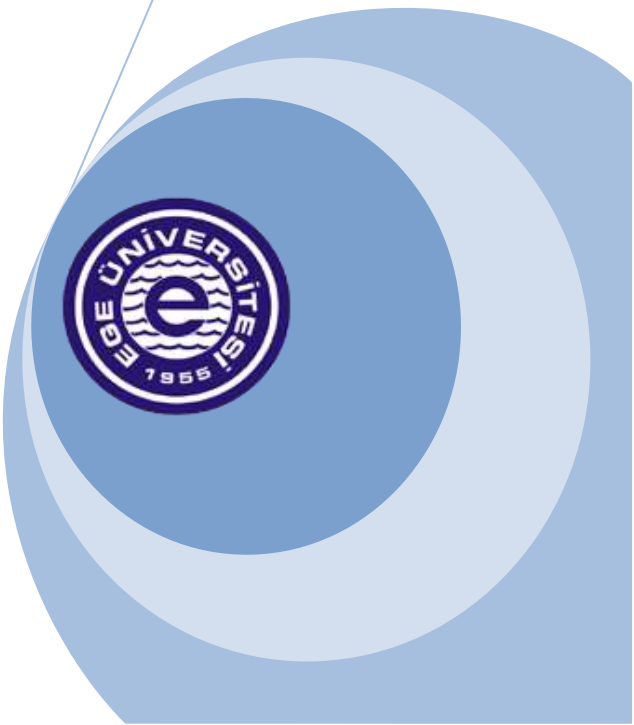
Genetik Knapsack

Umut BENZER 05-06-7670 http://www.ubenzer.com
Ege Üniversitesi Bilgisayar Mühendisliği
Yapay Zekâ

31.10.2011



İçindekiler

Programcı Kataloğu

Teknik Bilgiler

Uygulama JAVA'da geliştirilmiştir. Uygulamanın geliştirilirken ve test edilirken kullanılan sistem aşağıdaki gibidir:

Makine Özellikleri

Intel i7 2.0Ghz QuadCore processor, 8GB RAM

Yazılım Özellikleri

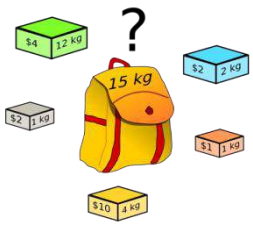
Microsoft Windows 7 Professional Service Pack 1, **Java 7 Update 1**

Uygulama Eclipse Helios IDE'sinde geliştirilmiştir.

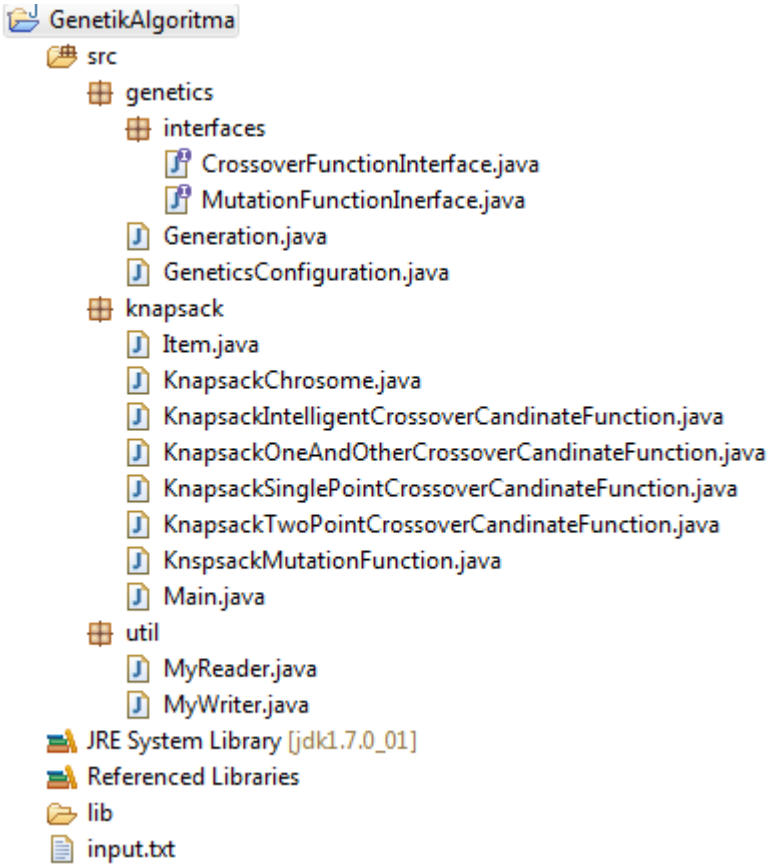
Problemin Kısa Tanımı

Bu projede temel amaç 0-1 Knapsack problemini genetik algoritmalar yardımıyla çözmektir. Knapsack problemi aşağıdaki gibi özetlenebilir:

0-1 sırt çantası probleminde mevcut n maddeden her biri ya 1 birim olarak çantaya konulur, ya da çantaya konulmaz, yani 0 birim çantaya koyulur. Çantaya konulup konulmama, sadece 1 ve 0 değerleri alan "çantada mevcut olup olmama" adı verilebilen iki kategorili değer alan bir karar değişkeni olur. Böylece karar değişkeni olan x_i ya 0 ya da 1 değeri alan iki kategorili "tamsayı değişkeni" olur. (Wikipedia)



Interface, Sınıf ve Metotlar



CrossoverFunctionInterface

Bir "jenerasyonu" alıp, kendisine verilen konfigürasyondaki kurallara uygun bir şekilde yeni bir "jenerasyon" yaratır.

Temel olarak yapması gereken işler şunlardır:

- Doğrudan geçecek kromozomları geçirmek.
- Crossover için kullanılacak kromozomları bir yönteme göre seçmek.
- Crossover yapmak.

Bu interface bir class tarafından implements edilmelidir. Crossover'ın tam olarak nasıl olacağı, crossover'a girecek

kromozomların belirlenmesi tamamıyla bu arayüzü implement eden sınıflara aittir. Bu proje kapsamında 4 farklı çaprazlama türü implement edilmiştir. Neler oldukları ilerleyen kısımlarda anlatılacaktır.

MutationFunctionInterface

Bir "jenerasyonu" alıp, kendisine verilen kurallar çerçevesinde "mutasyon" yapar. Algoritmanın iç işleyişi tamamıyla bu arayüzü implement eden sınıfa aittir. Bu projede bir tane mutasyon sınıfı vardır ve nasıl işlediği ilerleyen kısımlarda anlatılacaktır.

Generation

Bir nesli temsil eder. İçerisinde kromozomlar ve bunların içerisinde de genler bulunur. Bu sınıfın önemli tek metodu vardır:

```
public static void createGenerations(Generation firstGen, GeneticsConfiguration conf)
```

Genetik algoritmayı başlatmak için kullanılan temel bir metottur. firstGen ile, rastgele ya da herhangi bir algoritmayla yaratılmış ilk nesil ve hangi kural/algoritmalarla genetik algoritmanın işletileceğini tutan bir konfigürasyon sınıfı parametre olarak verilir. Sonuçlar doğrudan konsola yazdırılmaktadır.

GeneticsConfiguration

Genetik algoritmanın hangi ayarlarla çalışacağını tutan ve bir instanceının genellikle uygulama boyunca nesilden nesle geçirildiği ayar sınıfıdır. Programda genetik algoritma ile ilgili aşağıdaki ayarlar yapılabilmektedir:

```
public static enum CandinateFitnessType {
  MORE_IS_BETTER, LESS_IS_BETTER
};

private int chrosomeCount = 32; /* Bir nesildeki kromozom sayisi */
private int generationCount = 100; /* Algoritma kac nesil boyunca isletilecek? */
private CandinateFitnessType candinateFitnessType; /* Fitness için: az mi iyidir, cok mu? */
private CrossoverFunctionInterface generateCandinatesFunction; /* Crossover algoritmasi referansi */
private int directlyInheriedChrosomeCount = 1; /* Crossoversiz gecirilecek kromozom sayisi */
private boolean allowMutationsInDirectlyInheritedChrosomes = false; /* CO'siz gecirilen kromozomlarda
mutasyon yapilacak mi? */
private double mutationProbobalityPerGene = 0.1; /* Gen basina mutasyon olasiligi */
private int maxMutatedGeneCountInAChrosome = 2; /* Bir kromozomda en fazla kac gende mutasyon olabilir?
*/
private MutationFunctionInterface mutationFunction; /* Mutasyon fonksiyouna referans */
```

Bu ayarlar herhangi bir anda, bu sınıftan bir instance yaratacak ve bu ayarları constructorda geçirerek oluşturulur. Daha sonra genetik algoritmasına parametre olarak geçirilir.

Böylelikle çok farklı genetik algoritmaların bir ayar arayüzü ile kullanılabilmesi sağlanır.

Item

Knapsack probleminde, çantaya konabilecek, değeri ve ağırlığı olan her bir "Item"ı temsil eder. Klasik getter ve setter metotlardan oluşur. Nesneleri genelde bir dosyadan okunarak yaratılır. Bu sınıf ayrıca "Clonable" özelliğine sahiptir. **"Gen"e karşılık gelen bu sınıf** nesiller ve kromozomlar arasında sürekli kopyalandığından bu özelliğe ihtiyaç duyulmuştur.

KnapsackChrosome

Knapsack problemine özgü çeşitli bilgileri de içeren bir kromozomu temsil eder. İçerisinde genleri (Item) ve çantanın limitini taşır. Bu sınıf, Comparable sınıfını implement eder, karşılaştırılabilir. Bu, fitness değerlerine göre sıralama yaparken kullanılmaktadır.

Bu sınıfın önemli bir metodu vardır:

```
public double getChrosomeFitness()
```

Bu metot, kromozomun fitness değerini döndürür. Bunu yaparken (Knapsack Problemi için) içerisindeki genlerin değerlerini toplar. Eğer genlerin toplam ağırlığı çanta limitini aşıyorsa fitness değeri sıfırdır, değilse genlerin değerleri toplamına eşittir.

KnapsackIntelligentCrossoverCandinateFunction

Bu sınıf "akıllı crossover algoritması"dır. Ayrıntıları ve yorumlar diğer algoritmalarla birlikte ilerleyen kısımlarda anlatılacaktır.

KnapsackMutationFunction

Bu sınıf bir neslin kromozomları üzerinde konfigürasyonda verilenleri dikkate alarak mutasyon yapar. Mutasyon, bir Item'in çantaya alınıp alınmayacağını seçer. (isSelected = !isSelected)

KnapsackOneAndOtherCrossoverCandinateFunction

Bu sınıf "bir a kromozomundan gen alan, bir b kromozomundan gen alan crossover algoritması"dır. Ayrıntıları ve yorumlar diğer algoritmalarla birlikte ilerleyen kısımlarda anlatılacaktır.

KnapsackSinglePointCrossoverCandinateFunction

Bu sınıf "orta noktaya göre a ve b kromozomlarının tam ortadan ve tek noktadan birleşiminden oluşan crossover algoritması"dır. Ayrıntıları ve yorumlar diğer algoritmalarla birlikte ilerleyen kısımlarda anlatılacaktır.

KnapsackTwoPointCrossoverCandinateFunction

Bu sınıf "iki noktalı crossover algoritması"dır. Ayrıntıları ve yorumlar diğer algoritmalarla birlikte ilerleyen kısımlarda anlatılacaktır.

Main

Dosyadan problemin girdilerini okuyan, ilk nesli rastgele olarak yaratan ve her algoritma türü için problemi bir defa çözen, çalıştırılabilir olan sınıftır.

MyReader

Knapsack probleminin girdilerini, formatı daha önce belirlenmiş dosyadan okumak için kullanılır.

Dış Kaynaklardan Hazır Alınan Dosya ve Kütüphaneler

Dış kaynaklardan genetik algoritmalar ile ilgili herhangi bir kod örneği alınmamış, tamamı tarafımdan yazılmıştır. Ancak, diziler üzerinde yapılan çeşitli işlemleri kolaylaştıran (örneğin Array.reverse() gibi bir özellik sunan) Apache c**ommons-lang3-3.0.1.jar** kütüphanesinden yardımcı kütüphane olarak yararlanılmıştır.

Çalışma Süresi

Programı geliştirmek, incelenen örnekler ve ön araştırmalar ile birlikte yaklaşık 20 çalışma saati sürmüştür. Buna rapor için harcanan süre dahil değildir.

Kullanıcı Kataloğu

Programın Kullanımı

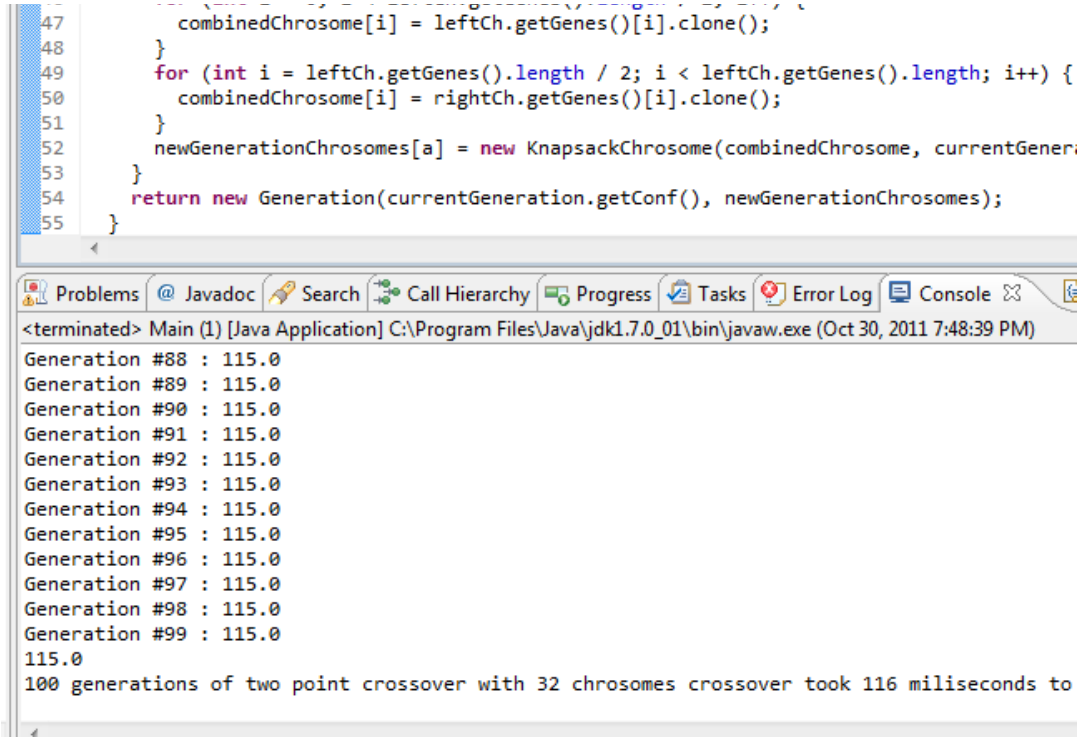
Programı bir Java Application olarak çalıştırdığınızda çalışacaktır. Program knapsack probleminin bilgilerini kendi klasörü içerisinde yer alan input.txt içerisinden alacaktır.

input.txt'nin formatı için aşağıdaki örnek verilebilir:

```
capacity = 10
item weight value
1      2      12
2      1      10
3      3      20
4      2      15
5      2      12
```

Rakamlar "tab" ile ayrılmıştır. Her satır bir Item'ı temsil eder. Capacity çantanın alabileceği maksimum ağırlıktır. Item metninin altındakiler her item için verilebilecek String tipinde birer isimdir. Projede şu an için bir kullanımı bulunmamaktadır. Weight ağırlık ve value değer sütunlarıdır. Program tam sayı kabul etmektedir, ancak bu bir kodlama kısıdıdır, çok hızlı bir refactoring ile virgüllü sayıları da kabul edebilir hale getirilebilir.

Projeyle beraber gönderilmiş dosyadaki knapsack problemi 16 itemdan oluşmaktadır ve doğru cevap **115**'tir. (Problem geçen sene Algoritma Analizi için yazdığım Knapsack algoritması ile çözülmüştür.)



Algoritmalar ve Deneysel Çalışma Sonucu Bulgular

Genel Bilgiler

Tüm algoritmalarda crossover edilecek genler rastgele olarak **rank selection** ile belirlenmektedir. Rank selection, Item'ların ağırlıkları çok farklı olabilecekleri için Roulette Whell'e göre daha iyi sonuç verecektir.

Tüm deneyler her nesil için **32 kromozom üzerinden** yapılmıştır.

Tüm çalışma süreleri **100 nesil için** ve milisaniye cinsindendir.

KnapsackSinglePointCrossoverCandinateFunction

Bu algoritma, klasik tek noktadan crossover algoritmasıdır. Algoritmanın 5 farklı uygulanışında (başlangıç değerleri de farklı, aynı problem için) genellikle 50'den daha az nesil sonucunda doğru sonucu bulduğu gözlenmiştir.

KnapsackTwoPointCrossoverCandinateFunction

Bu algoritma, klasik iki noktadan crossover algoritmasıdır. Yeni kromozomun ilk 1/3'ü ve son 1/3'ü bir kromozomdan, ortadaki 1/3'ü diğer kromozomdan alınarak birleştirilir. Algoritmanın 5 farklı uygulanışında (başlangıç değerleri de farklı, aynı problem için) genellikle 20'den daha az nesil sonucunda doğru sonucu bulduğu gözlenmiştir.

KnapsackOneAndOtherCrossoverCandinateFunction

Bu algoritma yeni kromozomun genlerini sırasıyla bir bir kromozomdan, bir diğer kromozomdan alır. Örneğin 6 genli A ve B kromozomları çaprazlanırsa sonuç ABABAB şeklinde olur. Algoritmanın 5 farklı uygulanışında (başlangıç değerleri de farklı, aynı problem için) genellikle 20'den daha az nesil sonucunda doğru sonucu bulduğu gözlenmiştir.

KnapsackIntelligentCrossoverCandinateFunction

"Zeki" olsun diye tasarlamaya çalıştığım özel crossover fonksiyonudur. Algoritma çaprazlanan kromozomlardan genleri alırken oluşturulan yeni kromozomun ağırlığının çantanın maksimum ağırlığını geçmemesine dikkat eder. Böylelikle oluşacak kromozomların unfeasible olmasını engellemeyi amaçlar.

Teoride bana mantıklı gelen bu algoritma pratikte işe yaramamıştır. Crossoverın temel amaçlarından olan "çeşitliliği artırmak" konusunda başarılı olamamıştır. Genellikle 100 nesilde doğru sonuç bulunamamıştır.

Yorumlar

Algoritmalar aynı knapsack problemi üzerinde tamamen rastgele başlangıç değerleri ile çalıştırılmışlardır. Her algoritma birinci deneyde üçer defa, ikinci deneyde 1000'er defa çalıştırılmış ve değerlendirme için ortalamaları alınmıştır. 1000 defa çalıştırılarak yapılan deneyde her sample için nesil değerleri tabloya tablonun çok karışmaması için eklenmemiş, doğrudan ortalamaları tabloda gösterilmiştir.

Intelligent algoritması, bir üst başlıkta anlatılan sebeplerden dolayı iyi bir crossover algoritması olamamıştır. Diğerleri bu algoritmaya göre bariz şekilde başarılı çalışmaktadır. Bu yüzden bu algoritma değerlendirmeye bile alınmayacaktır.

Diğer algoritmalar ek dokümandaki grafiklerde de görülebileceği gibi birbirlerine göre üstünlük sağlayamamışlardır.

Tek noktadan crossover yöntemi ortalama 20.968 nesil sonucunda doğru sonuca ulaşırken uniform 33.443, çift noktalı crossover ise 40.815 nesil sonucunda doğru sonuca ulaşmıştır. Bu sayılar tek noktadan crossover yöntemini daha iyi gibi gösterse de, bu sayılar birbirlerine çok yakın sayılar olduğundan ve genetik algoritmada şans çok önemli bir faktör olduğundan "birisi diğerinden daha iyidir" diyebileceğimiz kadar anlamlı değillerdir.

Belki başka problemlerde bir çaprazlama diğerine göre daha iyi olabilir. Ancak knapsack probleminin en önemli dez avantajı daha çok değere (value) sahip bir kombinasyon üreteyim derken kolaylıkla çantanın limitini aşabilmeniz ve bu yüzden aslında elde ettiğiniz yüksek değerin sıfır değerinde olmasıdır. Bu nedenle bir çaprazlama diğerine göre daha iyi nesiller sunsa da, aslında sunmuyor olabilir.

Üçü de bu problem için kullanılabilecek algoritmalardır.

Çalışma sürelerine göre one and other algoritması daha hızlı görünse de bu tamamıyla benim algoritmaları implemente etmemle alakalıdır. Single point ve two point crossover algoritmalarında birden fazla for döngüsü varken, one and other algoritmasında tek for döngüsü bulunmaktadır. Optimizasyon yapılıp tüm algoritmalar tek for ile çalışacak hale getirilebilir ve bu yapıldığında neredeyse hiç hız farkı kalmayacaktır.