

T.C.
KARADENİZ TEKNİK ÜNİVERSİTESİ * FEN FAKÜLTESİ
İSTATİSTİK ve BİLGİSAYAR BİLİMLERİ BÖLÜMÜ

IST4000 BİTİRME ÇALIŞMASI

MiniZinc İLE OPTİMİZASYON

Hazırlayanlar: 357795 Samet KARASU
348877 Ertuğrul EŞOL

Trabzon-2020

T.C.
KARADENİZ TEKNİK ÜNİVERSİTESİ * FEN FAKÜLTESİ
İSTATİSTİK ve BİLGİSAYAR BİLİMLERİ BÖLÜMÜ

IST4000 BİTİRME ÇALIŞMASI

MiniZinc İLE OPTİMİZASYON

Hazırlayanlar: 357795 Samet KARASU
348877 Ertuğrul EŞOL

Tez Danışmanı: Dr. Öğr. Üyesi Tolga BERBER

Trabzon-2020

İÇİNDEKİLER

İÇİNDEKİLER.....	II
GİRİŞ	1

BÖLÜM I

1. OPTİMİZASYON'A GİRİŞ VE TEMEL KAVRAMLAR.....	2
1.1. Optimizasyon Nedir ?.....	2
1.2. Doğrusal Programlama	4
1.2.1. Bir Doğrusal Programlama Modelinin Genel Yapısı.....	4
1.2.2. Doğrusal Programlamada Çözüm Yöntemleri.....	5
1.2.2.1. Grafik Yöntemi ve Örnekler	5

BÖLÜM II

2. MiniZinc ile OPTİMİZASYON	15
2.1. MiniZinc Nedir ?	15
2.1.1 MiniZinc'in Özellikleri	15
2.1.2 MiniZinc'de Kullanılan Bazı Komutlar	16
2.1.3 Bir Optimizasyon Modeline Genel Bakış	17
2.2. Kurulum.....	17
2.3. MiniZinc ile İlk Adımlar.....	19
2.3.1. MiniZinc IDE	19

BÖLÜM III

3. MiniZinc İLE UYGULAMALAR.....	25
3.1. Optimizasyon Problemleri MiniZinc'e Aktarılması ve Çözümü.....	25

GİRİŞ

Optimizasyon bir karar verme problemi için eniyi kararların belirlenmesine ilişkin sistematik yöntemleri içerir. Üzerinde durulan problem için anlamlı performans kriterlerinin ve dikkat edilmesi gereken kısıtların matematiksel olarak ifade edilerek formülasyonlarının geliştirilmesi optimizasyonun temelini oluşturur. Kararlar belirlenirken, mevcut kısıtları gözeterek eniyilik kriterine ilişkin bir amaç fonksiyonunun enküçüklenmesi veya enbüyüklenmesi söz konusudur.

MiniZinc ise ücretsiz ve açık kaynaklı bir kısıt modelleme dilidir ve MiniZinc'i, önceden tanımlanmış kısıtlamalardan oluşan geniş bir kütüphaneden yararlanarak optimizasyon sorunlarını üst düzey, çözücüden bağımsız bir şekilde modellemek için kullanılmaktadır.

Sınırlı kaynakların etkin bir şekilde kullanılıp artan rekabet ortamında başarılı sonuçların elde edilebilmesi için optimizasyon tekniklerinin kullanılmasının önemi aşikardır ve bu nedenle de optimizasyonun çok çeşitli sektörde kullanılması kaçınılmazdır. Bu çalışmamızda söz konusu sektörler için bu denli önemli olan optimizasyon problemlerinin daha hızlı ve etkin olarak çözüme kavuşması için açık kaynaklı program olan MiniZinc ile optimizasyon problemlerinin çözülmesi ve MiniZinc'in tanıtılması amaçlanmıştır.

BİRİNCİ BÖLÜM

1. OPTİMİZASYON'A GİRİŞ VE TEMEL KAVRAMLAR

1.1. Optimizasyon Nedir?

Optimizasyon, verilen amaç veya amaçlar doğrultusunda belirli kısıtlamaların sağlanarak en uygun çözümün elde edilme sürecidir. Optimizasyon, karar verme süreçlerini hızlandırmakta ve karar kalitesini arttırmakta kullanılarak gerçek hayatta karşılaşılan problemlerin etkin, doğru ve gerçek zamanlı çözümünde yararlanılmaktadır. Optimizasyon, ekonomik açılarından getirdiği kazançların yanında müşteri, işveren ve çalışanların tercih ve kısıtlarının karar sürecinde yer almasında ve sistemde yer alan kaynakların kalitesinin yükseltilmesinde de etkin bir şekilde başvurulan bir yöntem olarak kullanılmaktadır.

Matematiksel olarak ifade edilirse; optimizasyon, kısaca bir fonksiyonun minimize veya maksimize edilmesi olarak tanımlanabilir. Türkçe anlam karşılığı ise “uyumlaşma”, “uyumlaşım” ya da “uyumlaştırma” şeklindedir. Yani bir gerçel fonksiyonu minimize ya da maksimize etmek amacı ile gerçek ya da tamsayı değerlerini tanımlı bir aralıkta seçip fonksiyona yerleştirerek sistematik olarak bir problemi incelemek ya da çözmek işlemlerini ifade eder.

Optimizasyon, en iyi olanı aramaktır. Bulunan en iyi ise “optimum” olarak adlandırılır. Optimizasyonun amacı, tanımından da anlaşıldığı üzere, en iyi sonuca, en iyi hedefe ulaşmaktır. En iyi sonuca ulaşmak için de mevcut durum ya da durumlar üzerinde iyileştirmeler gerçekleştirilebilir. Bu noktada mühendislik devreye girer

Optimizasyonun Aşamaları

Bir sistemin tasarlanma sürecinde önce sorun gözlemlenerek ihtiyaçlar belirlenir. Daha sonra bu ihtiyaçlar doğrultusunda sistem tasarımı yapılarak denemelerle iyileştirilir. İhtiyaçlar belirlendikten sonra ise sistemin çalışması dahil olmak üzere tüm aşamalarda optimizasyon devreye girer. Bir sistem var oldukça, optimizasyon da o sistem için var olacaktır.

- Sistemin temelini oluşturulması
- Tasarım değişkenlerinin tanımlanması
- Hedef fonksiyonun belirlenmesi
- Kısıtların belirlenmesi
- Uygun optimizasyon metodunun seçilmesi ve uygulanması

Optimizasyonun Kullanım Alanları

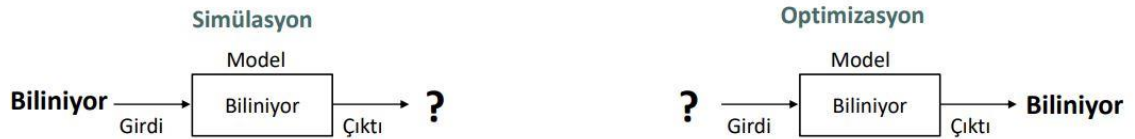
- Yapı-Araç İskeleti Dinamiği'ne ilişkin problemler
- Dizayn problemleri
- Ekonomi problemleri
- Zaman ve maliyet problemleri
- Operasyon araştırmaları
- Benzetim (Simülasyon) ile Senaryo Analizleri

En basit anlamı ile **optimizasyon** eldeki kısıtlı kaynakları en optimum biçimde kullanmak olarak tanımlanabilir. Matematiksel olarak ifade etmek gerekirse optimizasyon kısaca bir fonksiyonun minimize veya maksimize edilmesi olarak tanımlanabilir. Diğer bir deyişle optimizasyon “**en iyi**

amaç kriterinin en iyi değerini veren kısıtlardaki değişkenlerin değerini bulmaktır". Başka bir tanımlama ile **"belirli amaçları gerçekleştirmek için en iyi kararları verme sanatı"** veya **"belirli koşullar altında herhangi bir şeyi en iyi yapma"** olarak da tanımlanan optimizasyon kısaca **"en iyi sonuçları içeren işlemler topluluğudur"**.

Optimizasyonda bir amaç da maksimum kar veya minimum maliyeti sağlayacak üretim miktarını kısıtlara bağlı olarak tespit etmektir. Günümüzün bilgisayar teknolojisi kadar güncel bir kavram olan optimizasyon kavramı çok çeşitli endüstri kesimlerinde uygulama olanağı bulmuştur.

Optimizasyon, en genel anlamıyla, bir sistemde, belirli kısıtlar altında, belirlenmiş bir amaç fonksiyonunun değerinin en iyilenmesi amacıyla karar değişkenlerinin alacağı değerleri belirleme işlemidir. Diğer bir ifade ile, istenen bir çıktıyı elde etmek amacıyla, sistem girdilerinin ve/veya bu girdilerin değerlerinin ne olacağını belirleme sürecidir. Amaç fonksiyonunun en iyilenmesi demek, problemin türüne göre en küçüklenmesi (minimizasyon) veya en büyüklenmesi (maksimizasyon) olabilir. Örneğin bir çizelgeleme probleminin, geciken iş sayısını en küçüklemek amacıyla çözülmesi bir optimizasyon problemidir. Aynı şekilde, bir montaj hattı dengeleme probleminin istasyon sayısını minimize etmek (veya hat etkinliğini maksimize etmek) amacıyla, veya bir ekonomik stok miktarı belirleme probleminin toplam maliyeti minimize etmek amacıyla çözülmesi birer optimizasyon işlemidir.



Simülasyon, girdilerin bilindiği bir sistemde, çıktının tahmin edilmesi, belirlenmesi, sürecidir. Örneğin, siparişlerin geliş zamanının ve sistemde geçireceği sürelerin bilindiği bir üretim sürecinin simülasyonu sonucu, üretimin ne zaman tamamlanacağı, sistemdeki boş zamanlar, sistemin etkinliği vs. büyük oranda hesaplanabilir (tahmin edilebilir).

Optimizasyon ise, istenen çıktıyı elde edebilmek amacıyla, girdilerin veya bu girdilerin değerlerinin ne olacağını belirlenmesi sürecidir. Örneğin, istenen yüzey pürüzlülüğünün elde edilebilmesi amacıyla, bir 3B yazıcının parametrelerinin ne olacağını belirlenmesi optimizasyona örnektir.

Karar değişkeni, problem çözüldüğü zaman değeri belirlenecek olan sistem ögesidir. Yani aslında karar değişkenlerinin alacağı değerlerin belirlenme süreci, problem çözme sürecidir. Karar değişkenlerinin alacağı değerler, amaç fonksiyonunun değerinin ne olacağına etki eder. Fakat bu süreçte bazı kısıtların sağlanması gerekmektedir.

Çözüm uzayındaki, problemin kısıtlarını sağlayan herhangi bir çözüm, **uygun çözüm** olarak adlandırılır. **Optimum çözüm** veya **optimal çözüm** ise, uygun çözümler arasından, belirlenen amaca göre en iyi çözümü ifade etmektedir. Yani uygun çözüm, doğrusal programlama probleminin tüm kısıtlarını doyuran çözüm iken optimal çözüm tüm uygun çözümler arasında amaç fonksiyonunu iyi karşılayan çözümdür.

1.2. Doğrusal Programlama

Bir doğrusal programlama problemi genel itibari ile amaç fonksiyonu ve doğrusal sınır/sınırların yer aldığı iki kısımlı bir matematiksel ifadedir. Bu matematiksel ifade ile bir amaç maksimize ya da minimize edilir. Doğrusallık ifadesi modelde yer alan tüm değişkenler (fonksiyonlar) arasındaki ilişkinin doğrusal olmasından kaynaklanmaktadır. Doğrusal sınırların oluşturduğu kesişim kümesinden yola çıkılarak mümkün çözümler ya da uygun çözüm alanı belirlenir. Belirlenen uygun çözüm alanı ise amaç doğrultusunda eniyilemeye çalışılır.

Doğrusal programlama (DP) ile bağımsız değişkenlerden oluşan bir dizi fonksiyon ile yine bir dizi bağımsız değişkenlerin bir fonksiyonu olan bağımlı değişkenin optimal değeri belirlenmeye çalışılır. Bir başka ifade ile doğrusal programlama, belirli bir amacı eniyilemek maksadıyla sınırlı kaynakların nasıl dağıtılması gerektiğine çözüm arayan bir karar verme aracıdır.

Doğrusal programlama, karşılıklı ilişkileri doğrusal nitelikte olan ve sınırları kısıtlı kaynaklarla çevrili iki veya daha fazla değişkenlerin, belirlenen amacı eniyileyecek şekilde miktarlarının bulunması yöntemidir. Burada; eniyilenmek (max/min) üzere tek bir amaç, bu amacı sınırlandıran kısıtlar ve amaca ulaşmak için alternatif yollar mevcuttur. Bu üç öge birlikte doğrusal programlama tekniğinin ana çerçevesini oluşturmaktadır.

Bir Doğrusal Programlama Modeli, doğrusal kısıtlar altında doğrusal bir fonksiyonun değerini maksimize ya da minimize etmeye çalışır. Doğrusal Programlama belli bir amacı gerçekleştirmek için sınırlı kaynakların etkin kullanımını ve çeşitli seçenekler arasında en uygun dağılımını sağlayan matematiksel bir tekniktir.

1.2.1. Bir Doğrusal Programlama Modelinin Genel Yapısı

Amaç Fonksiyonu: $Z = \sum_{j=1}^N C_j x_j$

Kısıtlar: $\sum_{i=1}^M \sum_{j=1}^N a_{ij} x_j \begin{bmatrix} \leq \\ = \\ \geq \end{bmatrix} B_i$

Karar Değişkenleri: x_j

Pozitif Koşulu: $x_j \geq 0$

Bu formülde kullanılan matematik notasyonların açıklamaları ise şöyledir:

- $i = 1, 2, \dots, M$ (kısıt sayısı)
- $j = 1, 2, \dots, N$ (karar değişken sayısı)
- x_j = Karar değişkenleri
- C_j = Amaç fonksiyonu katsayıları, j karar değişkeninin katkı katsayıları
- a_{ij} = i kısıtındaki j karar değişkeninin teknik katsayısı
- B_i = i kısıtının sağ taraf değeri (STD) yahut kaynak değeri

Kapalı matematiksel formülü verilen bir doğrusal programlama modelinin cebirsel açılımı ise şöyledir:

Max/Min =	c_1x_1	+	c_2x_2	+	+	c_nx_n	STD
	$a_{11}x_1$	+	$a_{12}x_2$	+	+	$a_{1n}x_n$	$[<=,=,>=]$
	$a_{21}x_1$	+	$a_{22}x_2$	+	+	$a_{2n}x_n$	$[<=,=,>=]$
	$a_{31}x_1$	+	$a_{32}x_2$	+	+	$a_{3n}x_n$	$[<=,=,>=]$
Kısıtlar

	$a_{m1}x_1$	+	$a_{m2}x_2$	+	+	$a_{mn}x_n$	$[<=,=,>=]$
	x_1	,	x_2	,	,	x_n	$>=$
								0

1.2.2. Doğrusal Programlamada Çözüm Yöntemleri

Doğrusal programlama modelleri aşağıda sıralanan yöntemler ile çözülebilir:

1. Grafik Yöntemi
2. Cebirsel Yöntem
3. Simpleks Yöntem
4. İleri Doğrusal Programlama Çözüm Yöntemleri

Optimizasyon problemlerinin optimum değerlerini bulmak için bu yöntemlerden sadece grafik yöntemi kullanılacaktır.

1.2.2.1. Grafik Yöntemi ve Örnekler

Bir doğrusal programlama probleminin grafik çözümünde aşağıdaki adımlar izlenir:

1. Değişkenlerin koordinat sisteminin yatay ve dikey eksenlerine yerleştirilmesi,
2. Kısıtlayıcı fonksiyonların grafiğinin çizilmesi,
3. Uygun çözüm bölgesinin belirlenmesi,
4. En iyi çözümün araştırılması.

Soru 1:

Bir imalatçı A ve B olmak üzere iki farklı modelde karton katlama makinası satın almak istiyor. Model-A'daki makine dakikada 30 karton katlıyor ve 1 kişinin yardımına ihtiyacı var. Model-B ise dakikada 50 karton katlıyor ve 2 kişinin yardımına ihtiyaç duyuyor. İmalatçı dakikada 320 karton katlamak istiyor ve 12 kişiden fazlası kullanılamayacaktır diyor. Model-A'nın maliyeti 150 birim, Model-B'nin maliyeti ise 200 birimdir. İmalatçı maliyeti en küçük olacak biçimde her tip makinadan kaç tane alması modelini matematiksel olarak çözünüz.

x_1 : A katlama makinasından kaç tane alınması gerektiği
 x_2 : B katlama makinasından kaç tane alınması gerektiği

Soruda Model-A'nın maliyeti 150 birim, Model-B'nin maliyeti ise 200 birim olduğundan ve imalatçı da maliyeti en küçük olacak biçimde talep ettiğinden amaç fonksiyonumuz:

$$\text{Min } Z = 150.x_1 + 200.x_2$$

Sorudan hareketle Model-A'daki makine dakikada 30 karton katlıyor ve 1 kişinin yardımına ihtiyacı var. Model-B ise dakikada 50 karton katlıyor ve 2 kişinin yardımına ihtiyaç duyuyor. İmalatçı dakikada 320 karton katlamak istediğinden ve 12 kişiden fazlası kullanılamayacağından karton kısıtı(1) ve kişi kısıtı(2) olacak şekilde teknolojik(sistem) kısıtlarımız aşağıdaki gibi olacaktır:

$$(1). 30.x_1 + 50.x_2 \geq 320$$

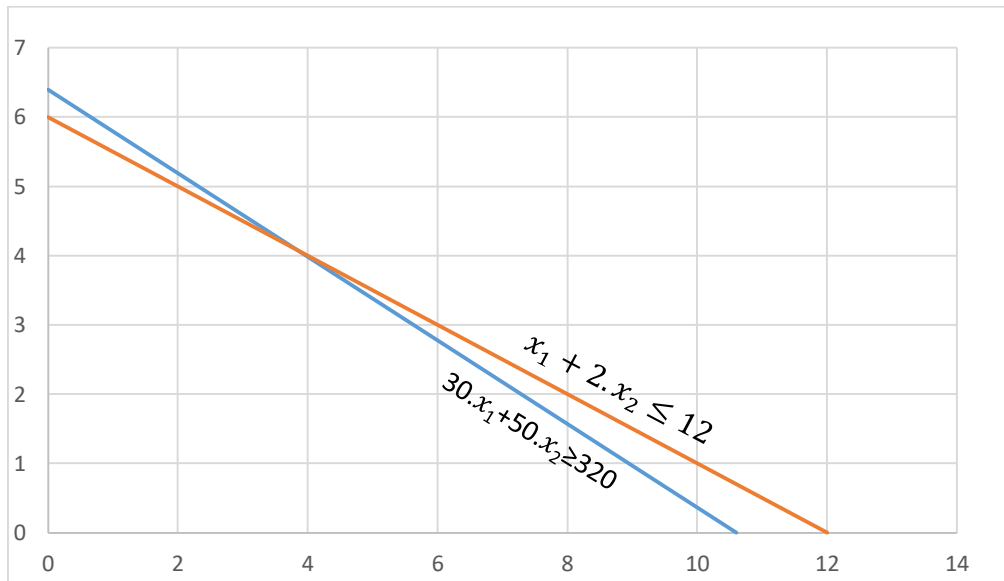
$$(2). x_1 + 2.x_2 \leq 12$$

İfadelerimiz sıfırdan büyük olacağından -yani üretmemiz gereken miktar negatif olamayacağından- negatif olmama kısıtımız da aşağıdaki gibi yazılmalıdır:

$$x_1, x_2 \geq 0$$

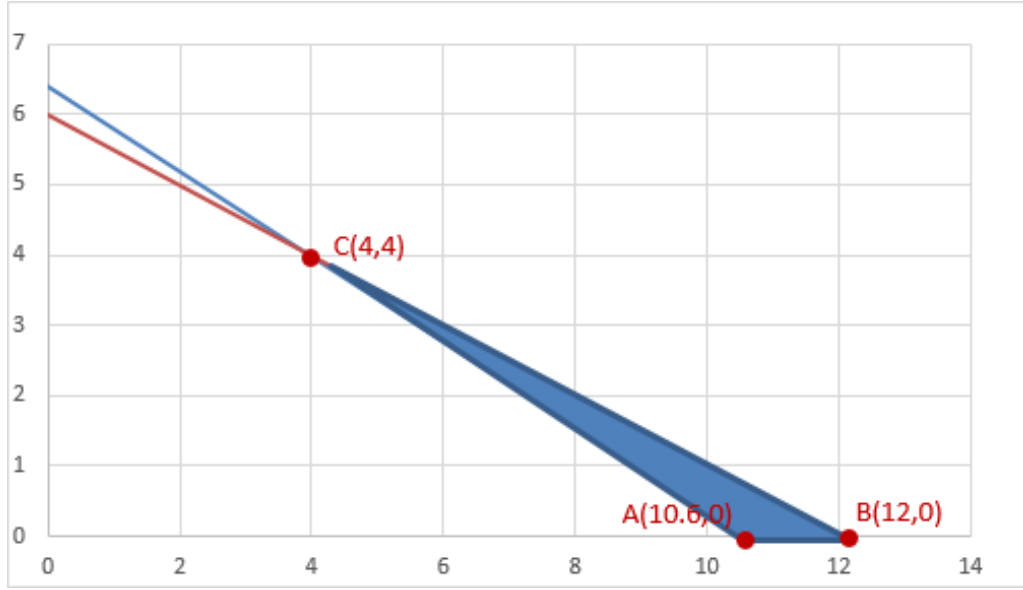
Bu sistem kısıtlarımızın grafiğini çizelim.

Grafik 1.1: Soru 1'deki Sistem Kısıtlarının Koordinat Sistemindeki Görünümü



Sistem kısıtlarımız koordinat düzleminde Grafik 1.1'deki gibi görünecektir. (1) kısıtımız “ \geq ”, (2) kısıtımız ise “ \leq ” ibaresi olduğundan “ $30.x_1 + 50.x_2 \geq 320$ ” kısıtı yukarı taraf, $x_1 + 2.x_2 \leq 12$ kısıtı ise aşağı taraf taranmalıdır. Bu açıklamaya göre uygun çözüm alanımız ve köşe noktalarımız ise aşağıdaki gibi olacaktır:

Grafik 1.2:Soru 1'in Uygun Çözüm Alanı ve Köşe Noktaları



Bu noktalarımızı amaç fonksiyonunda yerine koyacak olursak sonuçlar aşağıdaki gibi olacaktır:

Tablo 1.1: Grafik 1.2'deki Köşe Noktalarının Amaç Fonksiyonundaki Değer Tablosu

Noktalar	$150.x_1 + 200.x_2$
(10.6,0)	1590
(12,0)	1800
(4,4)	1400

Tablo 1.1'de görüldüğü üzere $(x_1, x_2) = (4, 4)$ noktası bizim optimal çözümümüzdür. Yani imalatçı, A modelinden ve B modelinden 4'er tane karton katlama makinesi satın alırsa, kısıtları sağlayacak şekilde minimum maliyetle(1400) en iyi sonucu elde etmiş olur.

Soru 2:

Bir şirket 2 çeşit ayakkabı üretmektedir. Her biri için gereken iş gücü sırasıyla 2 ve 3 saattir ve gereken malzeme miktarı sırasıyla 3 ve 2 kg/çifttir. Şirketin aylık 30 saat iş gücü ve 40kg malzemesi bulunmaktadır. Ayakkabıların çiftlerinden sırasıyla 4 ve 6 birim kar elde edilmektedir. Şirket kârı en yüksek olacak şekilde her bir çeşit ayakkabıyı ne kadar üretmesi modelini matematiksel olarak çözünüz.

x1: 1. ayakkabı x2: 2. ayakkabı

Soru-2'de 1. ayakkabının maliyeti 4 birim, 2. ayakkabının maliyeti ise 6 birim olduğundan ve şirket kârı en yüksek olacak biçimde ayakkabı üretmek istediğinden amaç fonksiyonumuz:

$$Max Z = 4.x_1 + 6.x_2$$

Sorudan hareketle her bir ayakkabı için gereken iş gücü sırasıyla 2 ve 3 saattir ve gereken malzeme miktarı sırasıyla 3 ve 2 kg/çifttir. Şirketin aylık 30 saat iş gücü ve 40kg malzemesi bulunduğundan saat kısıtı(1) ve malzeme miktarı(2) olacak şekilde sistem kısıtlarımız aşağıdaki gibi olacaktır:

$$(1). 2.x_1 + 3.x_2 \leq 30$$

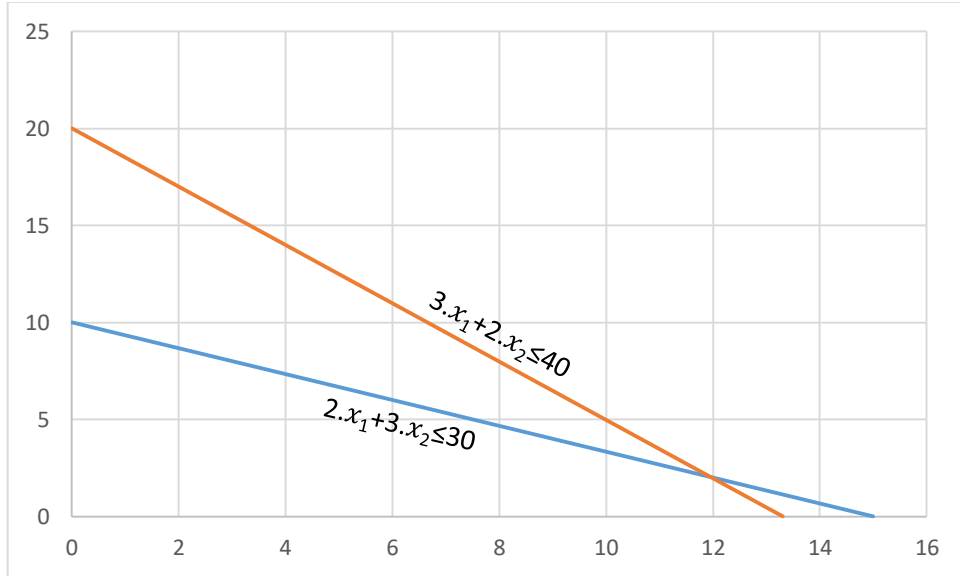
$$(2). 3.x_1 + 2.x_2 \leq 40$$

Tabi ki ifadelerimiz sıfırdan büyük olacağından -yani üretmemiz gereken miktar negatif olamayacağından- negatif olmama kısıtımız da aşağıdaki gibi yazılmalıdır:

$$x_1, x_2 \geq 0$$

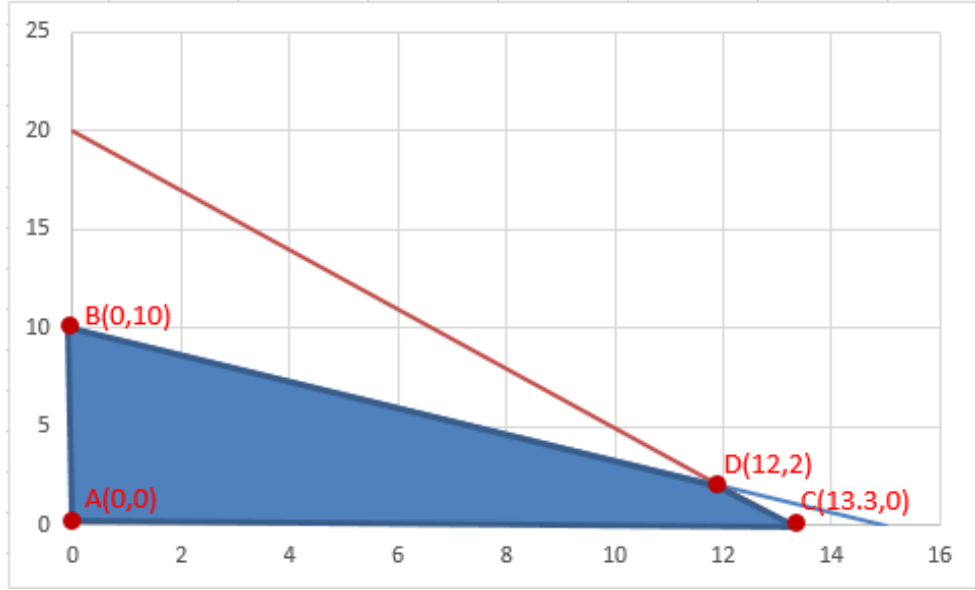
Sistem kısıtlarımızın grafiğini çizecek olursak:

Grafik 1.3: Soru 2’deki Sistem Kısıtlarının Koordinat Sistemindeki Görünümü



Sistem kısıtlarımız koordinat düzleminde Grafik 1.3’teki gibi görünecektir. Her iki kısıtta da “≤” ibaresi olduğundan aşağı taraf taranmalıdır. Bu açıklamaya göre uygun çözüm alanı ve köşe noktaları ise aşağıdaki gibi olacaktır:

Grafik 1.4: Soru 2'nin Uygun Çözüm Alanı ve Köşe Noktaları



Bu noktalarımızı amaç fonksiyonunda yerine koyacak olursak sonuçlar aşağıdaki gibi olacaktır:

Tablo 1.2: Grafik 1.4'teki Köşe Noktalarının Amaç Fonksiyonundaki Değer Tablosu

Noktalar	$4x_1 + 6x_2$
(0,0)	0
(0,10)	60
(13.3,0)	53,2
(12,2)	60

Tablo 1.2'de görüldüğü üzere iki farklı optimal çözüm elde edildi. Yani şirket 1. ayakkabıdan 0, 2. ayakkabıdan 10 adet üretse; veya 1. ayakkabıdan 12 adet, 2. ayakkabıdan ise 2 adet ayakkabı üretse dahi maksimum kârı(60) elde etmiş olur.

Soru 3:

Giapetto tahtadan oyuncak asker ve tren yapmaktadır. Satış fiyatları, bir oyuncak asker için \$3, bir oyuncak tren için \$2'dir. Her bir asker için 2 saat montaj ve 1 saat marangozluk gerekirken, her bir tren için 1 saat montaj ve 1 saat marangozluk gerekmektedir. Eldeki hammadde miktarı sınırsızdır, fakat haftada en çok 100 saat montaj ve 80 saat marangozluk kullanabilen Giapetto'nun haftada en fazla 40 oyuncak asker satabileceğini göz önünde bulundurarak kârını maksimum yapmak için hangi oyuncaktan haftada kaç adet üretmesi gerektiğini bulunuz.

x_1 = Bir haftada üretilen asker sayısı x_2 = Bir haftada üretilen tren sayısı

Soru-3'te bir oyuncak askerin maliyeti 3 birim, oyuncak trenin maliyeti ise 2 birim olduğundan ve Giapetto kârı en yüksek yapmak istediğinden amaç fonksiyonumuz:

$$\text{Max } Z = 3.x_1 + 2.x_2$$

Sorudan hareketle her bir asker için 2 saat montaj ve 1 saat marangozluk gerekirken, her bir tren için 1 saat montaj ve 1 saat marangozluk gerekmektedir. Eldeki hammadde miktarı sınırsızdır, fakat haftada en çok 100 saat montaj ve 80 saat marangozluk kullanabilen Giapetto'nun haftada en fazla 40 oyuncak asker satabileceğini göz önünde bulundurarak montaj kısıtı(1), marangozluk kısıtı(2) ve talep kısıtı(3) olacak şekilde sistem kısıtlarımız aşağıdaki gibi olmalıdır:

$$(1). 2.x_1 + x_2 \leq 100$$

$$(2). x_1 + x_2 \leq 80$$

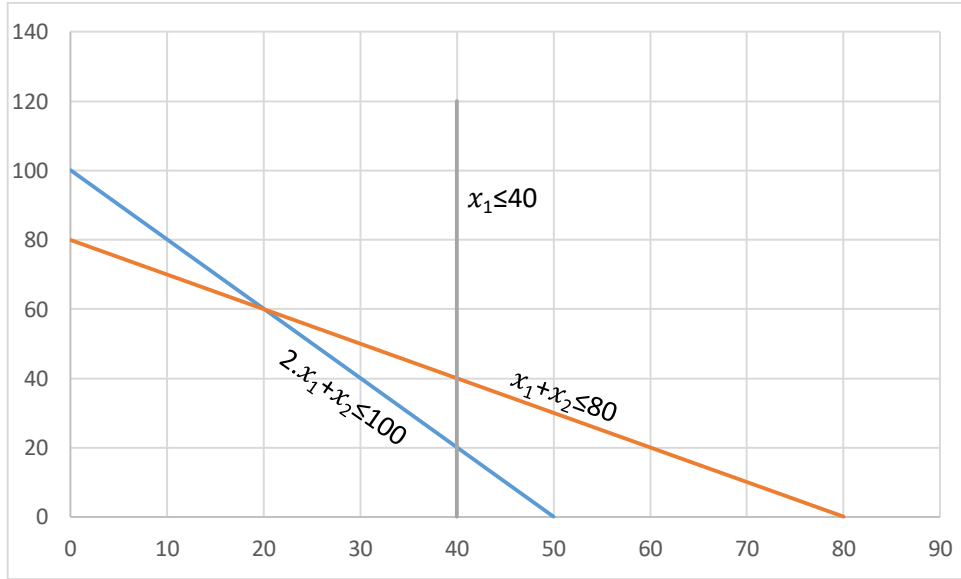
$$(3). x_1 \leq 40$$

Negatif olmama kısıtımız ise:

$$x_1, x_2 \geq 0$$

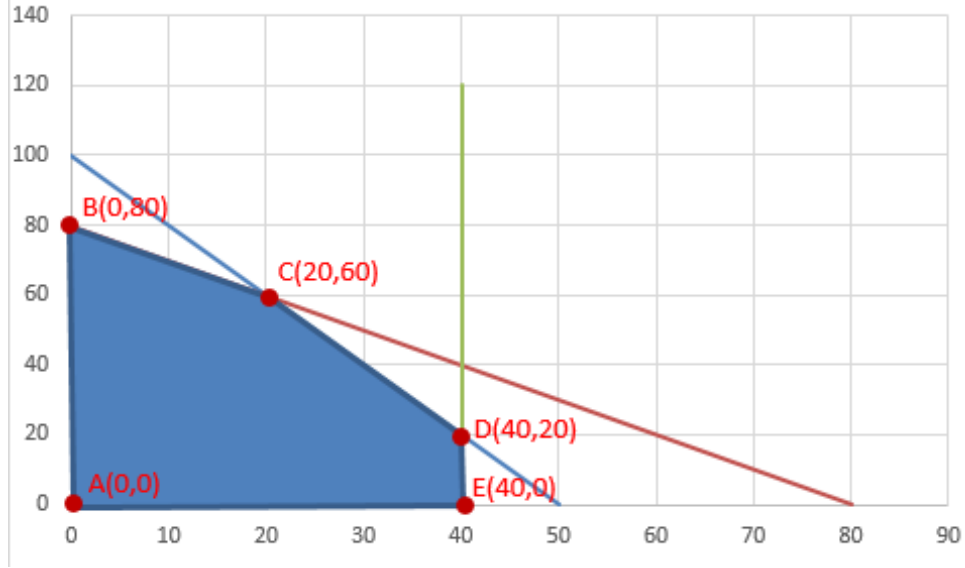
Sistem kısıtlarımızın grafiğini çizecek olursak:

Grafik 1.5: Soru 3'teki Sistem Kısıtlarının Koordinat Sistemindeki Görünümü



Sistem kısıtlarımız koordinat düzleminde Grafik 1.5'teki gibi görünecektir. Her iki kısıtta da “≤” ibaresi olduğundan aşağı taraf taranmalıdır. Bu açıklamaya göre uygun çözüm alanı ve köşe noktaları ise aşağıdaki gibidir:

Grafik 1.6: Soru 3'ün Uygun Çözüm Alanı ve Köşe Noktaları



Bu noktalarımızı amaç fonksiyonunda yerine koyacak olursak:

Tablo 1.3: Grafik 2.6'teki Köşe Noktalarının Amaç Fonksiyonundaki Değer Tablosu

Noktalar	$3x_1 + 2x_2$
(0,0)	0
(0,80)	160
(20,60)	180
(40,20)	160
(40,0)	120

Tablo 1.3'te görüldüğü gibi $(x_1, x_2) = (20, 60)$ noktası bizim optimal çözümümüz olacaktır. Yani Giapetto haftada 20 adet oyuncak asker, 60 adet de oyuncak tren üretirse, kısıtları sağlayacak şekilde maksimum kârda(180) en iyi çözümü elde etmiş olur.

Soru 4:

Bir elbise dikim firması, fabrika işçileri için iş elbisesi(üniforma) siparişi almaktadır ve sipariş için 240m kumaşı mevcuttur. Üniformalar A ve B tipli model olarak hazırlanacaktır. Her bir A tipli model yaklaşık 3 saat, B tipli model ise 1 saat işlem gerektirmekte ve bu üretim için günlük toplam 320 saatlik bir işgücü mevcut bulunmaktadır. A ve B tip her bir modelin gerektirdiği kumaş miktarları ise sırası ile 2m ve 1m kadardır. A ve B tip bir her bir modelin satışından elde edilecek kar ise sırasıyla 25 ve 20 TL'dir. Günlük üretimden elde edilecek olan karın maksimum olması için hangi modelden ne kadar üretilmelidir?

Soru-4'te A tip elbise modelinin maliyeti 25 birim, B tip elbise modelinin maliyeti ise 20 birim olduğundan ve dikim firması kârı en yüksek yapmak istediğinden amaç fonksiyonumuz aşağıdaki gibi olacaktır:

$$\text{Max } Z = 25 \cdot x_1 + 20 \cdot x_2$$

Sorudan hareketle sipariş için 240m kumaşı mevcuttur. Üniformalar A ve B tipli model olarak hazırlanacaktır. Her bir A tipli model yaklaşık 3 saat, B tipli model ise 1 saat işlem gerektirmekte ve bu üretim için günlük toplam 320 saatlik bir işgücü mevcut bulunmaktadır. A ve B tip her bir modelin gerektirdiği kumaş miktarları ise sırası ile 2m ve 1m kadardır. Bu istekler doğrultusunda saat(1) ve işgücü(2) olacak şekilde kısıtlarımız şu şekildedir:

$$(1). 2 \cdot x_1 + x_2 \leq 240$$

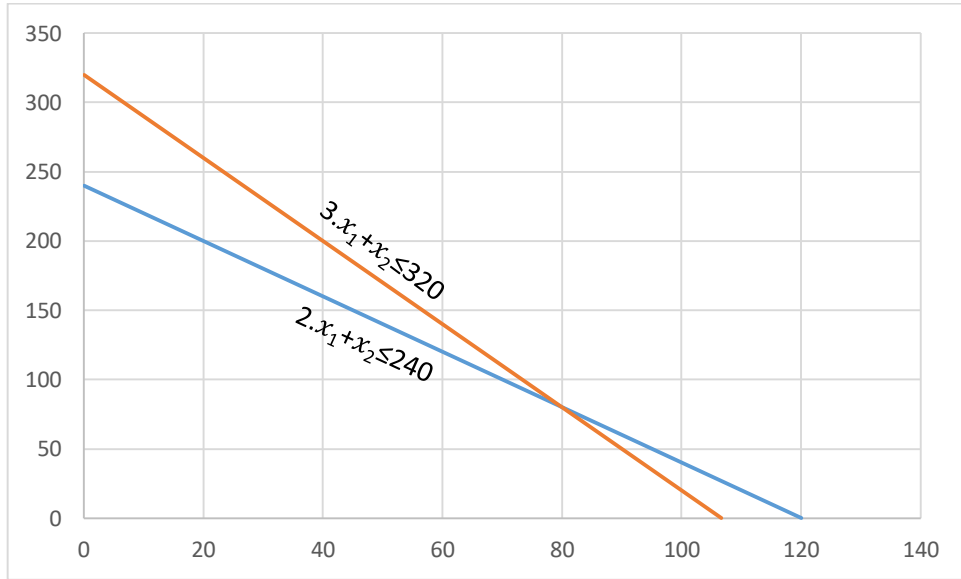
$$(2). 3 \cdot x_1 + x_2 \leq 320$$

Negatif olmama kısıtımız ise:

$$x_1, x_2 \geq 0$$

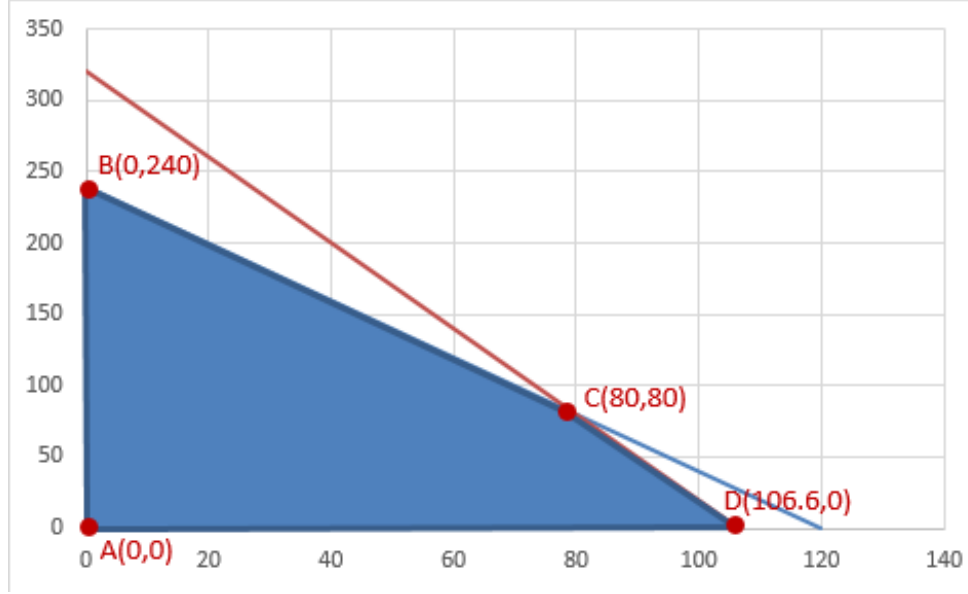
Sistem kısıtlarımızın grafiğini çizecek olursak:

Grafik 1.7: Soru 4'teki Sistem Kısıtlarının Koordinat Sistemindeki Görünümü



Sistem kısıtlarımız koordinat düzleminde Grafik 1.7'deki gibi görünecektir. Her iki kısıtta da “ \leq ” ibaresi olduğundan aşağı tarafa taranmalıdır. Bu açıklamaya göre uygun çözüm alanı ve köşe noktaları ise aşağıdaki gibi olacaktır:

Grafik 1.8: Soru 4'ün Uygun Çözüm Alanı ve Köşe Noktaları



Bu noktalarımızı amaç fonksiyonunda yerine koyacak olursak:

Tablo 1.4: Grafik 1.8'deki Köşe Noktalarının Amaç Fonksiyonundaki Değer Tablosu

Noktalar	$25.x_1 + 20.x_2$
(0,0)	0
(0,240)	4800
(80,80)	3600
(106.6,0)	2665

Tablo 1.4'te görüldüğü gibi $(x_1, x_2) = (0, 240)$ noktası bizim optimal çözümümüz olacaktır. Yani firma sadece B tip elbise modelinde 240 adet üretirse kısıtları sağlayacak şekilde maksimum kârı elde etmiş olur.

Soru-5:

Bu soru da ise optimal sonucu olmayan bir örnek ele alınmaktadır. Aşağıdaki doğrusal programlama problemini grafik yöntemiyle çözünüz.

$$\text{Max } Z = 6.x_1 + 3.x_2$$

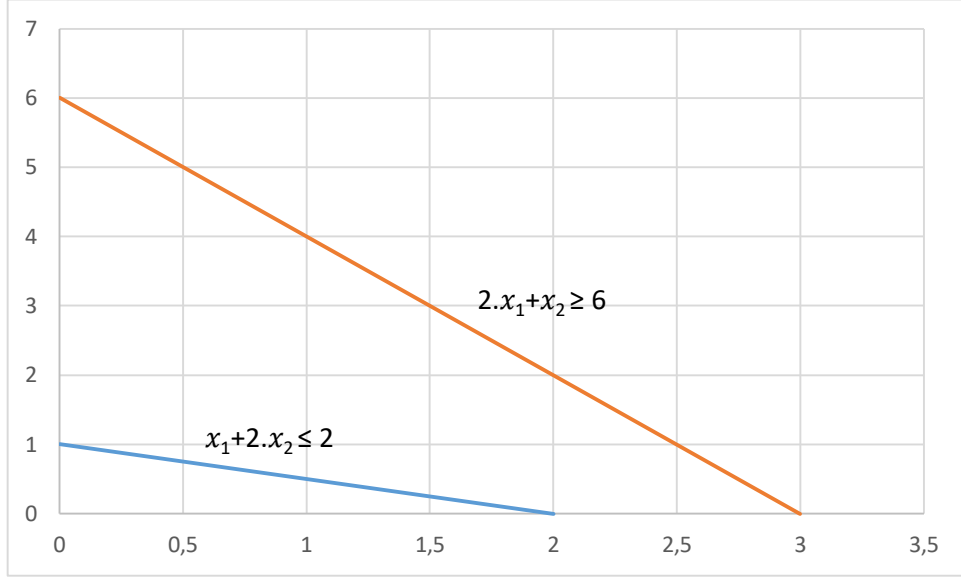
$$x_1 + 2.x_2 \leq 2$$

$$2.x_1 + x_2 \geq 6$$

$$x_1, x_2 \geq 0$$

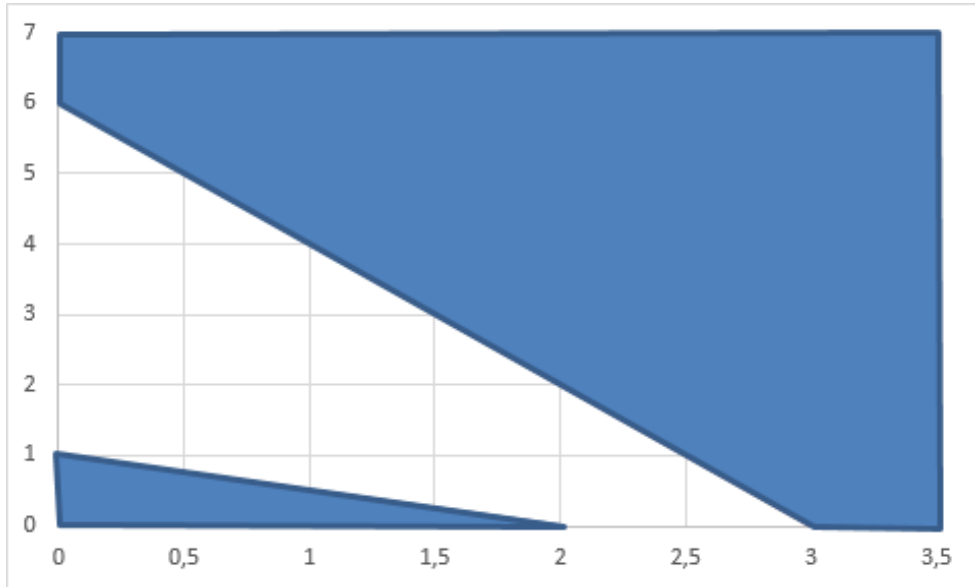
Bu sistem kısıtlarını ele aldığımızda grafiğimiz aşağıdaki gibi olacaktır:

Grafik 1.9: Soru 5'teki Sistem Kısıtlarının Koordinat Sistemindeki Görünümü



Uygun çözüm alanımız ise:

Grafik 1.10: Soru 5'nin Uygun Çözüm Alanı



Grafik 1.10'da görüldüğü gibi ortak çözüm alanı olmadığından bu sorunun uygun çözümleri ve optimal çözümü yoktur.

İKİNCİ BÖLÜM

2. MiniZinc'e İLE OPTİMİZASYON

2.1. MiniZinc Nedir ?

MiniZinc, tamsayılar ve gerçek sayılar üzerinde kısıtlı optimizasyon ve karar problemlerini belirlemek için kullanılan bir dildir. Bir MiniZinc modeli sorunun nasıl çözüleceğini belirtmez. MiniZinc derleyici, onu Kısıt Programlama (CP), Karışık Tamsayılı Doğrusal Programlama (MIP) veya Mantıksal İfadein Sağlanabilirliği (SAT) gibi çok çeşitli çözücüler için uygun farklı formlara dönüştürebilir.

MiniZinc dili, dizin kümeleri üzerinden toplamlar veya çıkarımlar, if-then-else ifadeleri ve mantıksal bağlaçlar gibi bilinen gösterimleri kullanarak modellerin, problemin matematiksel bir formülasyonuna yakın bir şekilde yazmasına izin verir. Ayrıca MiniZinc, kullanıcıların modellerini yapılandırmasına izin veren tahminleri ve işlevleri tanımlamayı destekler (normal programlama dillerindeki prosedürlere ve işlevlere benzer).

MiniZinc modelleri genellikle parametrikdir yani bir problem örneğini ele almaktan ziyade problemin tüm sınıfını tanımlar. Böylelikle bir araç yönlendirme problemi modeli, gelecek hafta için güncellenen müşteri talepleriyle somutlaştırılarak haftalık planlar oluşturmak için yeniden kullanılabilir.

MiniZinc, farklı arka plan derleyicilerine kolayca arayüz oluşturacak şekilde tasarlanmıştır. Bunu, MiniZinc modelini ve veri dosyasını FlatZinc modeline dönüştürerek yapar. Problem optimizasyon problemi ise FlatZinc modelleri nesnel fonksiyon tanımının yanı sıra değişken ifadelerinden ve kısıt tanımlarından oluşur. Ayrıca modele ait açıklamalar veya notlar modelin sunduğu anlamından bağımsız, kullanıcıya derleyicinin davranışına ince ayar yapmasına izin verir.

2.1.1. MiniZinc'in Özellikleri

MiniZinc yüksek seviyeli, çoğunlukla birinci dereceden işlevsel bir modelleme dilidir. MiniZinc:

- Matematiksel gösterime benzer söz dizimi (automatic coercions, overloading, iteration, sets, arrays)
- İfade kısıtlamaları (sonlu alan, küme, lineer aritmetik, tamsayı)
- Çeşitli sorunlara destek
- Verilerin modelden ayrılması
- Yüksek seviyeli veri yapıları (kümeler, diziler, numaralandırılmış türler)
- Genişletilebilirlik (kullanıcı tanımlı işlevler ve tahminler)
- Güvenilirlik (tip kontrolü, örnekleme kontrolü, iddialar)
- Çözücülerden bağımsız modelleme gibi özelliklere sahiptir

2.1.2. MinZinc’te Kullanılan Bazı Komutlar

Değişken Türleri	Açıklama	Genel Kullanım	Örnek
int	Diğer dillerde olduğu gibi tam sayı tanımlamak için kullanılabilir.	int: <var-name>	int: x1
var	Belirli sayılar aralığında değişken tanımlanmak için kullanılır.	var <i>.. <u>: <var-name><="" u=""></u>:>	var 1..100: x1
var int	Belirli sayılar arasında olmayan değişken tanımlanabilir.	var int: <var-name>	var int: x1
constraint	Modelimizdeki kısıtlarımızı tanımlamak için kullanılan değişken türüdür.	constraint $a * x_1 + b * x_2 + \dots +$ $p * x_n = c$ $>$ $<$	constraint $x1 + 2 * x2 \leq 12$
solve minimize	Minimum maliyet istendiğinde kullanılan değişken tipidir.	solve minimize $a * x_1 + b * x_2 + \dots + x_n$	solve minimize $50 * x1 + 60 * x2;$
solve maximize	Maximum kar istendiğinde kullanılan değişken tipidir.	solve maximize $a * x_1 + b * x_2 + \dots + x_n$	solve maximize $10 * x1 + 70 * x2$

output	Uygun çözümleri ekrana çıktı vermek için kullanılır.	output [<açıklama>\(parametre)\", . . .];	output["A makinası = \(x1)\n", "B makinası = \n(x2)\n"];
---------------	--	---	--

2.1.3. Bir Optimizasyon Modeline Genel Bakış

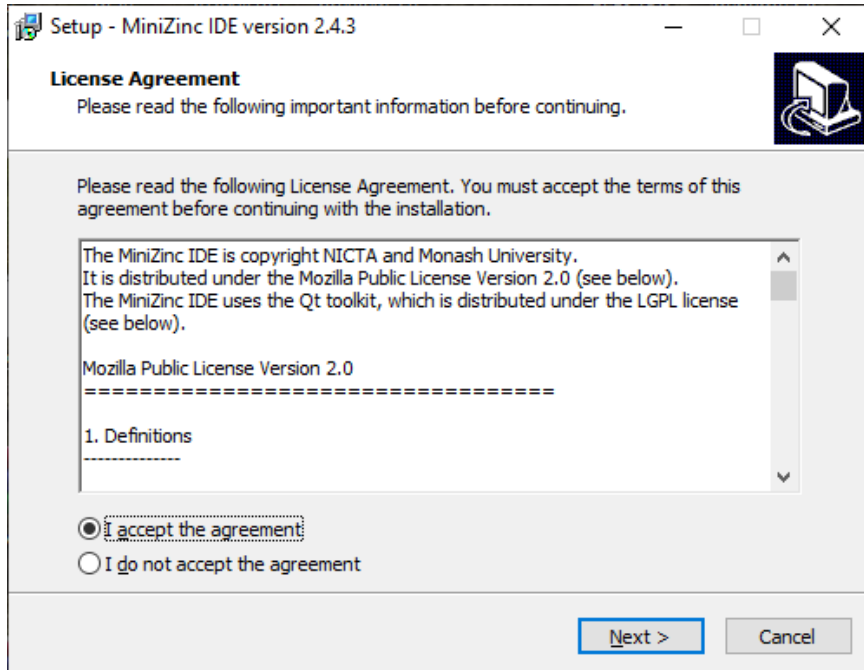
Kavramsal olarak, bir MiniZinc problemi iki bölümden oluşur:

- 1.Model: Problemin ana hatlarını tanımlar.
- 2.Ver: Model için girdileri tanımlar.

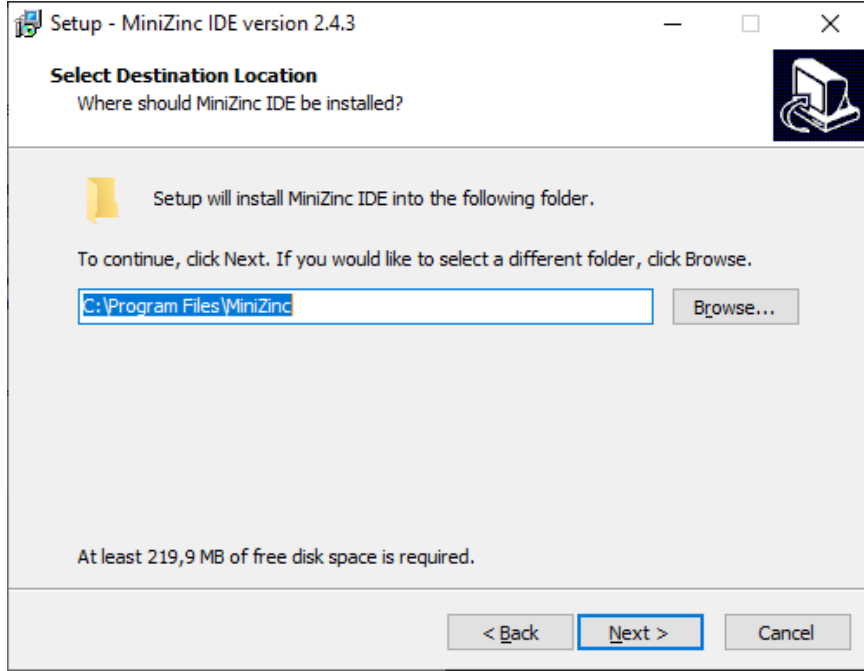
İki geniş problem sınıfı vardır: Uygunluk ve optimizasyon. Uygunluk problemlerinde tüm çözümler eşit derecede iyi kabul edilirken, optimizasyon problemlerinde çözümler amaca göre sıralanır. Amaç optimal çözümü bulmaktır.

2.2. Kurulum

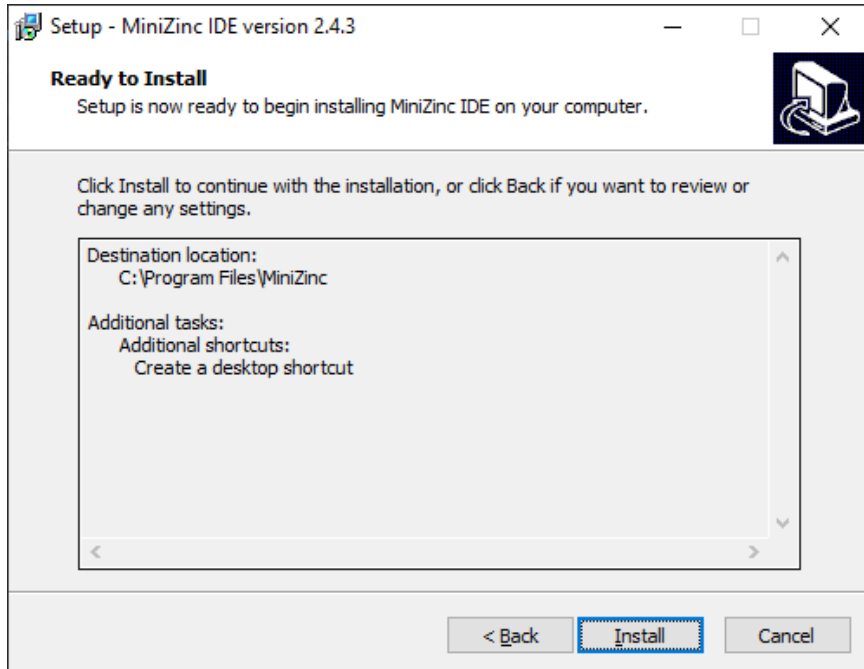
Bu bölümde Windows işletim sistemime MiniZinc programının kurulumu ele alınacaktır. MiniZinc IDE'yi <https://www.minizinc.org/software.html> adresinden indirilebilir. Adresini açtığımızda birçok işletim sistemine ait yükleme paketleri mevcuttur. 32 bit Windows yüklemeleri artık desteklenmemektedir bu sebeple 64bit mimariye sahip olan yükleme paketi indirilmelidir. Windows yükleyici paketini indirdikten ve kurmak için yükleme paketine çift tıkladıktan sonra ilk adım olan aşağıdaki pencere ile karşılaşacağız.



Karşımıza birçok yükleyicide olduğu gibi bazı kabul edilmesi gereken şartlar ve sözleşmeler mevcut bunları kabul etmek adına “I agree the agreement” seçeneğine tıkladıktan sonra “Next” ile bir sonraki adıma geçelim.



Burada ise varsayılan olarak MiniZinc programının kurulacağı dizini seçtikten sonra Next tuşuna tıklayarak aşağıdaki son adıma geçelim.



Görüldüğü gibi son adımda ise yükleme işlemi bittikten sonra kurulumu tamamlanmaktadır. Kurulumu tamamladıktan sonra .mzn, .dzn ve .fzn dosya uzantılarına sahip herhangi bir MiniZinc dosyasını çift tıklayarak IDE’de açabilirsiniz.

2.3. MiniZinc ile İlk Adımlar

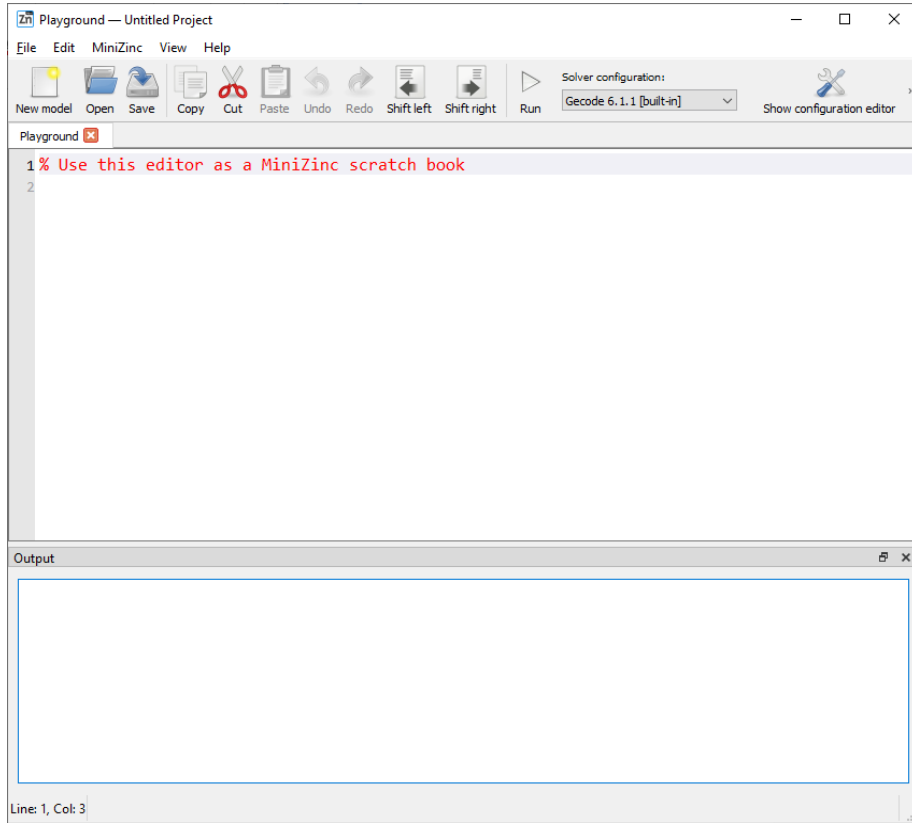
MiniZinc IDE, MiniZinc derleyicisini ve önceden yapılandırılmış birkaç çözücüyü içerir, böylece hemen başlayabilirsiniz. Bu bölümde, bazı çok temel örnekleri kullanarak MiniZinc IDE ve tanıtılmaktadır.

2.3.1. MiniZinc IDE

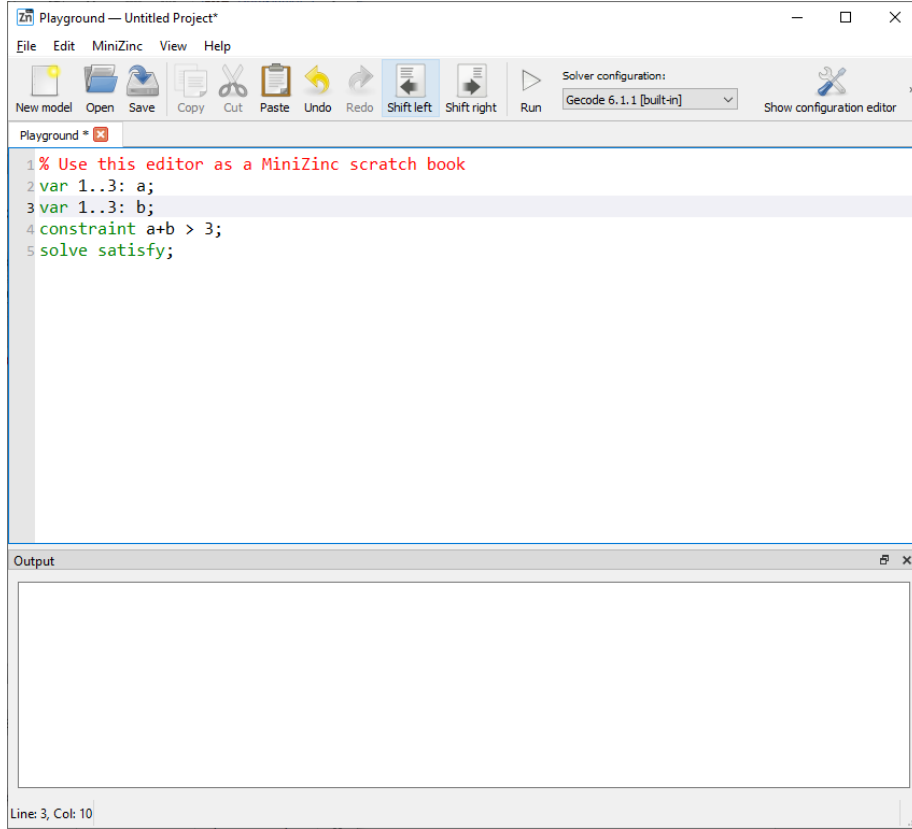
MiniZinc IDE, MiniZinc'in işlevselliğinin çoğuna basit bir arayüz sağlar. Model ve veri dosyalarını düzenlemenizi, MiniZinc tarafından desteklenen çözücülerden herhangi birini çözmenizi, hata ayıklama ve profil oluşturma araçlarını çalıştırmanızı ve çevrimiçi kurslara çözümler sunmanızı sağlar.

MiniZinc IDE'yi ilk kez açtığınızda, bir güncelleme olduğunda size bildirilmek isteyip istemediğinizi soracaktır. IDE'yi kaynaklardan yüklediyseniz, MiniZinc derleyicisinin kurulumunu bulmanızı isteyebilir.

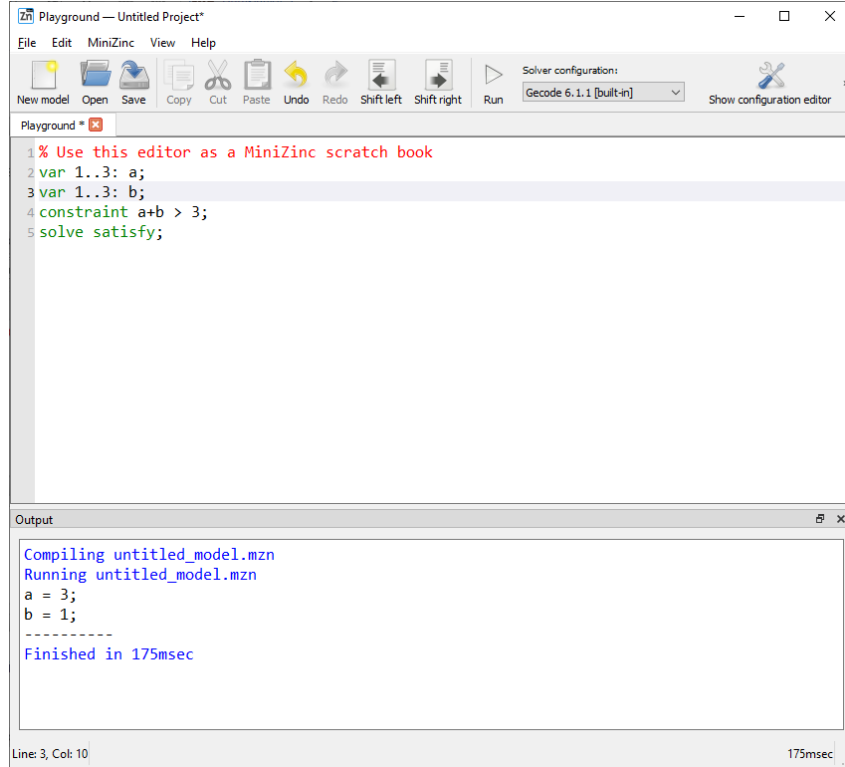
IDE sizi aşağıdaki gibi görünen bir pencere olan MiniZinc “PlayGround” ile karşılayacaktır.



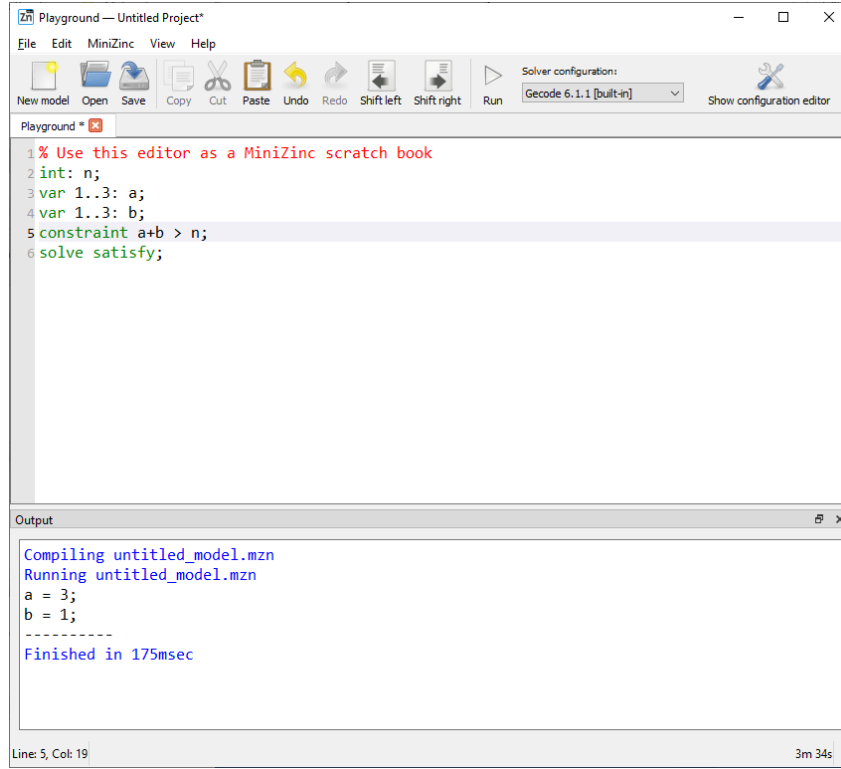
İlk MiniZinc modelinizi yazmaya başlayabilirsiniz! Çok basit bir şey deneyelim:



Modeli çözmek için araç çubuğundaki *Çalıştır* düğmesine tıklayın veya *Ctrl + R* klavye kısa yolunu kullanın (veya macOS'ta *+ R* komutu):



Gördüğünüz gibi, girdiğiniz soruna bir çözüm görüntüleyen bir çıkış penceresi açılır. Şimdi ek veri gerektiren bir model deneyelim.



The screenshot shows the Zn Playground IDE with the following content:

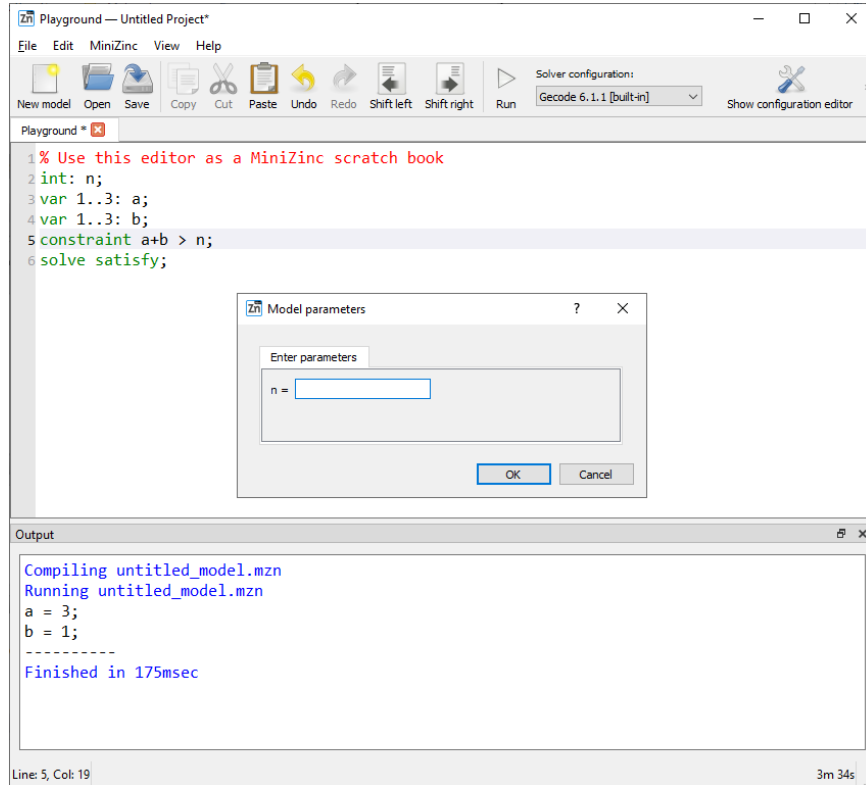
```
1 % Use this editor as a MiniZinc scratch book
2 int: n;
3 var 1..3: a;
4 var 1..3: b;
5 constraint a+b > n;
6 solve satisfy;
```

The Output window displays the following text:

```
Compiling untitled_model.mzn
Running untitled_model.mzn
a = 3;
b = 1;
-----
Finished in 175msec
```

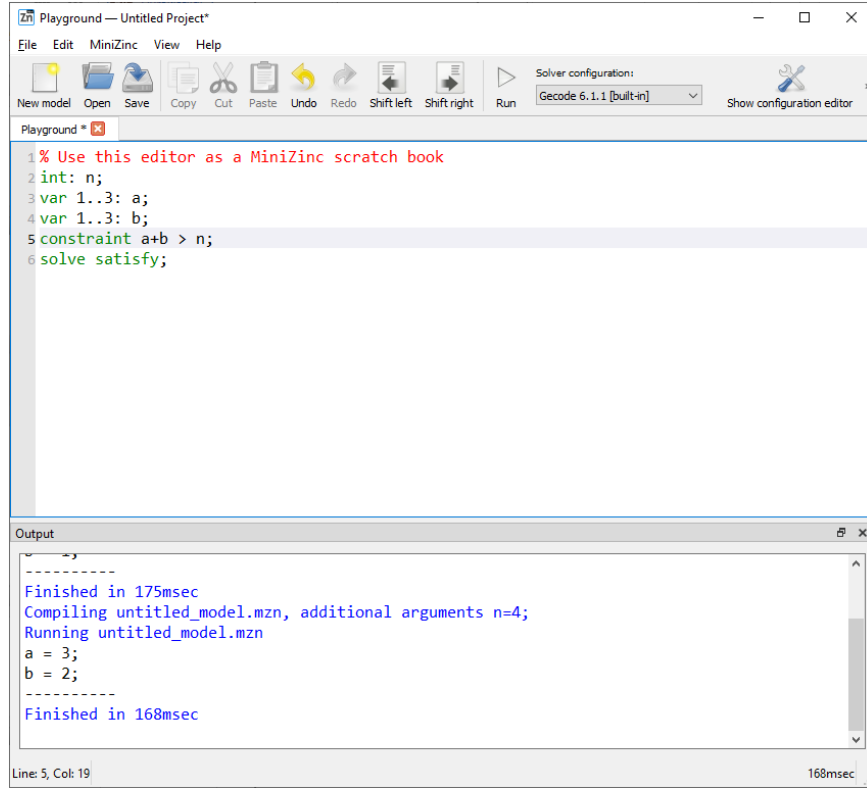
The status bar at the bottom indicates "Line: 5, Col: 19" and "3m 34s".

Bu modeli çalıştırdığımızda, IDE sizden n parametresi için bir değer girmenizi ister.



The screenshot shows the Zn Playground IDE with the same MiniZinc model as before. A "Model parameters" dialog box is open in the center, prompting the user to enter a value for n . The dialog box has a text input field labeled "n =" and buttons for "OK" and "Cancel". The Output window and status bar are the same as in the previous screenshot.

Örneğin n için 4 değerini girelim ve modelin çözümünü görmek için çalıştır düğmesine tıkladığımızda aşağıdaki gibi sonuçlar elde edilecektir.



The screenshot shows the MiniZinc Playground window titled "Playground — Untitled Project". The menu bar includes File, Edit, MiniZinc, View, and Help. The toolbar contains icons for New model, Open, Save, Copy, Cut, Paste, Undo, Redo, Shift left, Shift right, and Run. The Solver configuration is set to "Gecode 6.1.1 [built-in]". The main editor contains the following MiniZinc code:

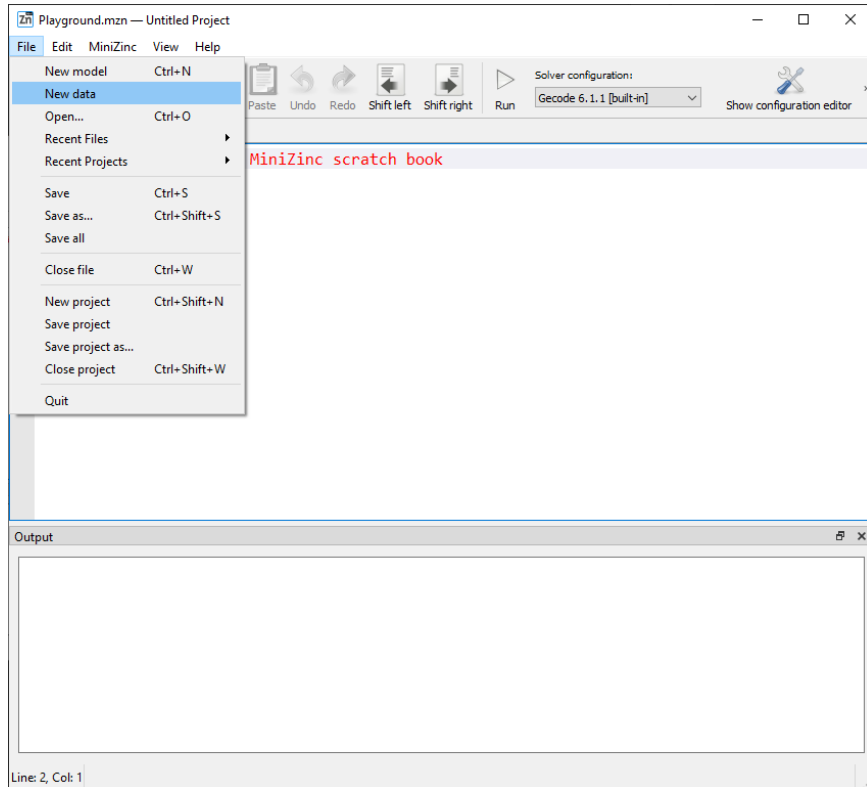
```
1 % Use this editor as a MiniZinc scratch book
2 int: n;
3 var 1..3: a;
4 var 1..3: b;
5 constraint a+b > n;
6 solve satisfy;
```

The Output window shows the following text:

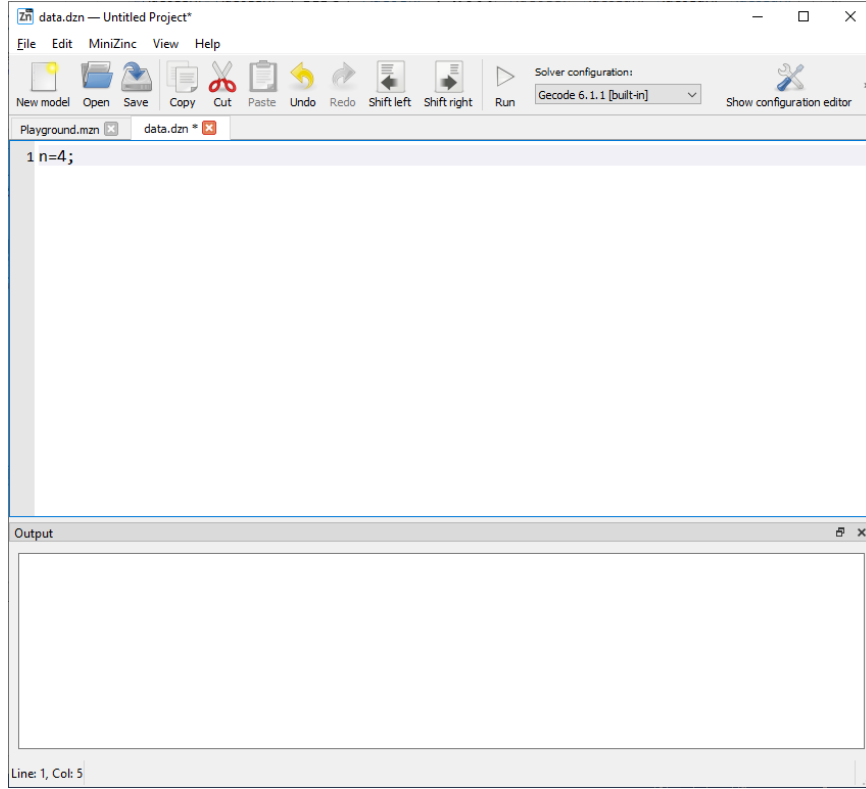
```
-----
Finished in 175msec
Compiling untitled_model.mzn, additional arguments n=4;
Running untitled_model.mzn
a = 3;
b = 2;
-----
Finished in 168msec
```

The status bar at the bottom indicates "Line: 5, Col: 19" and "168msec".

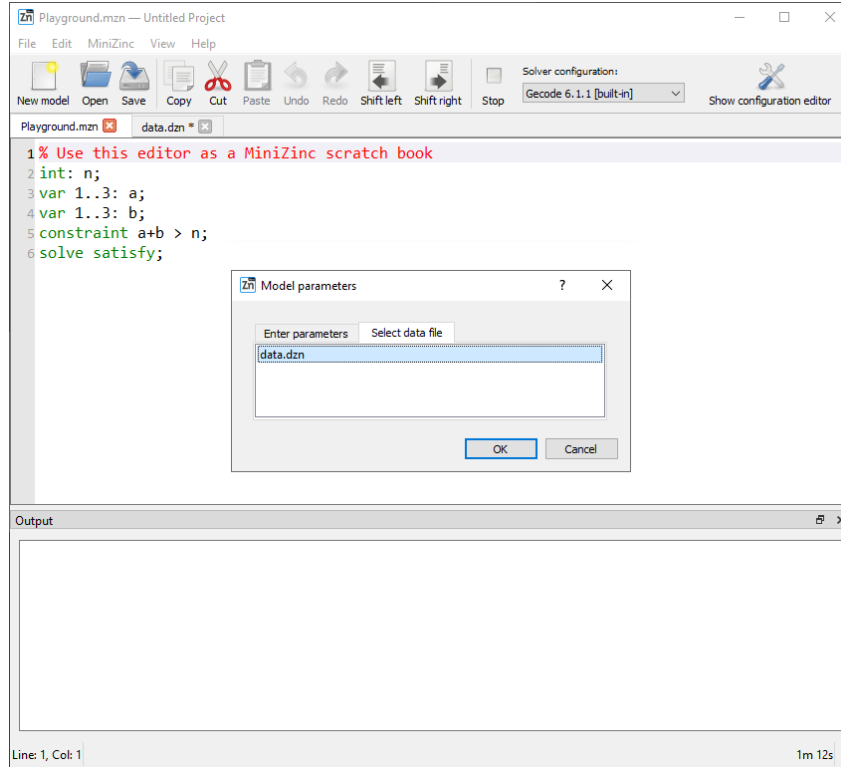
Ayrıca n için verimizi bir dosyadan aktarabiliriz. Bunun için aşağıda ki resim de görüldüğü gibi File-New data ile veri dosyamızı oluşturalım.



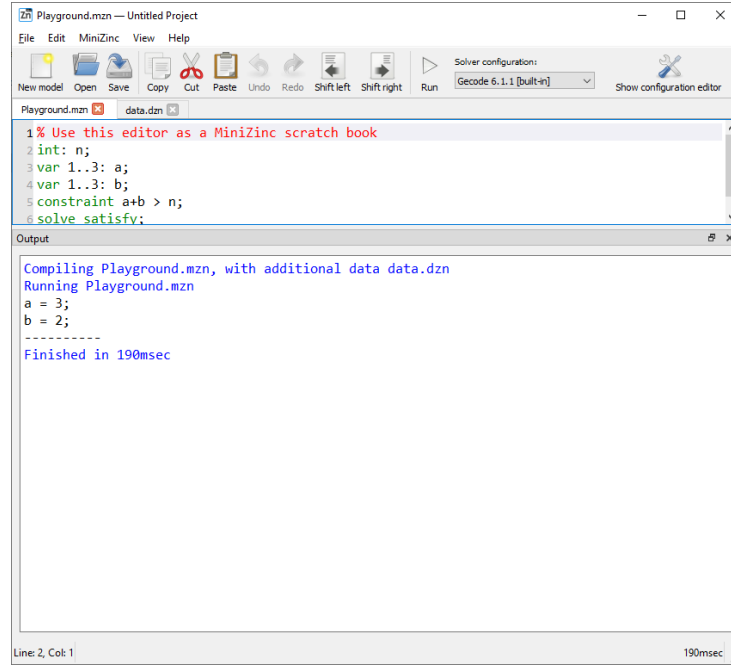
Oluşturduktan sonra n değerini 4 verelim.



Son olarak n değerimizi girdikten sonra Playground sekmesinde “Run” tuşuna tıkladığımızda, IDE size veri seçme seçeneği sunar.



Bu pencerede verimizi barındıran “data” adlı dosyayı seçelim ve modelimizin a ve b’nin sonuçlarını görmek için “OK” tuşuna tıklayalım.



Görüldüğü gibi a için 3 ve b için 2 değerleri elde edildi. Elbette oluşturduğunuz modelinizi bir dosyaya kaydedebilir ve bir dosyadan yükleyebilirsiniz.

ÜÇÜNCÜ BÖLÜM

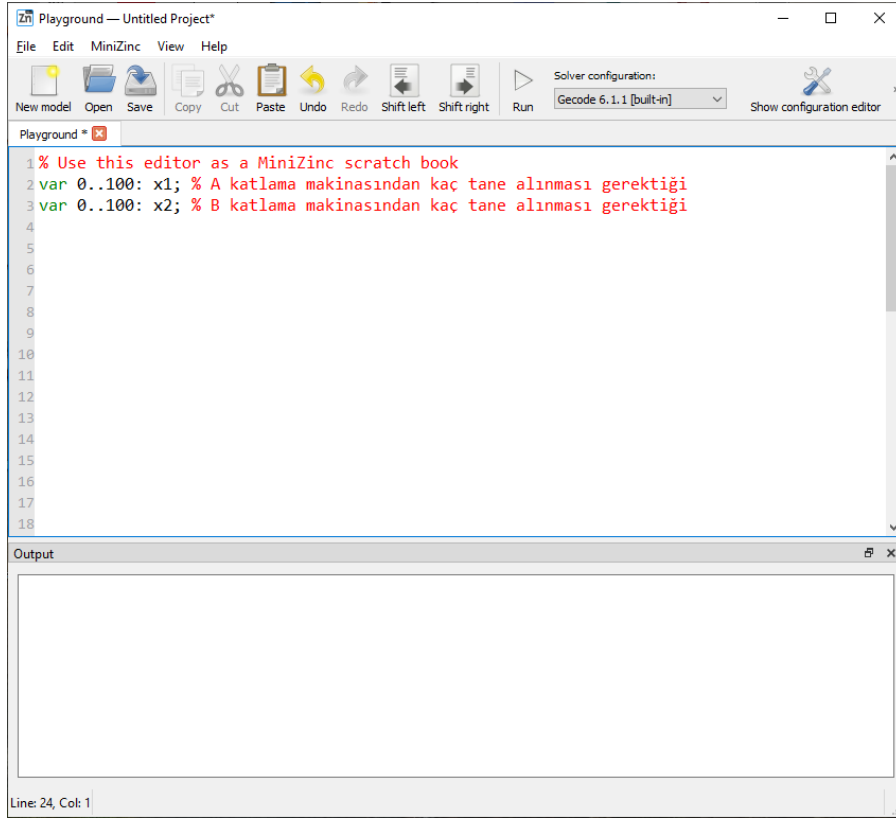
3. MiniZinc İLE UYGULAMALAR

3.1. Optimizasyon Problemlerinin Minizinc'e Aktarılması ve Çözümü

Bu bölümde daha önce 4 adet ele aldığımız optimizasyon problemlerini kullanarak bu problemlerin MiniZinc'e nasıl aktarıldığı ve modelin temel yapısı anlatılmaktadır.

Soru 1'in MiniZinc ile Çözümü:

İlk örneğimizden hareketle iki farklı tipte karton katlama makinası mevcuttur. x_1 , A katlama makinasından kaç tane alınması gerektiğini; x_2 ise B katlama makinasından kaç tane alınması gerektiğini temsil etmektedir. Bu değişkenleri MiniZinc'te tanımlayalım:



Soru-1'deki teknolojik(sistem) kısıtlar:

$$\begin{aligned} 30 \cdot x_1 + 50 \cdot x_2 &\geq 320 \\ x_1 + 2 \cdot x_2 &\leq 12 \end{aligned}$$

Bu teknolojik kısıtları ekledikten sonra MiniZinc'teki görünüm aşağıdaki gibi olacaktır:

The screenshot shows the MiniZinc Playground window with the following code in the editor:

```
1 % Use this editor as a MiniZinc scratch book
2 var 0..100: x1; % A katlama makinasından kaç tane alınması gerektiği
3 var 0..100: x2; % B katlama makinasından kaç tane alınması gerektiği
4
5 % 320 adet kutu kısıtı
6 constraint 30*x1+50*x2 >= 320;
7
8 % işçi kısıtı
9 constraint 1*x1+2*x2 <=12;
10
11
12
13
14
15
16
17
18
```

The output window is empty. The status bar at the bottom indicates "Line: 21, Col: 1".

Görüldüğü gibi kısıtlarımızı “constraint” komutu ile tanımlayabiliriz.

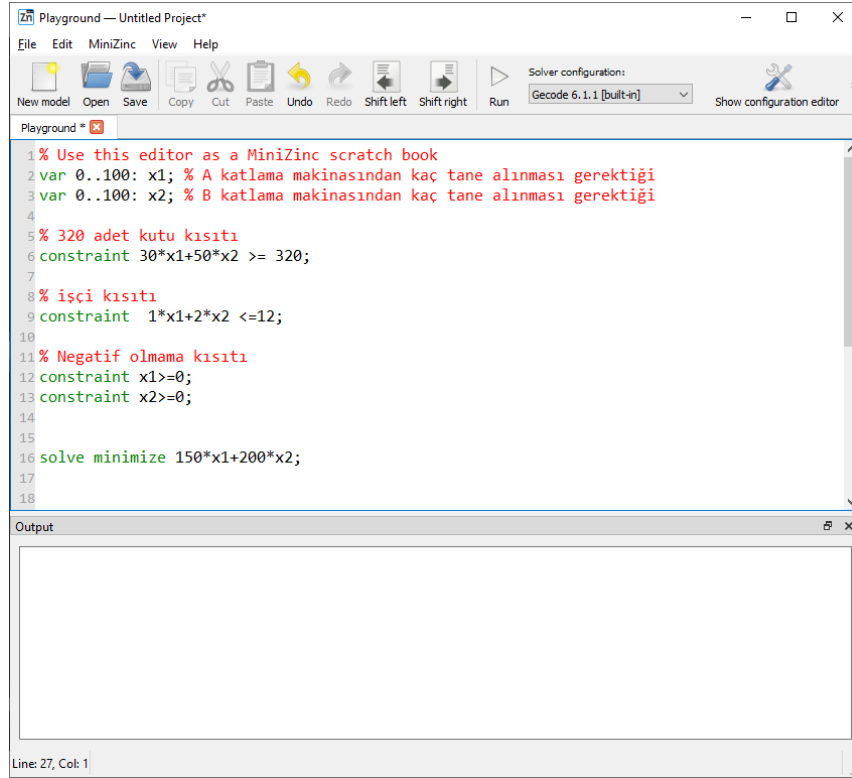
İfadelerimiz sıfırdan büyük olacağından -yani üretmemiz gereken miktar negatif olamayacağından- negatif olmama kısıtımız aşağıdaki gibi yazılmalıdır:

The screenshot shows the MiniZinc Playground window with the following code in the editor:

```
1 % Use this editor as a MiniZinc scratch book
2 var 0..100: x1; % A katlama makinasından kaç tane alınması gerektiği
3 var 0..100: x2; % B katlama makinasından kaç tane alınması gerektiği
4
5 % 320 adet kutu kısıtı
6 constraint 30*x1+50*x2 >= 320;
7
8 % işçi kısıtı
9 constraint 1*x1+2*x2 <=12;
10
11 % Negatif olmama kısıtı
12 constraint x1>=0;
13 constraint x2>=0;
14
15
16
17
18
```

The output window is empty. The status bar at the bottom indicates "Line: 24, Col: 1".

Söz konusu Soru-1'deki amaç fonksiyonumuz $Min Z = 150 \cdot x_1 + 200 \cdot x_2$ olarak tanımlanmıştı. Bu amaç fonksiyonumuzu MiniZinc'e aktaralım.

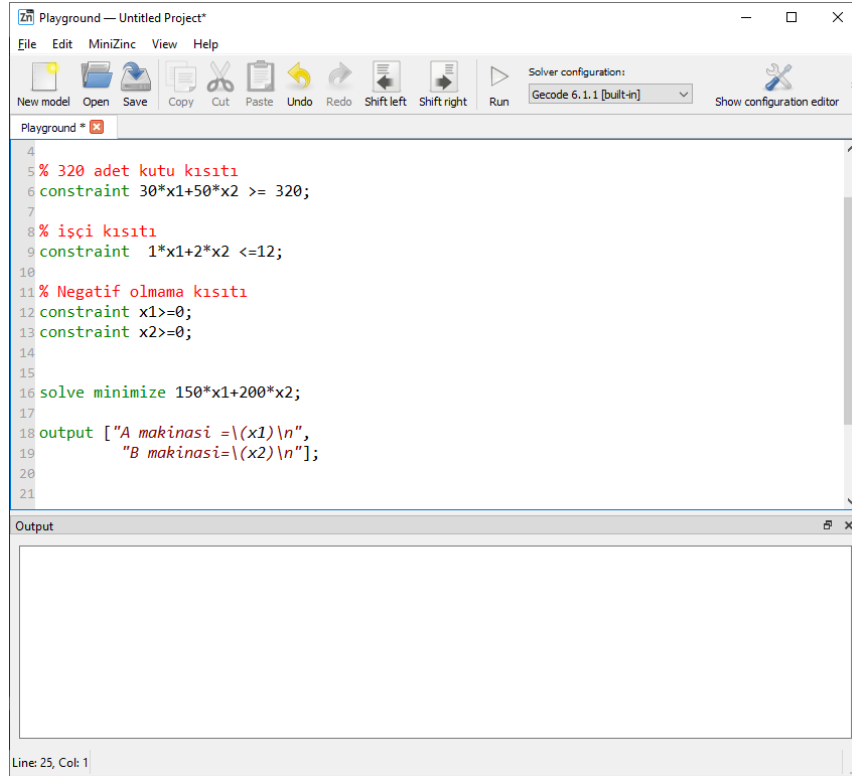


The screenshot shows the MiniZinc Playground window with the following code in the editor:

```
1 % Use this editor as a MiniZinc scratch book
2 var 0..100: x1; % A katlama makinasından kaç tane alınması gerektiği
3 var 0..100: x2; % B katlama makinasından kaç tane alınması gerektiği
4
5 % 320 adet kutu kısıtı
6 constraint 30*x1+50*x2 >= 320;
7
8 % işçi kısıtı
9 constraint 1*x1+2*x2 <=12;
10
11 % Negatif olmama kısıtı
12 constraint x1>=0;
13 constraint x2>=0;
14
15
16 solve minimize 150*x1+200*x2;
17
18
```

The Output panel is empty. The status bar at the bottom indicates "Line: 27, Col: 1".

Çıktılarımızı yazdırabilmek için ise aşağıdaki komut eklenmelidir:

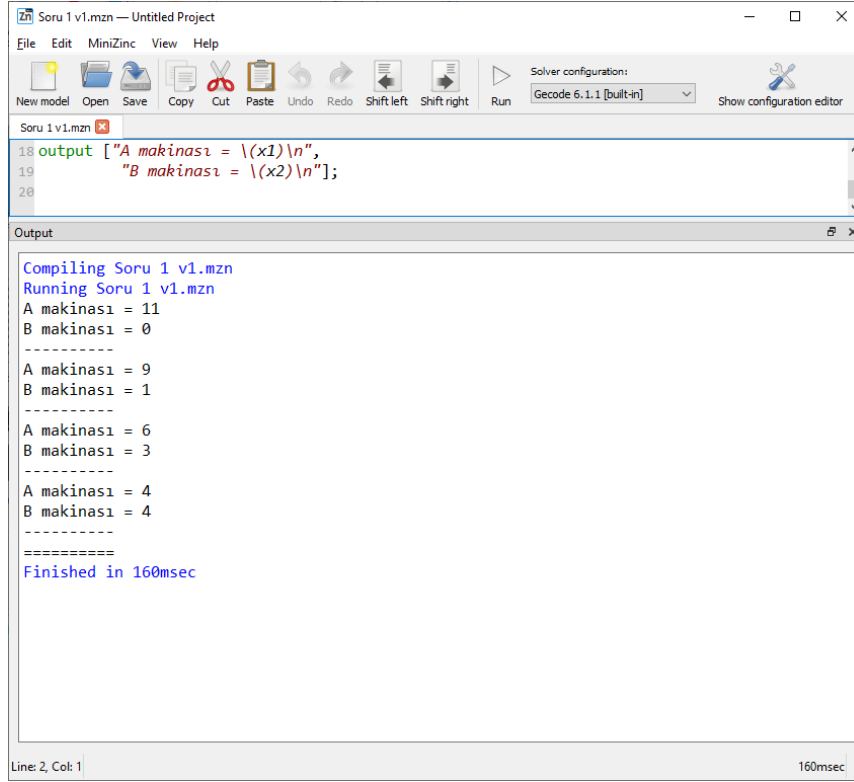


The screenshot shows the MiniZinc Playground window with the following code in the editor:

```
4
5 % 320 adet kutu kısıtı
6 constraint 30*x1+50*x2 >= 320;
7
8 % işçi kısıtı
9 constraint 1*x1+2*x2 <=12;
10
11 % Negatif olmama kısıtı
12 constraint x1>=0;
13 constraint x2>=0;
14
15
16 solve minimize 150*x1+200*x2;
17
18 output [ "A makinesi =\ (x1)\n",
19          "B makinesi=\ (x2)\n"];
20
21
```

The Output panel is empty. The status bar at the bottom indicates "Line: 25, Col: 1".

Uygun çözümlerimiz ise aşağıda listelenmiştir:



The screenshot shows the MiniZinc IDE interface. The top menu bar includes File, Edit, MiniZinc, View, and Help. Below the menu is a toolbar with icons for New model, Open, Save, Copy, Cut, Paste, Undo, Redo, Shift left, Shift right, Run, and Solver configuration. The Solver configuration dropdown is set to 'Gecode 6.1.1 [built-in]'. The main editor window displays the source code for 'Soru 1 v1.mzn' with the following content:

```
18 output ["A makinası = \"(x1)\\n",
19         "B makinası = \"(x2)\\n"];
20
```

Below the editor is the Output window, which shows the following text:

```
Compiling Soru 1 v1.mzn
Running Soru 1 v1.mzn
A makinası = 11
B makinası = 0
-----
A makinası = 9
B makinası = 1
-----
A makinası = 6
B makinası = 3
-----
A makinası = 4
B makinası = 4
=====
Finished in 160msec
```

The status bar at the bottom indicates 'Line: 2, Col: 1' and '160msec'.

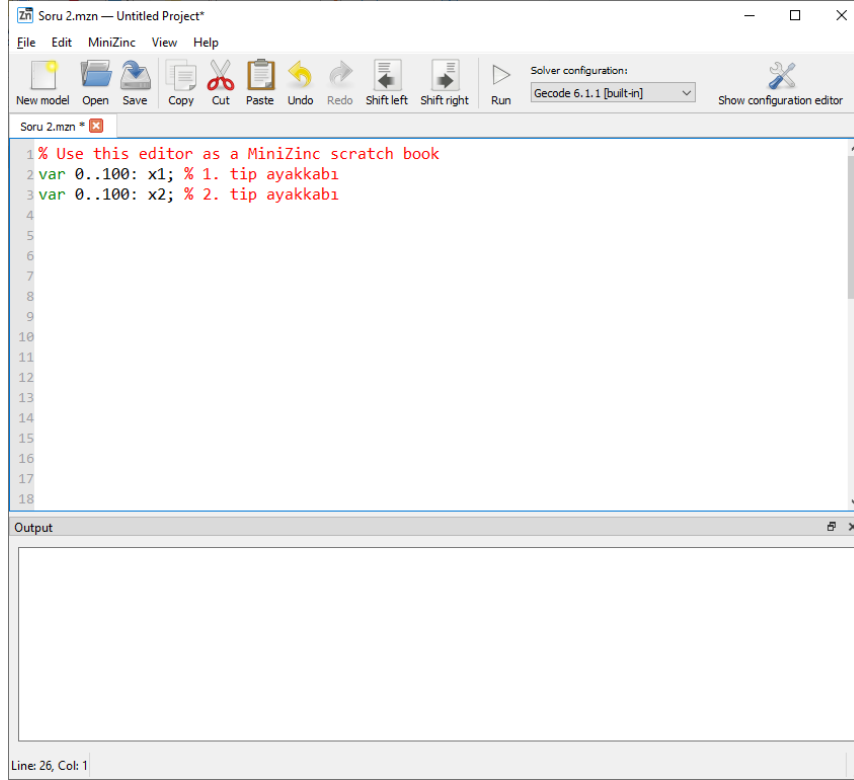
Grafik-1’den hareketle bu noktaların grafik yöntemiyle çözdüğümüz noktalar ile birebir aynı olmamasının sebebi MiniZinc programının uygun çözüm alanı içerisini taramasından kaynaklanmaktadır. Program, bir uç noktadan başlayıp alan içerisini taramaya başlar. Daha sonra optimal çözümü bulduğu anda diğer noktalara bakmadan en son sonucu ekranda gösterir. Çünkü program sadece uç noktalarla ilgilenmemektedir. Bizim optimal çözümümüz o alan içinde olduğu için program da amacına uygun olarak hareket etmektedir. Oysaki grafik yöntemi, yalnız bulduğu uç noktaları ele almaktadır. Zaten bizim burada odaklanmamız gereken nokta da optimal çözümdür.

Çıktı ekranında görüldüğü gibi uygun çözümler “-----“ ile ayrılmıştır. “=====” ifadesi programın optimal çözüme ulaştığını belirtmektedir. Buna göre optimal çözümümüz

$(x_1, x_2) = (4, 4)$ olarak bulunmuştur.

Soru-2'nin MiniZinc ile Çözümü:

2. örneğimizde ise iki tip ayakkabı mevcuttur. x_1 , 1. tip ayakkabıyı; x_2 ise 2. tip ayakkabıyı temsil etmektedir. Bu değişkenleri MiniZinc'te tanımlayacak olursak:



Soru-2'deki genel kısıtlarımız:

$$2.x_1 + 3.x_2 \leq 30$$

$$3.x_1 + 2.x_2 \leq 40$$

$$x_1, x_2 \geq 0$$

Bu kısıtları ekledikten sonra MiniZinc'teki görünüm aşağıdaki gibi olacaktır:


```
1 % Use this editor as a MiniZinc scratch book
2 var 0..100: x1; % 1. tip ayakkabı
3 var 0..100: x2; % 2. tip ayakkabı
4
5
6 % Zaman Kısıtı
7 constraint 2*x1 + 3*x2 <= 30;
8
9 % Malzeme Kısıtı
10 constraint 3*x1 + 2*x2 <= 40;
11
12 % Negatif Olmama Kısıtı
13 constraint x1 >= 0; % x1 için negatif olmama kısıtı
14 constraint x2 >= 0; % x2 için negatif olmama kısıtı
15
16
17
18
```

Output

Line: 21, Col: 1

Soru-2'deki amaç fonksiyonumuz $Max Z = 4.x_1 + 6.x_2$ olarak tanımlanmıştı. Bu amaç fonksiyonumuzu MiniZinc'e aktaralım ve çıktı komutumuzu ekleyelim:

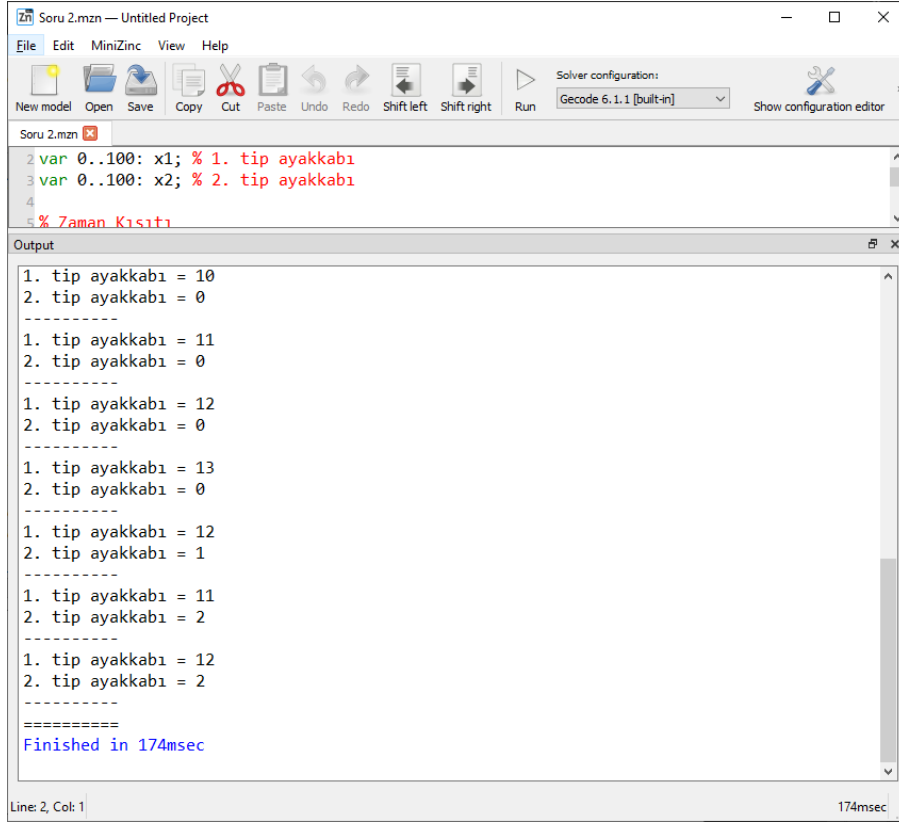
```
2 var 0..100: x1; % 1. tip ayakkabı
3 var 0..100: x2; % 2. tip ayakkabı
4
5 % Zaman Kısıtı
6 constraint 2*x1 + 3*x2 <= 30;
7
8 % Malzeme Kısıtı
9 constraint 3*x1 + 2*x2 <= 40;
10
11 % Negatif Olmama Kısıtı
12 constraint x1 >= 0; % x1 için negatif olmama kısıtı
13 constraint x2 >= 0; % x2 için negatif olmama kısıtı
14
15 solve maximize 4*x1 + 6*x2;
16
17 output ["1. tip ayakkabı = \"(x1)\\n\",
18        "2. tip ayakkabı = \"(x2)\\n\""];

```

Output

Line: 2, Col: 1

Elde ettiğimiz uygun çözümlerimiz ise aşağıdaki gibi olacaktır:



```
Soru 2.mzn — Untitled Project
File Edit MiniZinc View Help
New model Open Save Copy Cut Paste Undo Redo Shift left Shift right Run Solver configuration: Gecode 6.1.1 [built-in] Show configuration editor

2 var 0..100: x1; % 1. tip ayakkabı
3 var 0..100: x2; % 2. tip ayakkabı
4
5 % Zaman Kısıtı

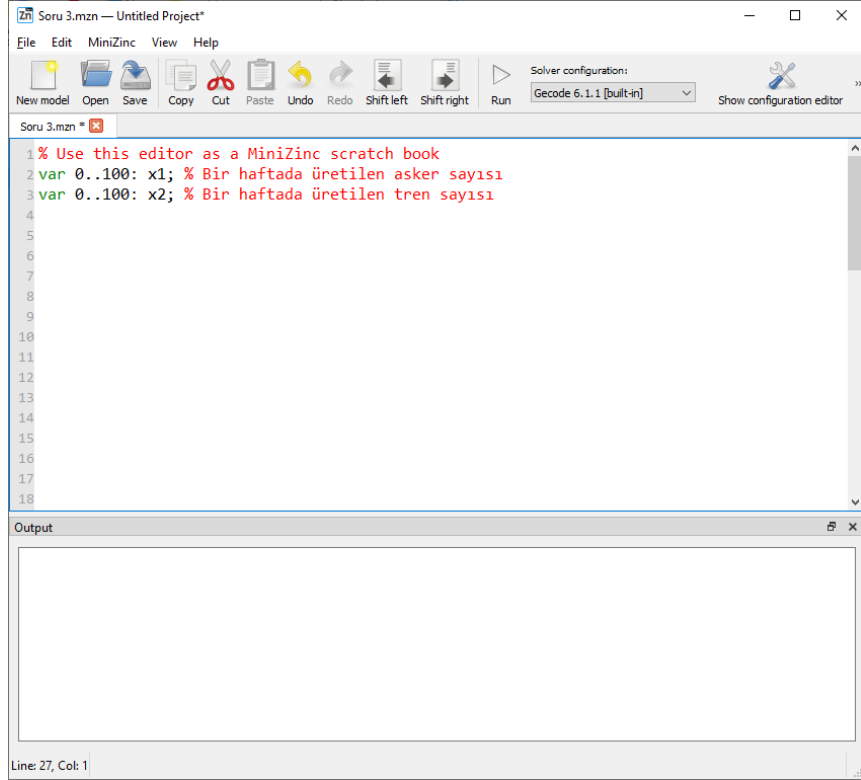
Output
1. tip ayakkabı = 10
2. tip ayakkabı = 0
-----
1. tip ayakkabı = 11
2. tip ayakkabı = 0
-----
1. tip ayakkabı = 12
2. tip ayakkabı = 0
-----
1. tip ayakkabı = 13
2. tip ayakkabı = 0
-----
1. tip ayakkabı = 12
2. tip ayakkabı = 1
-----
1. tip ayakkabı = 11
2. tip ayakkabı = 2
-----
1. tip ayakkabı = 12
2. tip ayakkabı = 2
-----
=====
Finished in 174msec

Line: 2, Col: 1 174msec
```

Görüldüğü gibi yalnız $(x_1, x_2) = (12, 2)$ sonucu bulunmuştur. Bunun sebebi Soru-1'de açıkladığımız gibi uygun çözüm alanının tamamını taramamasından kaynaklanmaktadır.

Soru-3'ün MiniZinc ile Çözümü:

3. örneğimizde Giapetto oyuncak asker ve oyuncak tren yapmak istemektedir. x_1 , bir haftada üretilen asker sayısını; x_2 ise bir haftada üretilen tren sayısını temsil etmektedir. Bu değişkenleri programa ekleyecek olursak:



Soru-3'deki genel kısıtlarımız:

$$2. x_1 + x_2 \leq 100$$

$$x_1 + x_2 \leq 80$$

$$x_1 \leq 40$$

$$x_1, x_2 \geq 0$$

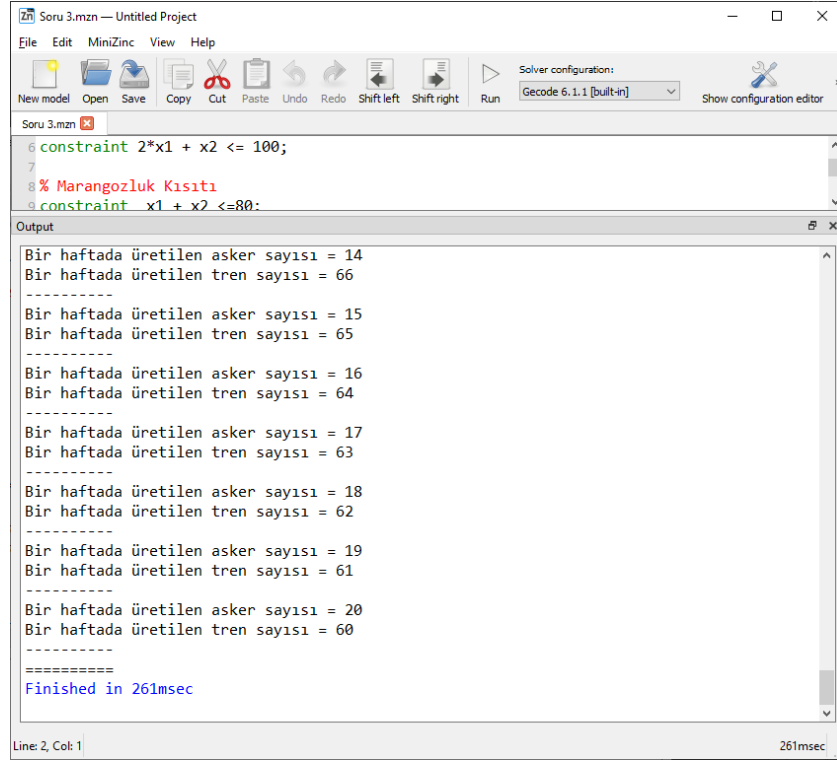
Bu kısıtlarımızın MiniZinc'teki görünümü aşağıdaki gibi olacaktır:

```
1 % Use this editor as a MiniZinc scratch book
2 var 0..100: x1; % Bir haftada üretilen asker sayısı
3 var 0..100: x2; % Bir haftada üretilen tren sayısı
4
5 % Montaj Kısıtı
6 constraint 2*x1 + x2 <= 100;
7
8 % Marangozluk Kısıtı
9 constraint x1 + x2 <= 80;
10
11 % Talep Kısıtı
12 constraint x1 <= 40;
13
14 % Negatif Olmama Kısıtı
15 constraint x1 >= 0; % x1 için negatif olmama kısıtı
16 constraint x2 >= 0; % x2 için negatif olmama kısıtı
17
18
```

Soru-3'deki amaç fonksiyonumuz $Max Z = 3.x_1 + 2.x_2$ olarak tanımlanmıştı. Bu amaç fonksiyonumuzu MiniZinc'e aktaralım ve çıktı komutunu ekleyelim:

```
7
8 % Marangozluk Kısıtı
9 constraint x1 + x2 <= 80;
10
11 % Talep Kısıtı
12 constraint x1 <= 40;
13
14 % Negatif Olmama Kısıtı
15 constraint x1 >= 0; % x1 için negatif olmama kısıtı
16 constraint x2 >= 0; % x2 için negatif olmama kısıtı
17
18 solve maximize 3*x1 + 2*x2;
19
20 output ["Bir haftada üretilen asker sayısı = \ (x1) \n",
21         "Bir haftada üretilen tren sayısı = \ (x2) \n"];
22
23
24
```

Uygun çözümlerimiz ise:



The screenshot shows the MiniZinc IDE interface. The main editor window displays a MiniZinc model with the following code:

```
6 constraint 2*x1 + x2 <= 100;  
7  
8 % Marangozluk Kısıtı  
9 constraint x1 + x2 <= 80;
```

The output window shows the results of the solver, listing the number of soldiers and trains produced each week for different values of x_1 and x_2 :

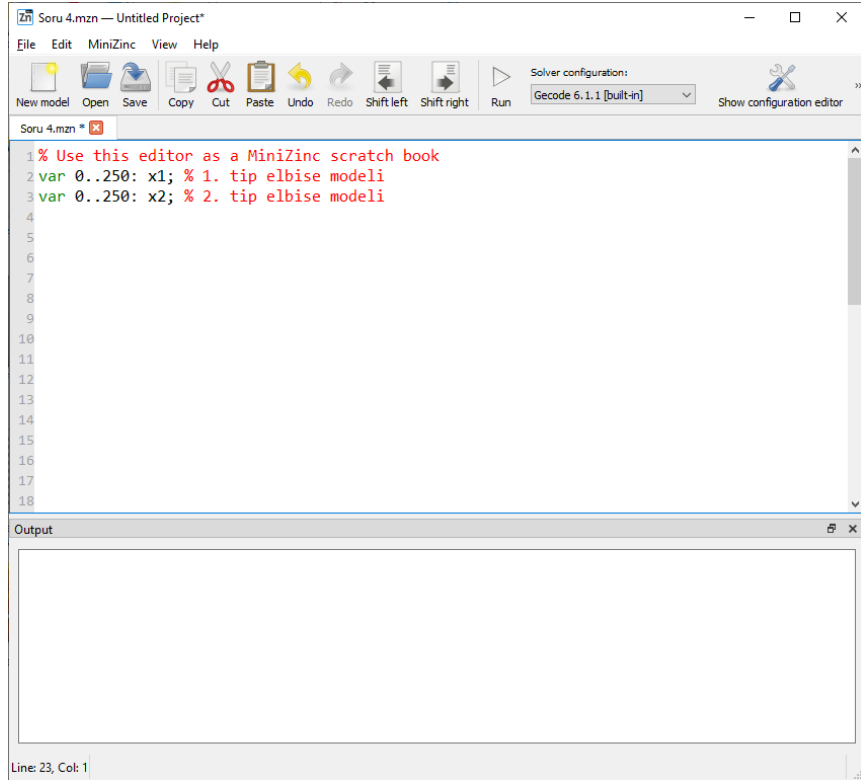
```
Bir haftada üretilen asker sayısı = 14  
Bir haftada üretilen tren sayısı = 66  
-----  
Bir haftada üretilen asker sayısı = 15  
Bir haftada üretilen tren sayısı = 65  
-----  
Bir haftada üretilen asker sayısı = 16  
Bir haftada üretilen tren sayısı = 64  
-----  
Bir haftada üretilen asker sayısı = 17  
Bir haftada üretilen tren sayısı = 63  
-----  
Bir haftada üretilen asker sayısı = 18  
Bir haftada üretilen tren sayısı = 62  
-----  
Bir haftada üretilen asker sayısı = 19  
Bir haftada üretilen tren sayısı = 61  
-----  
Bir haftada üretilen asker sayısı = 20  
Bir haftada üretilen tren sayısı = 60  
-----  
=====
```

The output concludes with "Finished in 261msec".

Görüldüğü üzere optimal çözümümüzü grafik yöntemiyle yaptığımız şekilde $(x_1, x_2) = (20, 60)$ olarak bulmuştur.

Soru-4'ün MiniZinc ile Çözümü:

Son sorumuzda ise bir elbise dikim firması, fabrika işçileri için iki tip üniforma üretmek istemektedir. x_1 , 1. tip elbiseyi; x_2 ise 2. tip elbiseyi belirtmektedir. Bu değişkenleri MiniZinc programına ekleyelim:



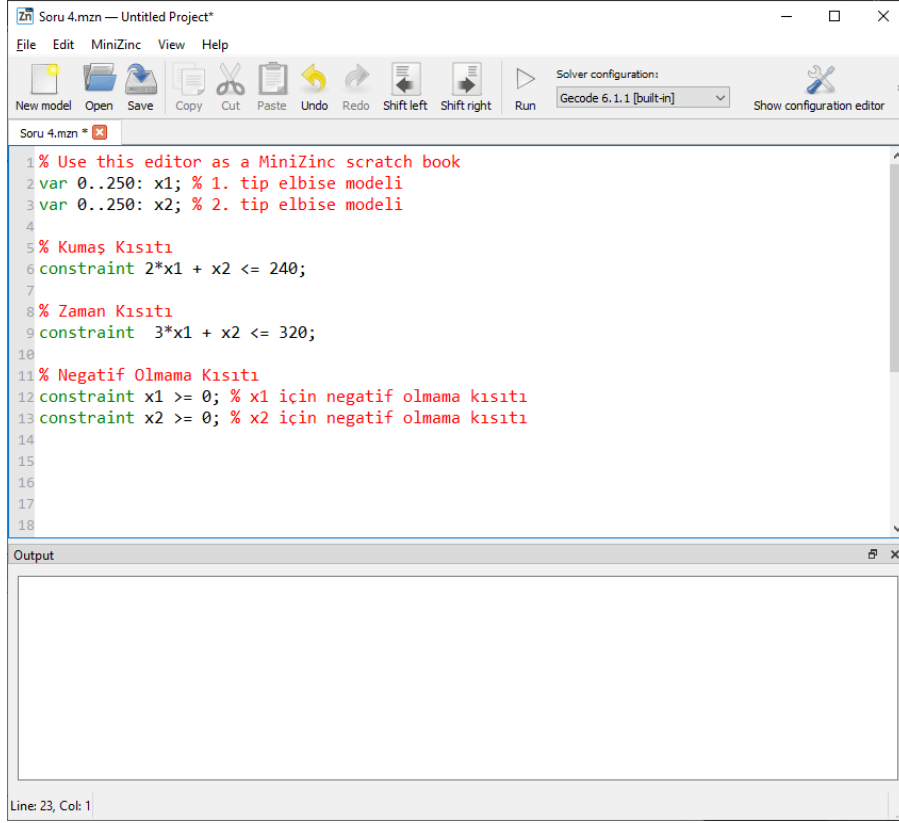
Soru-4'deki genel kısıtlarımız:

$$2. x_1 + x_2 \leq 240$$

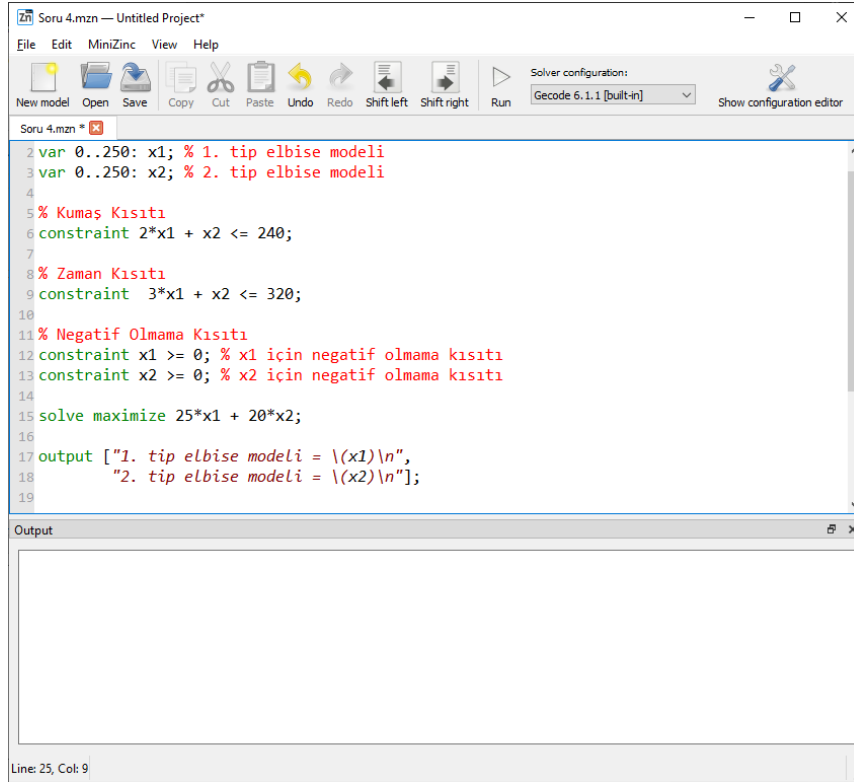
$$3. x_1 + x_2 \leq 320$$

$$x_1, x_2 \geq 0$$

Bu kısıtlarımızı da MiniZinc programına ekleyelim:



Soru-4'deki amaç fonksiyonumuz $Max Z = 25.x_1 + 20.x_2$ 'dir. Bu amaç fonksiyonumuzu da MiniZinc'e aktaralım ve çıktı komutumuzu ekleyelim:



Uygun çözümlerimiz ise aşağıdaki gibi olacaktır:

```

Soru 4.mzn
1 % Use this editor as a MiniZinc scratch book
2 var 0..250: x1; % 1. tip elbise modeli
3 var 0..250: x2; % 2. tip elbise modeli

Output
-----
1. tip elbise modeli = 0
2. tip elbise modeli = 97
-----
1. tip elbise modeli = 0
2. tip elbise modeli = 98
-----
1. tip elbise modeli = 0
2. tip elbise modeli = 99
-----
1. tip elbise modeli = 0
2. tip elbise modeli = 100
-----
[ 99 more solutions ]
1. tip elbise modeli = 0
2. tip elbise modeli = 200
-----
[ 39 more solutions ]
1. tip elbise modeli = 0
2. tip elbise modeli = 240
-----
=====
Finished in 268msec

```

Son sorumuz olan Soru-4'ün optimal sonucu ise $(x_1, x_2) = (0, 240)$ olarak bulunmuştur.

Soru-5'in MiniZinc ile Çözümü:

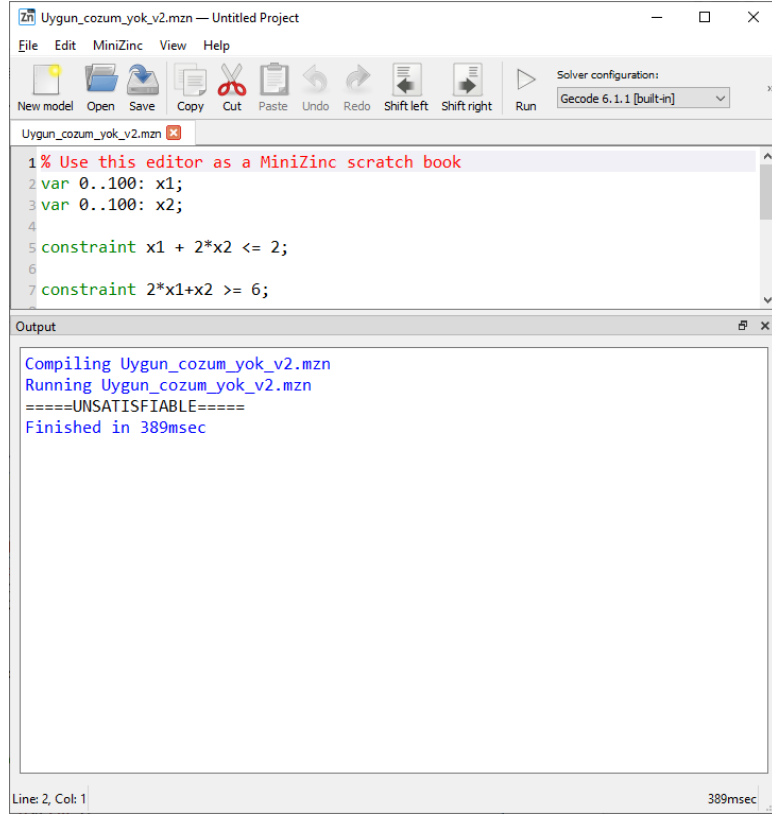
Soru-5'te yer alan değişkenlerimizi, kısıtlarımızı ve amaç fonksiyonumuzu ve MiniZinc'e aktaralım ve çıktıyı gösterecek komutumuzu(output) ekleyelim:

```

Uygun_cozum_yok_v2.mzn
1 % Use this editor as a MiniZinc scratch book
2 var 0..100: x1;
3 var 0..100: x2;
4
5 constraint x1 + 2*x2 <= 2;
6
7 constraint 2*x1+x2 >= 6;
8
9 constraint x1 >= 0;
10 constraint x2 >= 0;
11
12 solve minimize 6*x1 + 3*x2;
13
14 output ["x1 = \"(x1)\\n\",
15         \"x2 = \"(x2)\\n\""];
16
Output

```


Kodumuzu çalıştıralım ve sonucu görelim.



```
1 % Use this editor as a MiniZinc scratch book
2 var 0..100: x1;
3 var 0..100: x2;
4
5 constraint x1 + 2*x2 <= 2;
6
7 constraint 2*x1+x2 >= 6;
```

Output

```
Compiling Uygun_cozum_yok_v2.mzn
Running Uygun_cozum_yok_v2.mzn
====UNSATISFIABLE====
Finished in 389msec
```

Line: 2, Col: 1 389msec

Programda da görüldüğü üzere Soru-5’de ortak çözüm alanı olmadığı için ekrana “UNSATISFIABLE” sonucunu vermektedir.

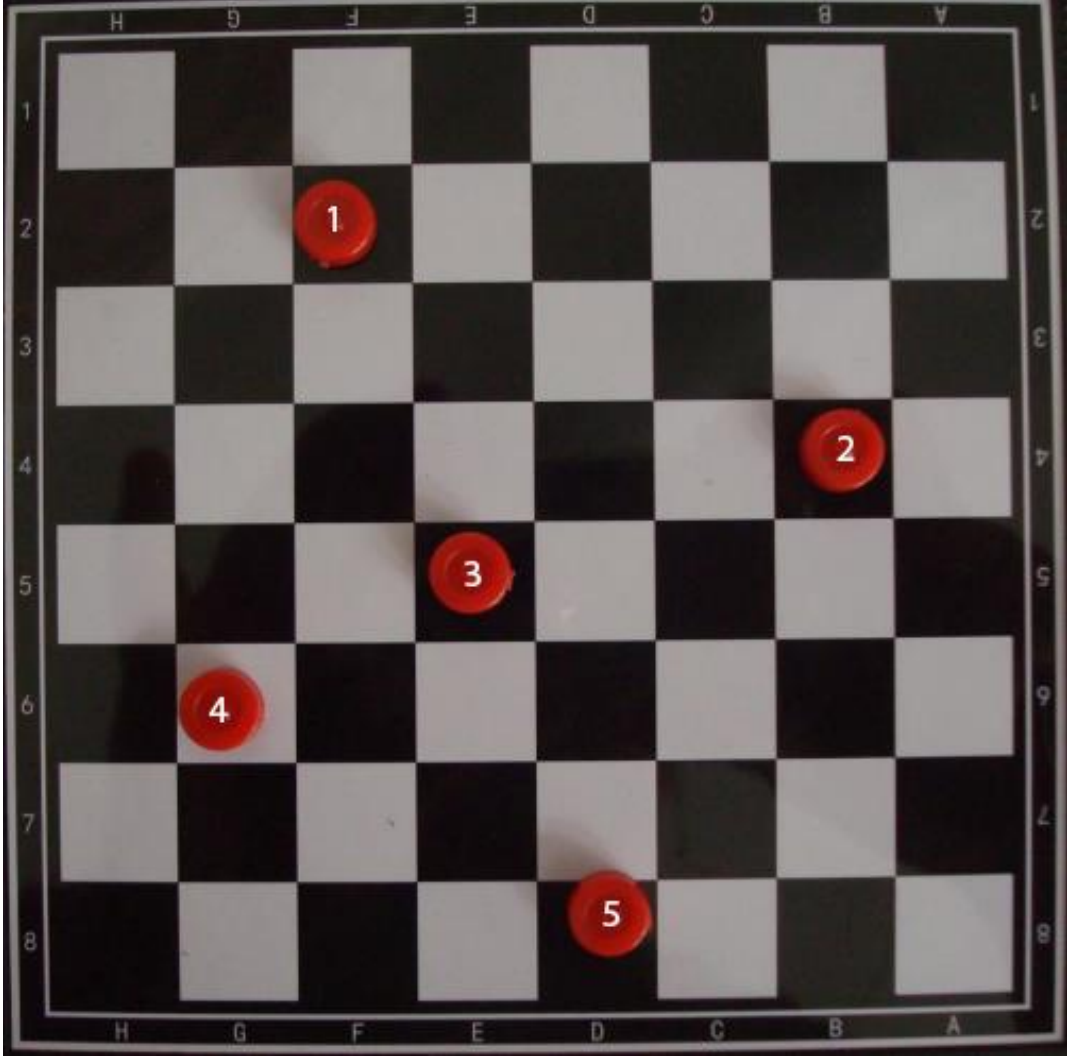
Soru 6:

Bu soruda ise yöneylem araştırması ve teorik bilgisayar bilimi alanlarında incelenen bir "kombinatorik optimizasyon" problemi olan gezgin satıcı problemini ele alacağız. Gezgin satıcı probleminin ne olduğuna değinmek gerekirse problem şu şekilde tanımlanabilir:

- Bir seyyar satıcı var.
- Bu satıcı, mallarını n şehirde satmak istiyor.
- Öte yandan, mantıklı bir şekilde, bu satıcı bu şehirleri mümkün olan en kısa şekilde ve her bir şehre maksimum bir kere uğrayarak turlamak istiyor.

Problemin amacı, satıcıya bu en kısa yolu sunabilmektir.

Bu problem 5 şehre indirildiğinde ve satranç tahtası üzerinde ele alındığında:



Satranç tahtası üzerindeki kırmızı taşların her biri şehirleri, siyah-beyaz kareler ise yolları temsil etmektedir. 5 şehrimizin olduğunu ve taşların uzaklıklarını matris olarak MiniZinc’te tanımlayalım:

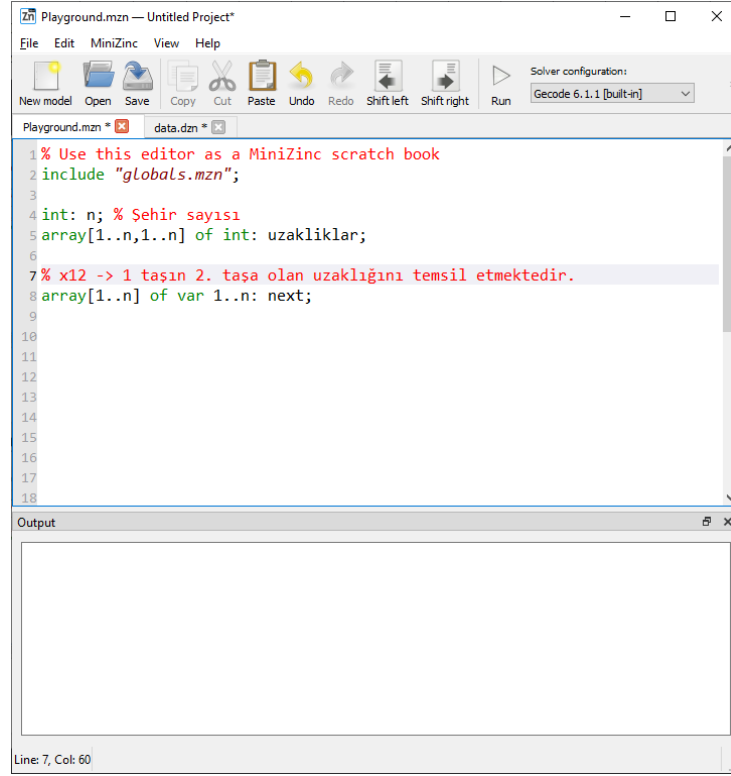
```
1 n = 5; %Şehir Sayısı
2 uzakliklar= [|0,6,4,5,8 % x11 x12 x13 x14 x15
3              |6,0,4,7,6 % x21.....x25
4              |4,4,0,3,4 % x31.....x35
5              |5,7,3,0,5 % x41.....x45
6              |8,6,4,5,0|]; % x51.....x55
7
8
9 % Tüm taşların birbirine olan uzaklık değerleri
10
```

Burada $n = 5$ şehir sayısını, alttaki matris değerleri ise i . taşın j . taşa olan uzaklığını temsil etmektedir(x_{12} : 1. taşın 2. taşa olan uzaklığı).

Öncelikle şehir sayısını temsil etmek için “ n ” değişkenini ve şehirler arası uzaklık matrisini tanımlayalım:

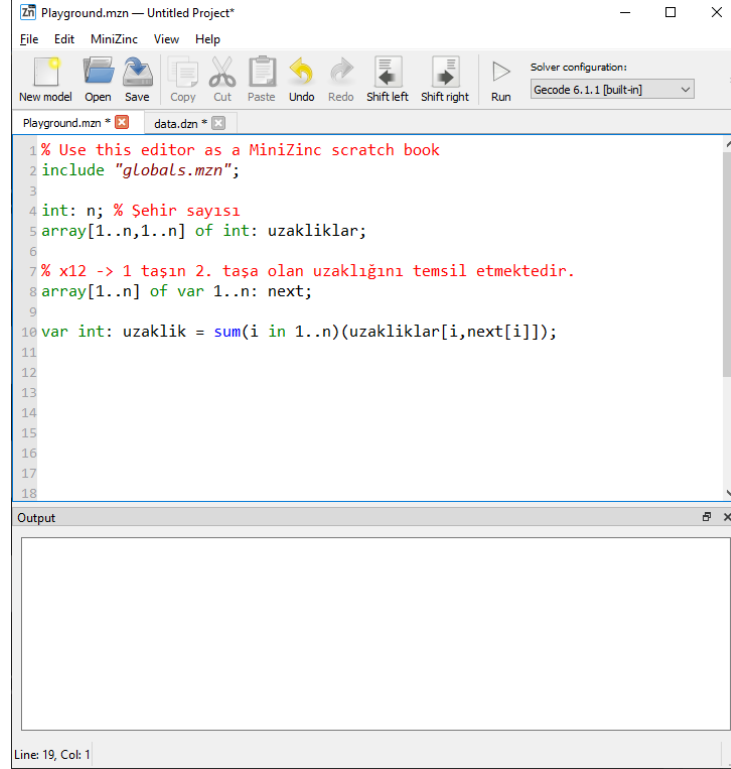
```
1 % Use this editor as a MiniZinc scratch book
2 include "globals.mzn";
3
4 int: n; % Şehir sayısı
5 array[1..n,1..n] of int: uzakliklar;
6
7
8
9
10
11
12
13
14
15
16
17
18
```

Bir sonraki şehri ziyaret edebilmek için şu kod satırını yazmak gerekir:



```
1 % Use this editor as a MiniZinc scratch book
2 include "globals.mzn";
3
4 int: n; % Şehir sayısı
5 array[1..n,1..n] of int: uzakliklar;
6
7 % x12 -> 1 taşın 2. taşa olan uzaklığını temsil etmektedir.
8 array[1..n] of var 1..n: next;
9
10
11
12
13
14
15
16
17
18
```

Alınan toplam mesafe için aşağıdaki uzaklık değişkenini yani amaç fonksiyonunu tanımlayalım:



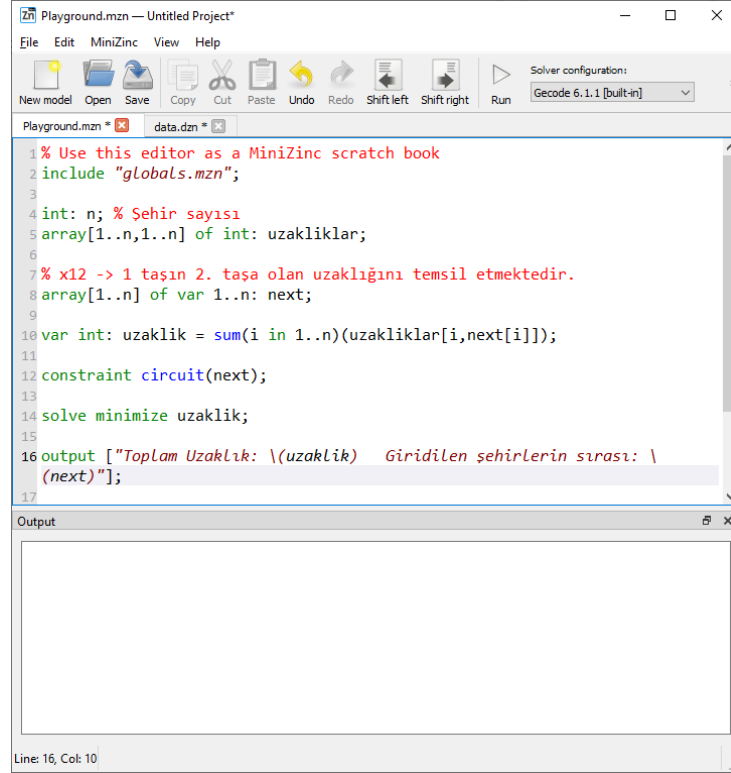
```
1 % Use this editor as a MiniZinc scratch book
2 include "globals.mzn";
3
4 int: n; % Şehir sayısı
5 array[1..n,1..n] of int: uzakliklar;
6
7 % x12 -> 1 taşın 2. taşa olan uzaklığını temsil etmektedir.
8 array[1..n] of var 1..n: next;
9
10 var int: uzaklik = sum(i in 1..n)(uzakliklar[i,next[i]]);
11
12
13
14
15
16
17
18
```

Tüm şehirlere maksimum 1 kere gidilebilmesi için; yani aynı şehre tekrar uğramaması için aşağıdaki komut satırı yazılmalıdır:

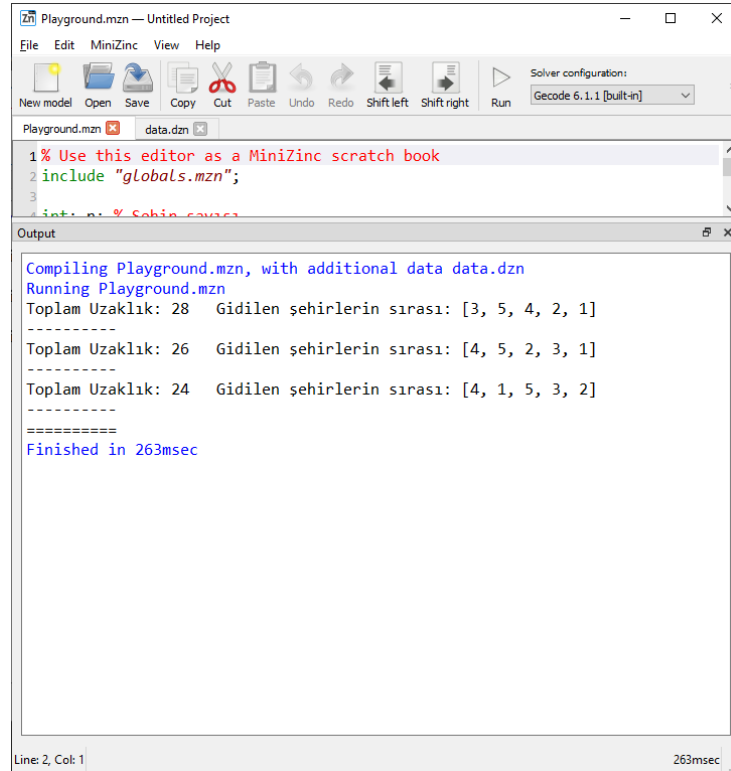
```
1 % Use this editor as a MiniZinc scratch book
2 include "globals.mzn";
3
4 int: n; % Şehir sayısı
5 array[1..n,1..n] of int: uzakliklar;
6
7 % x12 -> 1 taşın 2. taşa olan uzaklığını temsil etmektedir.
8 array[1..n] of var 1..n: next;
9
10 var int: uzaklik = sum(i in 1..n)(uzakliklar[i,next[i]]);
11
12 constraint circuit(next);
13
14
15
16
17
18
```

Görüldüğü gibi kısıtımız katedilen yoldan oluşmaktadır. Dolayısıyla ifademiz “constraint” komutu ile yazılmıştır. “circuit” fonksiyonu ile yazılmasının sebebi, bu yol bir daire gibi düşünüldüğünde gidilen bir şekilde tekrar gidilmeyeceğinden, bu daire en küçük olacak şekilde optimize edilmelidir. Dolayısıyla bu komut satırı ile -tıpkı bir daire gibi- her şehre maksimum 1 defa gidilmiş ve aynı şehre tekrar gidilmemiş olur.

Son olarak amaç fonksiyonunu minimum yapacak komutu yazalım ve çıktıyı ekrana gösterecek komutu da ekleyerek sonuçları görelim:



Sonuçlar aşağıdaki gibidir:



Görüldüğü üzere sırasıyla [4, 1, 5, 3, 2] şehirlerine gidildiğinde en kısa yoldan(24) şehirler ziyaret edilmiş olur.

Soru 7:

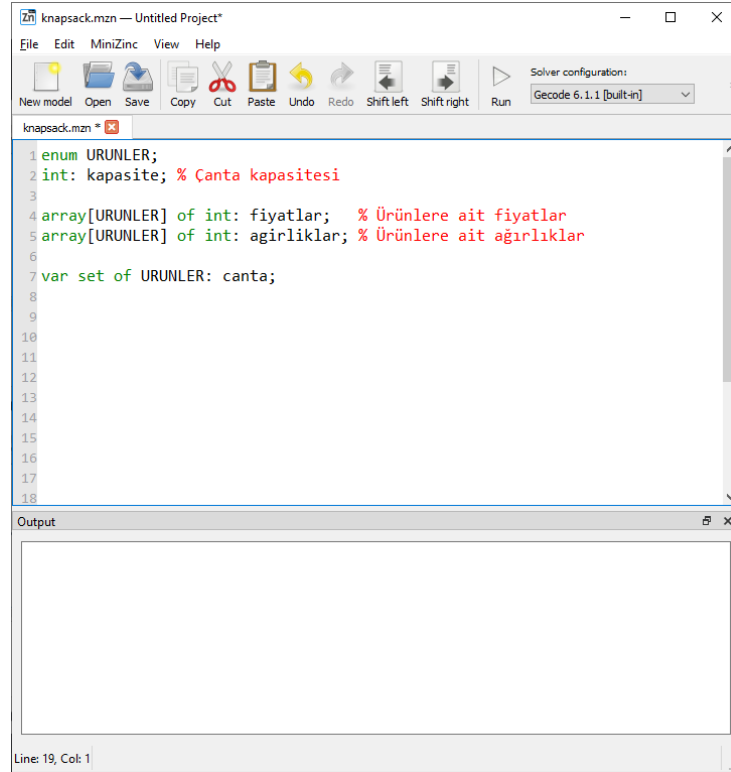
Son soruda ise klasik optimizasyon problemlerinden bir diğeri olan çanta problemi(knapsack problem) ele alınacaktır. Çanta problemi, basitçe bir çantanın içerisine maksimum fiyatta en fazla eşya konulması hedeflenir. Örnek olarak elimizde aşağıdaki eşyalar bulunsun:

1. Saat 200g 100TL
2. Fotoğraf makinesi 500g 300TL
3. Kamera 700g 700TL
4. Cep telefonu 300g 500TL
5. Anahtar 100g 10TL

Problemimizin amacı, 1000g kapasiteli çantaya yukarıdaki eşyaları maksimum fiyatta en fazla eşya konulacak şekilde yerleştirilmesidir.

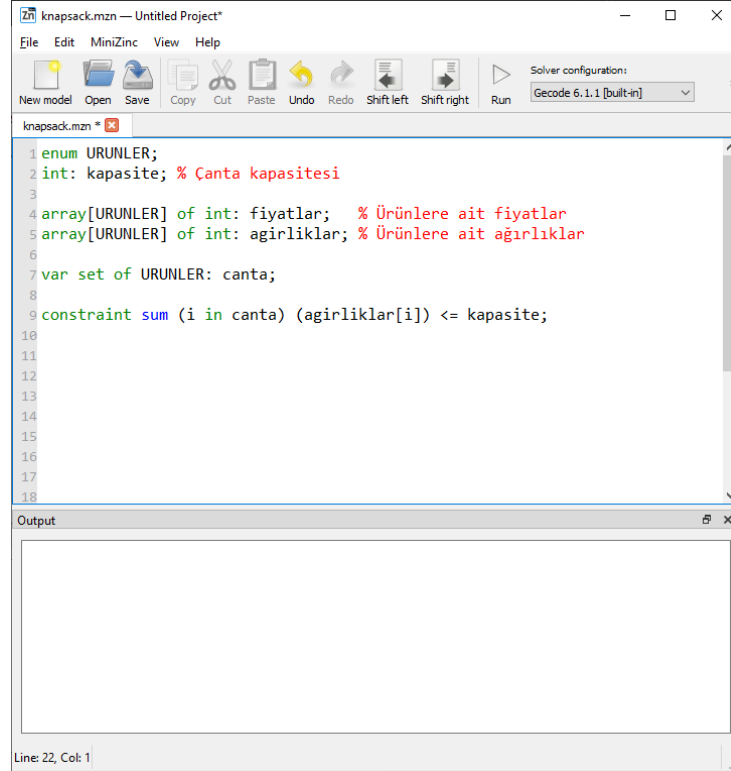
Çözümümüz ise ürünler arasından kamera ve cep telefonunu seçtiğimizde 1200 TL ile maksimum değeri elde etmiş oluruz. Şimdi bu problemi MiniZinc programında tanımlayalım:

İlk adımda her zamanki gibi değişkenlerimizi tanımlayalım:



```
1 enum URUNLER;  
2 int: kapasite; % Çanta kapasitesi  
3  
4 array[URUNLER] of int: fiyatlar; % Ürünlere ait fiyatlar  
5 array[URUNLER] of int: ağırlıklar; % Ürünlere ait ağırlıklar  
6  
7 var set of URUNLER: çanta;  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18
```

Bizim burada tek kısıtımız olan ağırlık kısıtı, çanta kapasitesini geçmeyeceği için bu komut satırı aşağıdaki gibi yazılmalıdır:

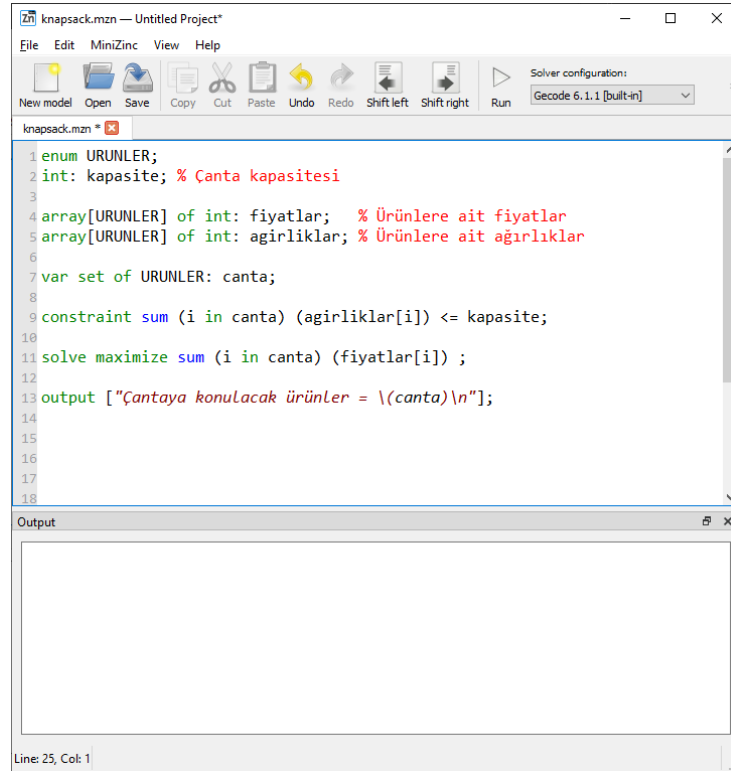


```
1 enum URUNLER;
2 int: kapasite; % Çanta kapasitesi
3
4 array[URUNLER] of int: fiyatlar; % Ürünlere ait fiyatlar
5 array[URUNLER] of int: agirliklar; % Ürünlere ait ağırlıklar
6
7 var set of URUNLER: canta;
8
9 constraint sum (i in canta) (agirliklar[i]) <= kapasite;
10
11
12
13
14
15
16
17
18
```

Output

Line: 22, Col: 1

Bu adımda ise ağırlık kısıtı haricinde maksimum fiyatı elde etmek istediğimiz için bu komut satırı ve çıktı komutu aşağıdaki gibi olmalıdır:

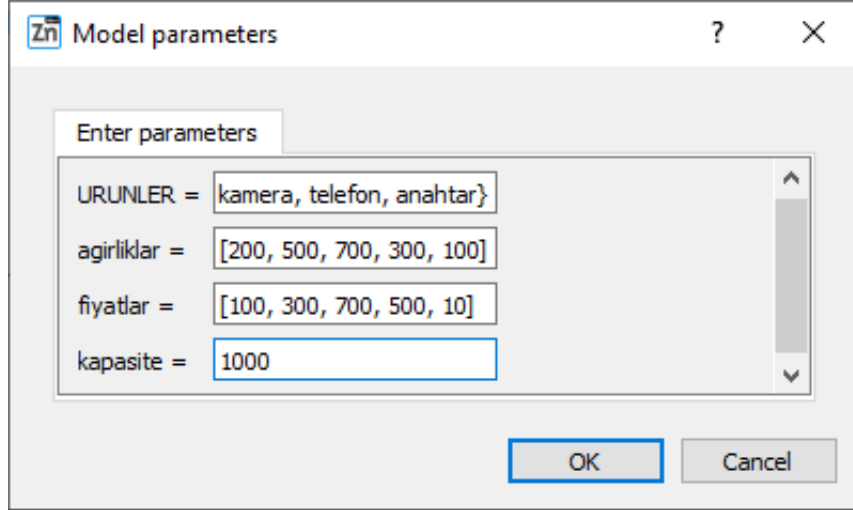


```
1 enum URUNLER;
2 int: kapasite; % Çanta kapasitesi
3
4 array[URUNLER] of int: fiyatlar; % Ürünlere ait fiyatlar
5 array[URUNLER] of int: agirliklar; % Ürünlere ait ağırlıklar
6
7 var set of URUNLER: canta;
8
9 constraint sum (i in canta) (agirliklar[i]) <= kapasite;
10
11 solve maximize sum (i in canta) (fiyatlar[i]) ;
12
13 output ["Çantaya konulacak ürünler = \"(canta)\\n\""];
14
15
16
17
18
```

Output

Line: 25, Col: 1

Program çalıştırıldığında bizden parametre değerleri istenecektir:



Model parameters

Enter parameters

URUNLER = kamera, telefon, anahtar}

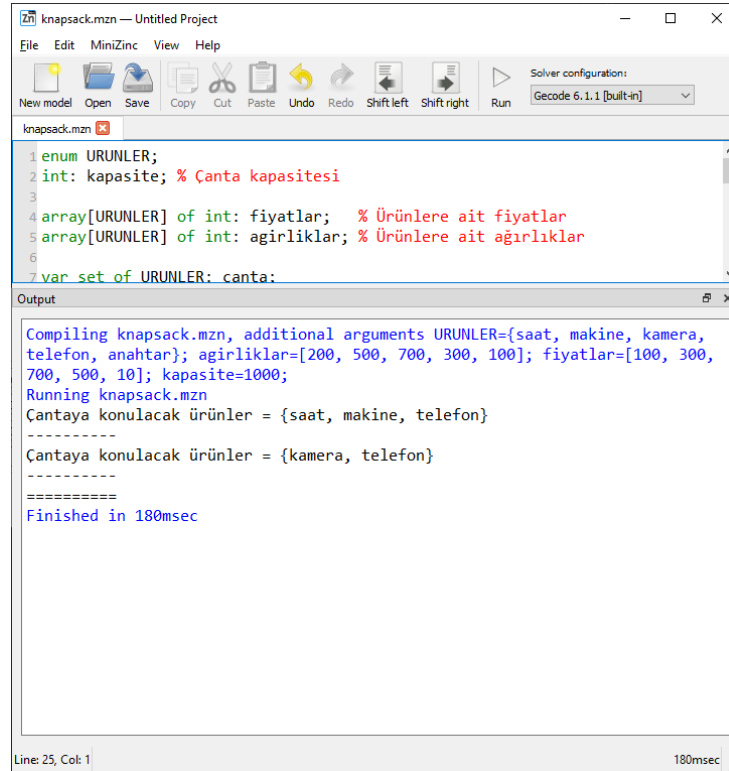
agirliklar = [200, 500, 700, 300, 100]

fiyatlar = [100, 300, 700, 500, 10]

kapasite = 1000

OK Cancel

Sonuçlarımız aşağıdaki gibidir:



```

1 enum URUNLER;
2 int: kapasite; % Çanta kapasitesi
3
4 array[URUNLER] of int: fiyatlar; % Ürünlere ait fiyatlar
5 array[URUNLER] of int: agirliklar; % Ürünlere ait ağırlıklar
6
7 var set of URUNLER: canta;

```

Output

```

Compiling knapsack.mzn, additional arguments URUNLER={saat, makine, kamera,
telefon, anahtar}; agirliklar=[200, 500, 700, 300, 100]; fiyatlar=[100, 300,
700, 500, 10]; kapasite=1000;
Running knapsack.mzn
Çantaya konulacak ürünler = {saat, makine, telefon}
-----
Çantaya konulacak ürünler = {kamera, telefon}
-----
=====
Finished in 180msec

```

Line: 25, Col: 1 180msec

Görüldüğü üzere daha önce çözdüğümüz gibi “kamera” ve “telefon” sonuçları elde edilmiştir.

YARARLANILAN KAYNAKLAR

URL, “MiniZinc”, <https://www.minizinc.org/>, (2014).

URL, “Optimizasyona Giriş ve Temel Kavramlar”,<http://ikucukkoc.baun.edu.tr/lectures/EMM3208/EMM3208W2.pdf>, (2018).

URL, “Optimizasyon Nedir”, <https://www.ceyrekmuhendis.com/optimizasyon-nedir/>, 02.03.2019).

URL, “Optimizasyon”, <https://tr.wikipedia.org/wiki/Optimizasyon>, (11.04.2020).

URL, “Doğrusal Programlama”,
http://www.ktu.edu.tr/dosyalar/ormanamenajmani_4ef1e.pdf,(2018).

URL, “Doğrusal Programlama”,
<http://www.ozyazilim.com/ozgur/marmara/karar/simplex.htm>, (1997).

URL, “Grafik Yöntemi ve Örnekler”,
http://www.ktu.edu.tr/dosyalar/ormanamenajmani_1a08b.pdf,(2018).

URL, “Gezgin Satıcı Problemi”,
https://tr.wikipedia.org/wiki/Gezgin_sat%C4%B1c%C4%B1_problemi, (19.04.2020)

URL, ”Knapsack Problemi”,
<http://bilgisayarkavramlari.sadievrenseker.com/2008/03/24/torba-problemi-knapsack-problem/>,(24.03.2008).