücreleri arasında ardışık olmama koşulu olsun, bu iki hücre arasındaki kısıt Denklem (3.9) ile verilmiştir. Benzer şekilde diğer hücreler için de denklemler oluşturulur.

$$x_{12k} - x_{13k} > 1 \tag{3.9}$$

Bu şekilde, ardışık sudoku için gerekli olan kısıtlar eklenmiş olur ve matematiksel programlama modeli oluşturulmuş olur.

3.2. Sudoku Bulmacası Oluşturma Algoritması

Tez çalışmasının ikinci kısmında, Sudoku problemini çözmenin çözümün yanı sıra sudoku bulmacası oluşturma algoritması geliştirilip Unity editör ortamında C# programlama dili ile kodlanıp bir oyun tasarımı gerçekleştirilmiştir.

Sudoku bulmacası oluşturmak için geliştirilen önerilen geri-izleme algoritmasının adımları aşağıda verilmiştir.

Adım (0)-Başlangıç Adımı: 9x9 Sudoku matrisini 1'den 9'a kadar olan tamsayılarla rassal olarak doldur. Sudoku matrisinin doldurulmasında, her yeni tamsayı girişinde, matrisin sudoku bulmacasının kısıtlarını sağlayıp sağlamadığını kontrol et. Kısıtlar sağlanmıyorsa yeni bir tamsayı dene. Matrisin tüm elemanları doldurulunca kadar işleme devam et. Oluşan matrisi güncel bulmaca matrisi olarak işaretle.
Adım (1): Güncel bulmaca matrisinden rassal olarak seçilen bir değeri kaldır. Oluşan matrisi geçici bulmaca matrisi olarak işaretle.
Adım (2): Mevcut geçici bulmaca matrisi için bir çözüm olup olmadığını görmek ve alternatif çözümlerin sayısını tutmak için matrisi üzerinde kaba-kuvvet (brute-force) algoritmasını uygula. Eğer mevcut geçici matris için sadece tekil bir çözüm varsa, mevcut matrisi güncel bulmaca matrisi olarak işaretle. Bulmaca için zorluk seviyesi uygun ise dur. Aksi halde, Adım (1) ile devam et.

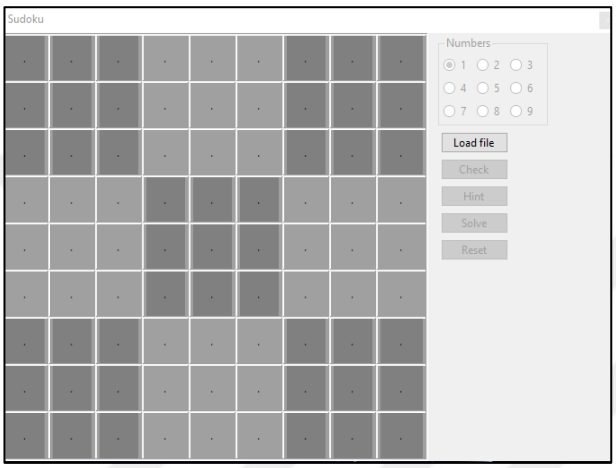
Şekil 3.1. Sudoku bulmacası oluşturma algoritması

Zorluk seviyesi görünürdeki rakam sayısı ile ilişkilendirilmiştir. "Easy (Kolay)" zorluk seviyesinde deneme sayısı 5, "Medium (Orta)" zorluk seviyesinde deneme sayısı 20 ve "Hard (Zor)" zorluk seviyesinde deneme sayısı ise 35 olarak alınmıştır. Deneme sayısı arttıkça başlangıç matrisinden çıkarılan değer sayısı artmakta ve dolayısıyla görünürdeki rakam sayısı ve bulmaca zorlaşmaktadır.

4. BULGULAR

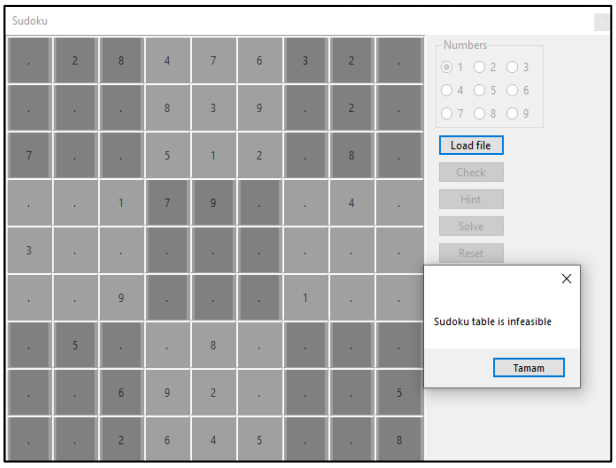
4.1 Matematiksel Programlama Modeli

Bu alt bölümde, Visual Studio ortamında sudoku problemini çözmek amacıyla geliştirilmiş olan matematiksel programlama bazlı yaklaşımın uygulaması sunulmaktadır. Visual Studio ortamında oluşturulan uygulamanın başlangıç ekranı Şekil 4.1'deki gibidir.



Şekil 4.1. Başlangıç ekranı

"Load file" butonuna tıklanarak dışarıdan başlangıç matrisi dosyası yüklenir. Yüklenen matrisin geçerliliği kontrol edilir. Geçerliliği kontrol etmek için tamsayılı programlama modeli Gurobi ile çözdürülür. Eğer bulmacanın çözümü yoksa "Sudoku table is infeasible (Sudoku tablosu uygun değil)" uyarısı verilir. Eğer bulmacanın çözümü varsa Gurobi çözümü saklanır ve süreölçer başlatılır. Şekil 4.2'de başlangıç matrisinin geçersiz olmasının sebebi $x_{18} = 2$ değerinin satırda $x_{12} = 2$ değeri ile sütunda ise $x_{28} = 2$ değeri ile çakışmasıdır.



Şekil 4.2. Geçersiz başlangıç matrisi

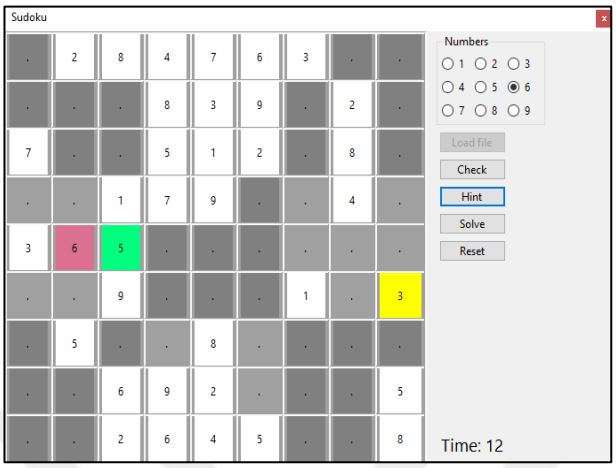
Yüklenen geçerli başlangıç matrisi "P_1" Denklem 4.1 ile verilmiştir.

$$P_1 = \begin{bmatrix} . & 2 & 8 & 4 & 7 & 6 & 3 & . & . \\ . & . & . & 8 & 3 & 9 & . & 2 & . \\ 7 & . & . & 5 & 1 & 2 & . & 8 & . \\ . & . & 1 & 7 & 9 & . & . & 4 & . \\ 3 & . & . & . & . & . & . & . & . \\ . & . & 9 & . & . & . & 1 & . & . \\ . & 5 & . & . & 8 & . & . & . & . \\ . & . & 6 & 9 & 2 & . & . & . & 5 \\ . & . & 2 & 6 & 4 & 5 & . & . & 8 \end{bmatrix} \quad (4.1)$$

Uygulamada sağ tarafta yer alan "Numbers (Sayılar)" kısmından her seferinde bir rakam seçilerek hücreler doldurulur. "Check (Kontrol)" butonu doldurulan hücreyi saklanan Gurobi çözümü ile kıyaslayıp, doğru olması durumunda yeşil, yanlış olması durumunda ise kırmızı ile renklendirir. Şekil 4.3'te $x_{52} = 6$ değeri yanlış olup kırmızı ile renklendirilmiştir, $x_{53} = 5$ değeri ise doğru olup yeşil ile renklendirilmiştir.

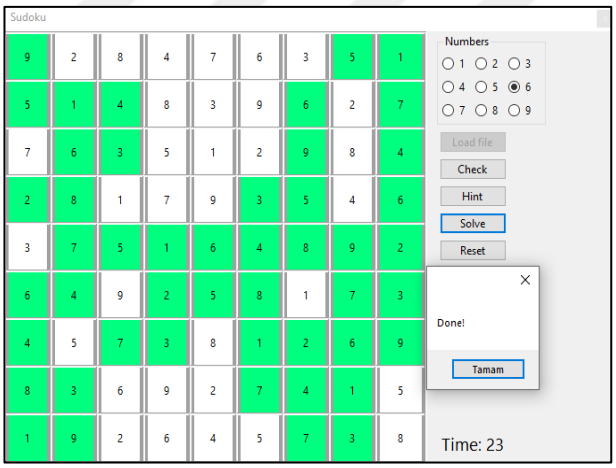
"Hint (İpucu)" butonuna basıldığı zaman önce kullanıcının doldurmadığı hücre olup olmadığı kontrol edilir, boş hücre var ise bu hücreler arasından rastgele bir hücre seçilip, bu hücreye denk gelen cevabı saklanan Gurobi çözümünden çekilip ilgili yere yazdırılır, sarı ile renklendirilir. Eğer doldurulmamış hücre yoksa tüm hücrelerin doğru olup

olmadığına bakılır, çözüm ile eşleşmeyen bir hücre bulunursa bu hücreyi doğru cevapla değiştirilir. Şekil 4.3'te $x_{69} = 3$ değeri ipucu olarak verilmiştir.



Şekil 4.3. Check & Hint butonu

"Solve (Çöz)" butonu ile tamsayılı programlama modelinin çözümü getirilir ve süreölçer durdurulur. Şekil 4.4'te "P_1" matrisinin çözümü verilmiştir.



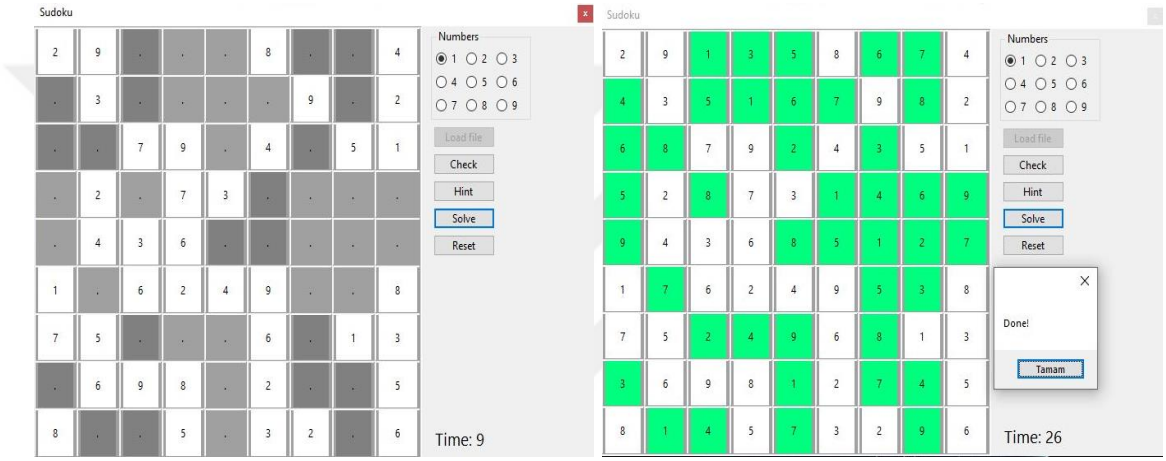
Şekil 4.4. P_1 matrisinin çözümü

"Reset (Yenile)" butonu ile bulmaca sıfırlanır ve Şekil 4.1'deki başlangıç ekranına dönülerek yeni başlangıç matrisi yüklenmesine olanak sağlanır.

Yüklenen yeni geçerli başlangıç matrisi "P_2" Denklem 4.2 ile verilmiştir.

$$P_2 = \begin{bmatrix} 2 & 9 & . & . & . & 8 & . & . & 4 \\ . & 3 & . & . & . & . & 9 & . & 2 \\ . & . & 7 & 9 & . & 4 & . & 5 & 1 \\ . & 2 & . & 7 & 3 & . & . & . & . \\ . & 4 & 3 & 6 & . & . & . & . & . \\ 1 & . & 6 & 2 & 4 & 9 & . & . & 8 \\ 7 & 5 & . & . & . & 6 & . & 1 & 3 \\ . & 6 & 9 & 8 & . & 2 & . & . & 5 \\ 8 & . & . & 5 & . & 3 & 2 & . & 6 \end{bmatrix} \quad (4.2)$$

"P_2" matrisine karşılık gelen bulmaca ve çözümü Şekil 4.5'te verilmiştir.



(a) (b)

Şekil 4.5. P_2 matrisine karşılık gelen bulmaca (a) ve çözümü (b)

4.2 Sudoku Bulmacası Oluşturma Algoritması

Bu alt bölümde, Unity ortamında sudoku bulmacası oluşturmak amacıyla geliştirilmiş olan algoritmanın uygulaması sunulmaktadır.

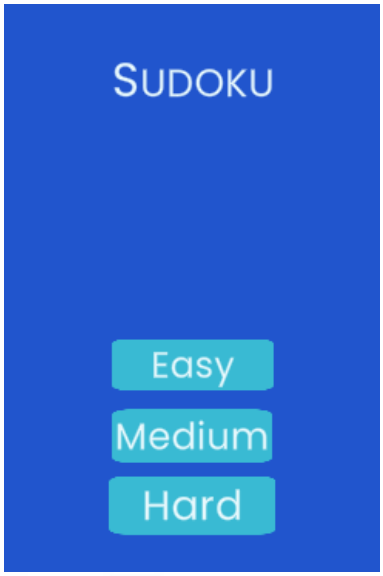
Oluşturulan bulmacanın geçerli ve tek bir çözümle çözülebilmesi için yeterli sayıda dolu hücreye sahip olması gerekmektedir. Barlett ve diğerleri (2008), makalelerinde bu sayının 17 olduğunu belirtmişlerdir.

Sudoku bulmacası oluşturulurken tüm olası çözümleri araştırmak için 3.2 bölümünde de adımları verilen geri-izleme algoritması kullanılmıştır. Algoritmanın adımları aşağıdaki gibidir:

Adım (0)-Başlangıç Adımı: 9x9 Sudoku matrisini 1'den 9'a kadar olan tamsayılarla rassal olarak doldur. Sudoku matrisinin doldurulmasında, her yeni tamsayı girişinde, matrisin sudoku bulmacasının kısıtlarını sağlayıp sağlamadığını kontrol et. Kısıtlar sağlanmıyorsa yeni bir tamsayı dene. Matrisin tüm elemanları doldurulunca kadar işleme devam et. Oluşan matrisi güncel bulmaca matrisi olarak işaretle.
Adım (1): Güncel bulmaca matrisinden rassal olarak seçilen bir değeri kaldır. Oluşan matrisi geçici bulmaca matrisi olarak işaretle.
Adım (2): Mevcut geçici bulmaca matrisi için bir çözüm olup olmadığını görmek ve alternatif çözümlerin sayısını tutmak için matrisi üzerinde kaba-kuvvet (brute-force) algoritmasını uygula. Eğer mevcut geçici matris için sadece tekil bir çözüm varsa, mevcut matrisi güncel bulmaca matrisi olarak işaretle. Bulmaca için zorluk seviyesi uygun ise dur. Aksi halde, Adım (1) ile devam et.

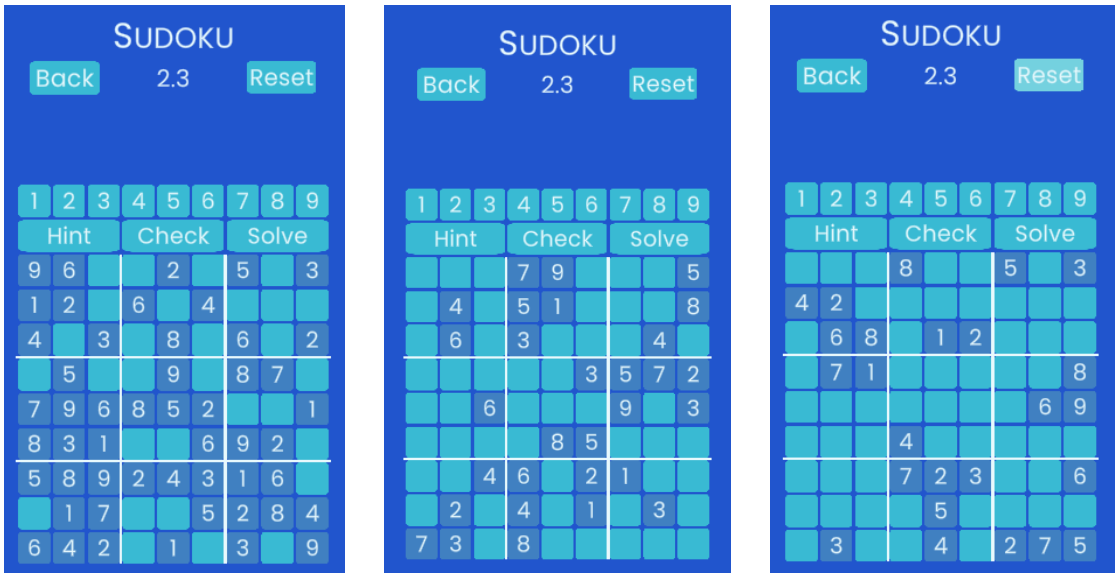
Şekil 4.6. Sudoku bulmacası oluşturma algoritması

Unity ortamında oluşturulan bulmacanın ana ekranı Şekil 4.7'de verilmiştir. Zorluk seviyesi bu ekranda seçilmektedir. Zorluk seviyesi seçildikten sonra süreölçer başlamaktadır.



Şekil 4.7. Ana ekran

Şekil 4.8'de her zorluk seviyesine ait bir örnek verilmiştir. Çalışma kapsamında "Easy (Kolay)" zorluk seviyesinde deneme sayısı 5, "Medium (Orta)" zorluk seviyesinde deneme sayısı 20 ve "Hard (Zor)" zorluk seviyesinde deneme sayısı ise 35 olarak alınmıştır. Deneme sayısı arttıkça başlangıç matrisinden çıkarılan değer sayısı artmakta ve dolayısıyla bulmaca zorlaşmaktadır.



(a) (b) (c)

Şekil 4.8. Easy (Kolay) (a), Medium (Orta) (b), Hard (Zor) (c) zorluk seviyelerine ait başlangıçlar

Uygulamanın devamında Şekil 4.8'deki "Hard (Zor)" seviyesine ait başlangıç matrisi ile devam edilmiştir. Uygulamada üst kısımda 1'den 9'a kadar rakamlar bulunmaktadır. Buradan rakam seçilerek hücreler doldurulur. "Check (Kontrol)" butonu doldurulan hücreyi çözüm ile kıyaslayıp, doğru olması durumunda yeşil ile renklendirir, yanlış olması durumunda kırmızı ile renklendirir. Şekil 4.9'da $x_{67} = 4$ değeri yanlış olup kırmızı ile renklendirilmiştir, $x_{69} = 2$ değeri ise doğru olup yeşil ile renklendirilmiştir.

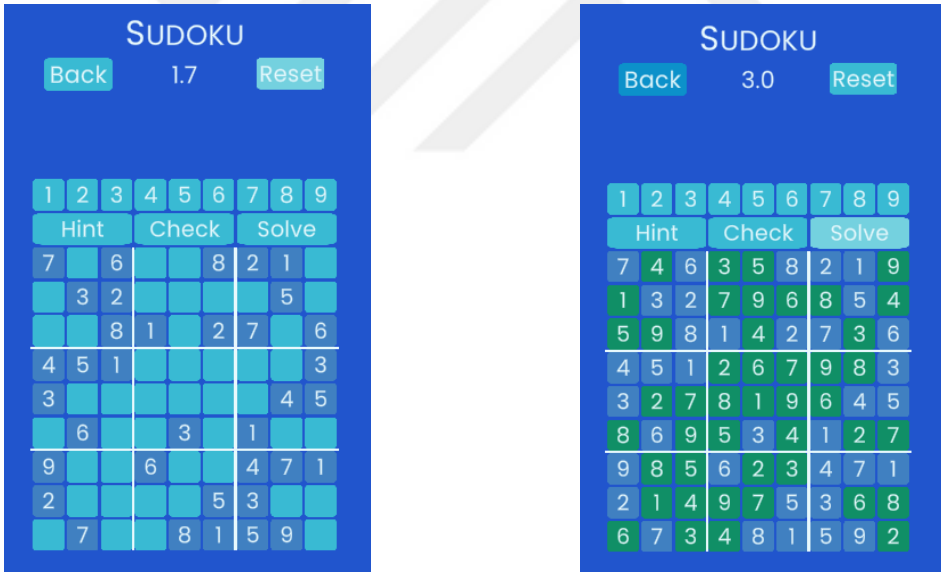
Şekil 4.9. Check (Kontrol) & Hint (İpucu) butonu

"Hint (İpucu)" butonuna basıldığı zaman önce kullanıcının doldurmadığı hücre olup olmadığı kontrol edilir, boş hücre var ise bu hücreler arasından rastgele bir hücre seçilip, bu hücreye denk gelen cevabı çözümden çekilip ilgili yere yazdırılır, gri ile renklendirilir. Eğer doldurulmamış hücre yoksa tüm hücrelerin doğru olup olmadığına bakılır, çözüm ile eşleşmeyen bir hücre bulunursa bu hücreyi doğru cevapla değiştirilir.
Şekil 4.9'da $x_{34} = 3$ değeri ipucu olarak verilmiştir.

"Solve (Çöz)" butonu ile başlangıçta oluşturulmuş olan çözüm getirilir ve süreölçer durdurulur. Şekil 4.10'da "Hard (Zor)" seviyesine ait bulmacanın çözümü verilmiştir.

Şekil 4.10. Hard (Zor) seviyesi çözümü

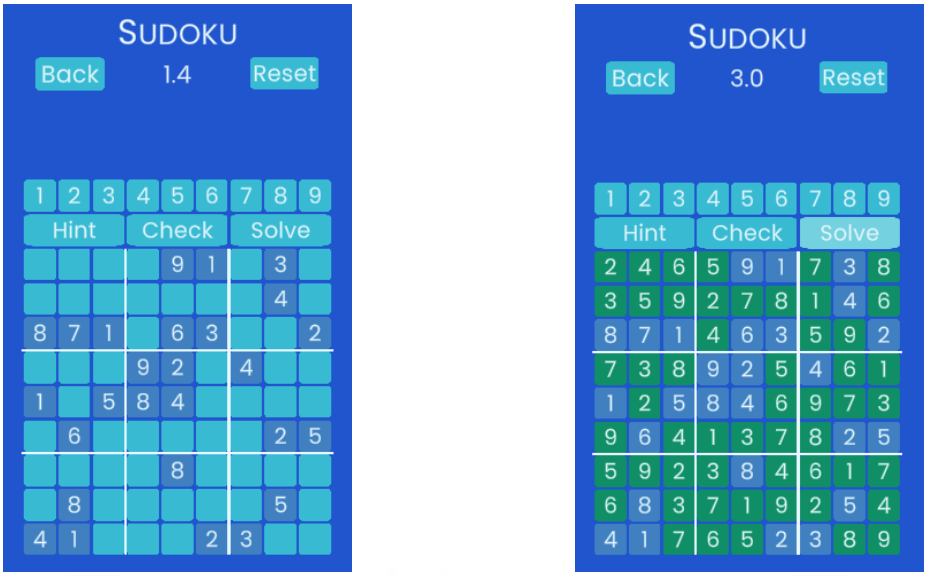
"Easy (Kolay)" zorluk seviyesine ait oluşturulan bir bulmaca ve çözümü Şekil 4.11'de verilmiştir.



(a) (b)

Şekil 4.11. Easy (Kolay) zorluk seviyesi başlangıç matrisi (a) ve çözümü (b)

"Medium (Orta)" zorluk seviyesine ait oluşturulan bir bulmaca ve çözümü Şekil 4.12'de verilmiştir.



(a) (b)

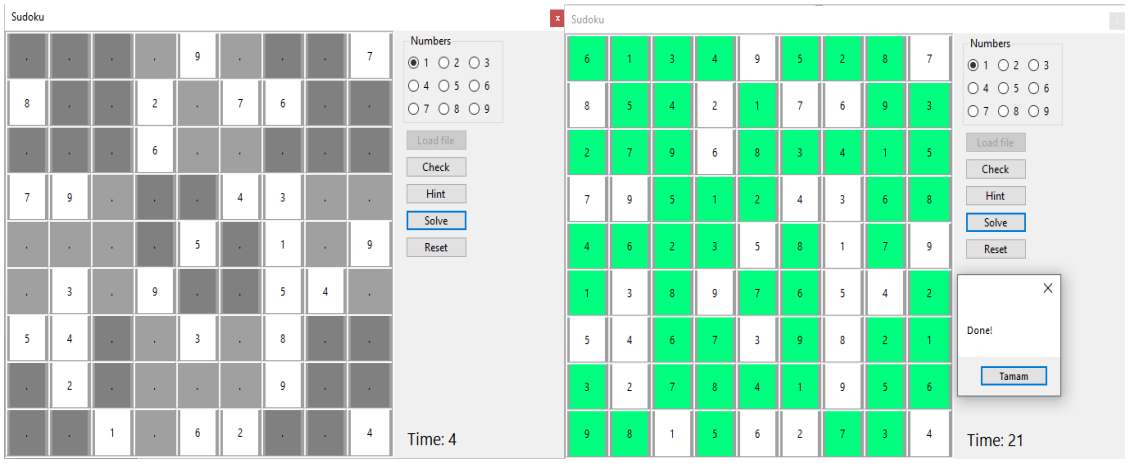
Şekil 4.12. Medium (Orta) zorluk seviyesi başlangıç matrisi (a) ve çözümü (b)

4.3 Gurobi Çözümü & Kaba-Kuvvet Algoritması Çözümü Karşılaştırması

Bu çalışmanın ilk kısmında sudoku problemi Gurobi çözücü ile çözülmüştür, ikinci kısımda ise herhangi bir çözücü kullanılmadan kaba-kuvvet algoritması ile çözülmüştür. Bu iki çözümü karşılaştırmak amacıyla kullanılan başlangıç matrisi "P_3" Denklem 4.3 ile verilmiştir. Boş olan 53 hücreden 30 tanesinin yani %56'sının aynı rakamla doldurulduğu görülmüştür.

$$P_3 = \begin{bmatrix} . & . & . & . & 9 & . & . & . & 7 \\ 8 & . & . & 2 & . & 7 & 6 & . & . \\ . & . & . & 6 & . & . & . & . & . \\ 7 & 9 & . & . & . & 4 & 3 & . & . \\ . & . & . & . & 5 & . & 1 & . & 9 \\ . & 3 & . & 9 & . & . & 5 & 4 & . \\ 5 & 4 & . & . & 3 & . & 8 & . & . \\ . & 2 & . & . & . & . & 9 & . & . \\ . & . & 1 & . & 6 & 2 & . & . & 4 \end{bmatrix} \tag{4.3}$$

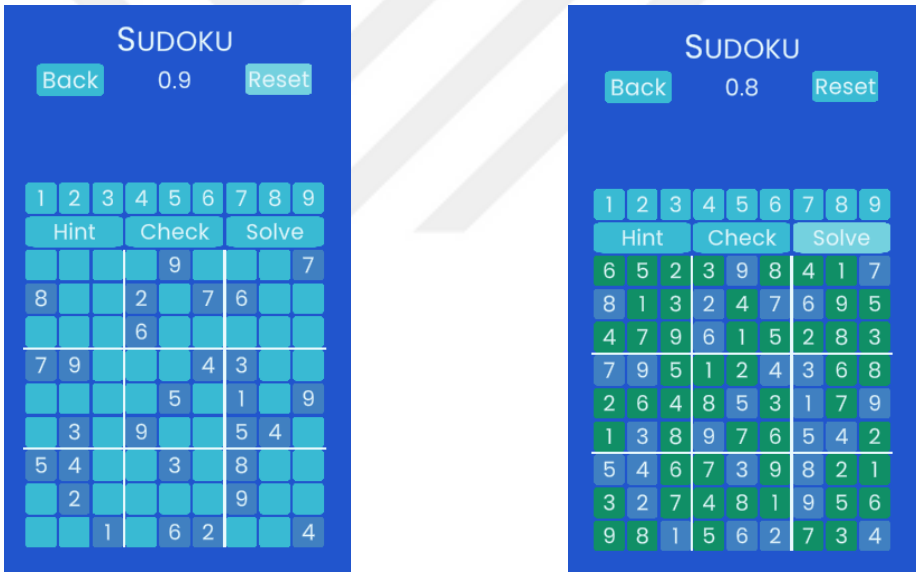
"P_3" matrisine ait Gurobi çözümü Şekil 4.13'te verilmiştir.



(a) (b)

Şekil 4.13. P_3 başlangıç matrisi (a) ve Gurobi ile çözümü (b)

"P_3" matrisine ait kaba-kuvvet algoritması ile çözüm Şekil 4.14'te verilmiştir.



(a) (b)

Şekil 4.14. P_3 başlangıç matrisi (a) ve kaba- kuvvet algoritması ile çözümü (b)

Aynı P_3 başlangıç matrisine ait uygun (feasible) alternatif çözümler olabileceği görülmektedir.

5. TARTIŞMA ve SONUÇ

Sudoku dünya çapında popülerlik kazanmış bir mantık bulmacasıdır. Günümüzde hem geleneksel platformlarda, hem de bilgisayar ortamında ve mobil platformlarda sudoku bulmacaları sıklıkla paylaşılmaktadır. Aynı zamanda bu popülerliğin de bir sonucu olarak, sudoku bulmacasıyla ilgili konularla ilgili olarak özellikle matematik ve mühendislik gibi alanlar başta olmak üzere çeşitli akademik çalışmaların gerçekleştirildiği görülmektedir.

Literatürde sudoku probleminin çözümü için farklı yaklaşımların kullanıldığı gözlemlenmektedir. Bu çalışmaların bazılarında sudoku probleminin bir sağlanabilirlik veya uygunluk problemi olarak ele alınarak çözülmesinin önerildiği matematiksel programlama temelli yaklaşımlar dikkate alınmaktadır.

Bu tez çalışmasında da öncelikle matematiksel programlama kullanılarak sudoku probleminin çözümü incelenmiş ve sonrasında ise bir sudoku bulmacası oluşturma algoritması geliştirilerek, çalışmanın her iki aşamasında da ilgili yaklaşımları temel alan oyun tasarımları gerçekleştirilmiştir. Çalışma kapsamında standart sudoku bulmacası dikkate alınmış olmakla birlikte, önerilen yaklaşımların geliştirilip genelleştirilerek diğer sudoku çeşitleri üzerinde de uygulanabileceği belirtilebilir.

Tez çalışması kapsamında öncelikle sudoku probleminin çözümü için bir matematiksel programlama formülasyonu dikkate alınmış, problem bir sağlanabilirlik veya uygunluk problemi olarak modellenerek, Visual Studio ortamında C# programlama dili ve Gurobi çözücü ile çözülmüştür. Buna ek olarak, yine Visual Studio ortamında bir oyun tasarımı gerçekleştirilmiştir. Tez çalışmasının ikinci kısmında, herhangi bir çözücü kullanılmadan kaba-kuvvet algoritmasıyla Sudoku problemini çözmenin yanı sıra, ayrıca bir sudoku bulmacası oluşturma algoritması geliştirilip Unity ortamında C# programlama dili ile kodlanıp bir oyun tasarımı gerçekleştirilmiştir. Çalışmanın her iki aşamasında da önerilen yaklaşımlar test edilmiş, örnek problemler ve bulmacalar oluşturulmuştur.

Gelecek çalışmalar kapsamında önerilen yaklaşımların geliştirilip genelleştirilerek farklı sudoku çeşitleri üzerinde de kullanılması söz konusu olabilir. Buna ek olarak, sudoku probleminin çözümü ve bulmaca oluşturulması sırasında farklı yaklaşımların dikkate alınması da gelecek çalışmalar kapsamında dikkate alınabilir. Bir diğer gelecek çalışma kapsamının ise sudoku bulmacalarındaki zorluk seviyesinin belirlenmesine yönelik olabileceği düşünülmektedir. Son olarak, önerilen yaklaşımların bulmaca oluşturmak amacıyla kullanılabilmesi ve daha çok kişiyle paylaşılabilmesinin sağlanması amacıyla farklı platformlarda sunulacak bir tasarım gerçekleştirilmesi de gelecek çalışmalar kapsamında dikkate alınabilir.

KAYNAKLAR

Agrawal, A., & Bonde, P. (2015). Study on the Performance Characteristics of Sudoku Solving Algorithms. *International Journal of Computer Applications*, *122*(1).

Asif, M., & Baig, R. (2009, October). Solving NP-complete problem using ACO algorithm. In *2009 International conference on emerging technologies* (pp. 13-16). IEEE.

Bartlett, A., Chartier, T. P., Langville, A. N., & Rankin, T. D. (2008). An integer programming model for the Sudoku problem. *Journal of Online Mathematics and its Applications*, *8*(1).

Berggren, P., & Nilsson, D. (2012). A study of Sudoku solving algorithms. *Royal Institute of Technology, Stockholm*.

Chadwick, S. B., Krieg, R. M., & Granade, C. E. (2007). Ease and toil: Analyzing sudoku. *UMAP Journal*, *29*(3), 363.

Chae, R. H., & Regan, A. C. (2021). An analysis of Harmony Search for solving Sudoku puzzles. *Soft Computing Letters*, *3*, 100017.

Chang, C., Fan, Z., & Sun, Y. (2007). A Difficulty Metric and Puzzle Generator for Sudoku. *UMAPJournal*, 305.

Chlond, M. J. (2005). Classroom exercises in IP modeling: Su Doku and the log pile. *INFORMS Transactions on Education*, *5*(2), 77-79.

Coelho, L. C., & Laporte, G. (2014). A comparison of several enumerative algorithms for Sudoku. *Journal of the Operational Research Society*, *65*(10), 1602-1610.

Deng, X. Q., & Li, Y. D. (2013). A novel hybrid genetic algorithm for solving Sudoku puzzles. *Optimization Letters*, *7*(2), 241-257.

Ernstberger, K. W., & Venkataramanan, M. A. (2016). A genetic spreadsheet model for solving Sudoku problems. *Business Management Dynamics*, *6*(3), 1.

Friesen, D. D., Patterson, M. C., & Harmel, B. (2013). A spreadsheet optimization model for solving sudoku problems. *Business Management Dynamics*, *2*(9), 15.

Gabor, A. F., & Woeginger, G. J. (2010). How∗ not∗ to solve a Sudoku. *Operations research letters*, *38*(6), 582-584.

Geem, Z. W. (2007, September). Harmony search algorithm for solving sudoku. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems* (pp. 371-378). Springer, Berlin, Heidelberg.

http://www.dailysudoku.com/sudoku/play.shtml / Erişim Tarihi : (17.01.2022)

https://en.wikipedia.org/wiki/Glossary_of_Sudoku / Erişim Tarihi (05.12.2022)

https://en.wikipedia.org/wiki/Sudoku / Erişim Tarihi: (17.01.2022)

Koch, T. (2006). Rapid mathematical programming or how to solve sudoku puzzles in a few seconds. In *Operations Research Proceedings 2005* (pp. 21-26). Springer, Berlin, Heidelberg.

Lewis, R. (2007). Metaheuristics can solve sudoku puzzles. *Journal of heuristics*, *13*(4), 387-401.

Lloyd, H., & Amos, M. (2019). Solving Sudoku with ant colony optimization. *IEEE Transactions on Games*, *12*(3), 302-311.

Lynce, I., & Ouaknine, J. (2006). Sudoku as a SAT Problem. *ISAIM*, *11*(1), 6-13.

Mahdian, M., & Mahmoodian, E. S. (2015). Sudoku rectangle completion. *Electronic Notes in Discrete Mathematics*, *49*, 747-755. [12] Soto, R., Crawford, B., Galleguillos, C., Monfroy, E., & Paredes, F. (2013). A hybrid ac3-tabu search algorithm for solving sudoku puzzles. Expert Systems with Applications, 40(15), 5817-5821.

Maji, A. K., Jana, S., & Pal, R. K. (2013). An algorithm for generating only desired permutations for solving Sudoku puzzle. *Procedia Technology*, *10*, 392-399.

Mantere, T., & Koljonen, J. (2007, September). Solving, rating and generating Sudoku puzzles with GA. In *2007 IEEE congress on evolutionary computation* (pp. 1382-1389). IEEE.

Oleinik, E. B., & Rogulin, R. S. (2018, October). Sudoku: Another aspect of the application for solving the problem of optimal allocation of resources. In *The International Science and Technology Conference" FarEastCon"* (pp. 536-543). Springer, Cham.

Polnik, M., Kumięga, M., & Byrski, A. (2013). Agent-based optimization of advisory strategy parameters. *Journal of Telecommunications and Information Technology*, (2), 49-55.

Santos-García, G., & Palomino, M. (2007). Solving Sudoku puzzles with rewriting rules. *Electronic Notes in Theoretical Computer Science*, *176*(4), 79-93.

Schiff, K. (2015). An ant algorithm for the Sudoku problem. *Journal of Automation Mobile Robotics and Intelligent Systems*, *9*.

Simonis, H. (2005, October). Sudoku as a constraint problem. In *CP Workshop on modeling and reformulating Constraint Satisfaction Problems* (Vol. 12, pp. 13-27). Citeseer.

Soto, R., Crawford, B., Galleguillos, C., Monfroy, E., & Paredes, F. (2013). A hybrid ac3-tabu search algorithm for solving sudoku puzzles. *Expert Systems with Applications*, *40*(15), 5817-5821.

Weiss, H. J., & Rasmussen, R. A. (2007). Lessons from modeling sudoku in excel. *INFORMS Transactions on Education*, *7*(2), 178-184.

Weyland, D. (2015). A critical analysis of the harmony search algorithm—how not to solve sudoku. *Operations Research Perspectives*, *2*, 97-105.

Yu, H., Tang, Y., & Zong, C. (2016, August). Solving odd even sudoku puzzles by binary integer linear programming. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)* (pp. 2226-2230). IEEE.