



Технически Университет – София

Курсова работа

по

Метаевристика

Тема:

Artificial Bee Colony algorithm

Изготвили:

Кирил Александров, №121312011, гр 224

Георги Петков, №121312060, гр 222

Валентин Бахтев, №121312009, гр 225

1. Алгоритъмът „Изкуствено пчелно семейство“ (Artificial Bee Colony).

Алгоритъмът „Artificial bee colony“ е метаевристичен алгоритъм за оптимизиране на числени проблеми, въведен през 2005 г. от Karaboga. Идеята за алгоритъма се базира на модела, предложен от „Tereshko and Loengarov(2005)“, за интелигентното поведение на медоносните пчелни колонии по време на събиране на прашец от цветята. Този модел се състои от три съществени компонента, а именно:

- 1) Пчели работнички
- 2) Пчели скаути
- 3) Източници на нектар

Тук първите два компонента (пчелите работнички и пчелите скаути) търсят богати източници на нектар близо до техния кошер, които се явяват третия основен компонент на модела.

Моделът също така се характеризира с два водещи начина на поведение, които са необходими за интелигентното, самостоятелно организиране и колективност: събирането на множество скаути на едно място, представляващо богат източник на нектар, води до положителна обратна връзка, докато напускането на скаутите от дадено място, представляващо беден източник на нектар, води до отрицателна обратна връзка. При алгоритъмът „Artificial Bee Colony“ имаме изкуствени пчелни колонии, търсещи изкуствени богати източници на храна (нектар). При прилагане на алгоритъма, конкретния оптимизационен проблем първо се преобразува до проблем за откриване на най – добрия параметричен вектор за минимизиране на обективна функция. След това преобразуване, изкуствените пчелни колонии започват да разкриват първоначални вектори на решения на случаен принцип, след което на база множество итерации подобряват решението на следния принцип: преминаване към по – добро решение базирайки се на механизма за търсене в съседство, като същевременно с това по – бедните и неточни решения започват да отпадат.

Пример за глобален оптимизационен проблем би могъл да се дефинира по следния начин:

Търсене на параметричния вектор \vec{x} за минимизиране на обективната функция $f(\vec{x})$:

$$\text{minimize } f(\vec{x}), \quad \vec{x} = (x_1, x_2, \dots, x_i, \dots, x_{n-1}, x_n) \in R^n \quad (1)$$

ограничавайки от следните неравенства и/или равенства:

$$l_i \leq x_i \leq u_i, \quad i=1, \dots, n \quad (2)$$

Ако:

$$g_j(\vec{x}) \leq 0, \text{ for } j=1, \dots, p \quad (3)$$

$$h_j(\vec{x}) = 0, \text{ for } j=p+1, \dots, q \quad (4)$$

$f(\vec{x})$ е декларирана в пространството на търсене, S , което е n – размерен правоъгълник в R^n ($S \subseteq R^n$).

Областите на променливата са ограничени от техните долни и горни граници(2) .

Този проблем е известен още като „Constrained Optimization Problem“.

1.1. Метаевристичен алгоритъм с изкуствени пчелни колонии.

При метаевристичния алгоритъм „Artificial Bee Colony“, изкуствените пчелни колонии се състоят от три основни вида пчели:

- 1) Пчели работнички – тези пчели са асоциирани със специфични източници на храна(нектар)
- 2) Пчели наблюдателки – тези пчели наблюдават танца на пчелите работнички в рояка с цел избор на източник на нектар.
- 3) Пчели скаути – тези пчели търсят нови източници на нектар близо до кошера на произволен принцип.

Пчелите скаути и пчелите наблюдателки се наричат още неработещи(незаети). Първоначално всички посоки на източници на нектар започват да се разучават от пчелите скаути. След това нектара от източниците на храна бива източен от пчелите работнички и пчелите наблюдателки, като продължителното източване води и до тяхното изтощаване. След източването на конкретния източник на нектар, заетите пчели се превръщат в скаути, започвайки отново търсене на нов богат източник на храна. С други думи пчелите наблюдателки и работнички, чиито източник на нектар се изчерпа, се превръщат в пчели скаути.

При алгоритъма с изкуствени пчелни колонии, позицията на източника на нектар представлява едно потенциално решение на проблема, докато количеството нектар би могло да се асоциира с качеството на конкретното решение или оценка на резултата. Броят на заетите пчели е равен на броя на източниците на нектар(решенията), тъй като всяка една от заетите пчели а асоциирана единствено и само към един източник на храна(потенциално решение).

Примерната схема на алгоритъма би могла да бъде следната:

Initialization Phase

REPEAT

Employed Bees Phase

Onlooker Bees Phase

Scout Bees Phase

Memorize the best solution achieved so far

UNTIL(Cycle=Maximum Cycle Number or a Maximum CPU time)

1.1.1. Фаза на инициализиране(Initialization Phase).

Всички вектори на източници на храна „ xm^{\rightarrow} “ са инициализирани ($m = 1 \dots SN$, SN : размерът) от пчелите скаути.

Тъй като всеки един от източник на храна(xm^{\rightarrow}) представлява вектор на решение за даден оптимизационен проблем, всеки един от векторите „ xm^{\rightarrow} “ се състои от „ n “ на брой променливи($xmi, i=1 \dots n$), които трябва да бъдат оптимизирани с цел свеждане до минимум конкретната обективна функция.

Следната дефиниция, би могла да бъде използвана за целите на инициализацията:

$$xmi = li + rand(0,1) * (ui - li) \quad (5)$$

където: li и ui се явяват респективно горната и долната граница на параметъра xmi .

1.1.2. Фаза на заетите пчели(Employed Bee Phase).

Заетите пчели търсят за нови източници на храна(Vm^{\rightarrow}), които да съдържат по – голямо количество нектар и се намират в тяхно съседство (xm^{\rightarrow}). При откриване на нов източник те преоценяват рентабилността.

Като пример би могло да се даде следното:

Заетите пчели откриват нов по – богат на нектар източник в тяхно съседство (Vm^{\rightarrow}) използвайки следната формула:

$$v_{mi} = x_{mi} + \phi_{mi}(x_{mi} - x_{ki}) \quad (6)$$

където:

- 1) x_k - източник на нектар, избран на случаен принцип
- 2) i – параметър индекс, избран на случаен принцип.
- 3) ϕ_{mi} – число в интервала $[-a, a]$, избрано на случаен принцип.

След получаване на новия източник на нектар(Vm^{\rightarrow}) се изчислява и неговата оценка.

Пример за изчисляване на оценка или „fitness“ стойност за конкретно решение $fitm(xm^{\rightarrow})$ за минимизиране на проблеми би могъл да бъде следния:

$$fitm(\vec{x}_m) = \begin{cases} \frac{1}{1 + f_m(\vec{x}_m)} & \text{if } f_m(\vec{x}_m) \geq 0 \\ 1 + abs(f_m(\vec{x}_m)) & \text{if } f_m(\vec{x}_m) < 0 \end{cases} \quad (7)$$

Където:

$fitm(\vec{x}_m)$ е стойност на обективната функция за решение \vec{x}_m

1.1.3. Фаза на пчелите наблюдателки(Onlooker Bees Phase).

Незаетите пчели се състоят от две групи:

- 1) Пчели наблюдателки
- 2) Пчели скаути

Пчелите работнички споделят информация за източниците на нектар с пчелите наблюдателки чакащи в рояка, след което пчелите наблюдателки избират съответния източник на база тази информация.

При алгоритъма с изкуствени пчелни колонии, пчелите наблюдателки избират конкретните източници на нектар на база вероятностни стойности, изчислени с помощта на фитнес стойностите, предоставени от пчелите работнички. За тази цел, биха могли да се използват селектиращи техники на база предоставените фитнес стойности, подобно на селектирането при въртяща се ролетка(Goldberg, 1989).

Вероятностната стойност p_m с избрания от пчела наблюдател източник \vec{x}_m би могла да се изчисли използвайки следния израз:

$$p_m = \frac{fit_m(\vec{x}_m)}{\sum_{m=1}^{SN} fit_m(\vec{x}_m)} . \quad (8)$$

След избор на конкретен източник \vec{x}_m от пчела наблюдателка, съседния източник на нектар \vec{V}_m се определя използвайки равенство (6), след което се изчислява и неговата фитнес стойност.

Като резултат получаваме насочване на множеството пчели наблюдателки към източниците на по – големи количества нектар.

1.1.4. Фаза на пчелите скаути(Scout Bees Phase).

Незаетите пчели, които избират източниците на нектар на произволен принцип се наричат пчели скаути. Пчелите работнички, чиито резултати не биха могли да се подобрят след определен брой проучвания, специфицирани от потребителите на алгоритъма и наречени „минимален“ или „критерии за изоставяне“, се превръщат в пчели скаути, като техните решения биват изоставени(пренебрегнати). След това пчелите превърнали се в скаути, започват да търсят нови, по – богати на нектар източници на произволен принцип.

Например:

Ако решение „ \vec{x}_m “ се пренебрегне, то ново решение получено от пчелата на „ \vec{x}_m “ преобразувала се в скаут, би могло да се дефинира чрез (5). В резултат на това, първоначално бедните източници на нектар или обеднените източници в резултат на

извлечен вече нектар, предизвикват негативна обратна връзка с цел балансиране на позитивната такава.

1.1.5. Заключение

Като заключение за Алгоритъма с изкуствени пчелни колонии(Artificial Bee Colony), бихме могли да обобщим следното:

- 1) Алгоритъмът е вдъхновен от интелигентното поведение на медните пчели.
- 2) Представлява алгоритъм за цялостна(глобална) оптимизация.
- 3) Алгоритъмът първоначално е бил предложен за частична оптимизация.
- 4) Този алгоритъм би могъл да бъде използван за комбинаторни оптимизационни проблеми.
- 5) Алгоритъмът би могъл да се използва както за ограничени, така и за неограничени оптимизационни проблеми.

2. Проблем и имплементираното от нас решение.

Тук ще разгледаме проблемът за търсене на конкретна информация в голямо количество от данни. Трудността на задачата идва от там, че не можем да обходим цялата налична информация, за да намерим това, което ни трябва. Това би било много бавна задача за изпълнение, която изисква твърде много ресурси. Ами какво би се случило, ако не намерим нищо, което да отговаря на критериите за търсене? Ново търсене с подобрени критерии?

Отново обхождане на цялата налична информация?

Добър пример за такъв проблем е например една електронна социална мрежа. Нека въпросната мрежа има много потребители (над един милион) и за всеки потребител се пази различна информация (пол, години, локация, интереси, любими филми/книги, работно място, образование и др.). Ако искаме да намерим конкретен човек, това би представлявало много трудна задача, ако базата от данните добре конструирана.

Но нека разгледаме проблемът с търсене на група от хора. Например искаме да извлечем информация за всички потребители, които са на възраст м/у 20 и 30 години, чели са „Властелинът на пръстените“, обичат филми на ужасите и са фенове на

Манчестър Юнайтед. Нека направим задачата още по-интересно – нека някои от критериите са задължителни, а останалите – не, нека всеки от критериите има различна важност (тежест). Такава задача не би мигла да се реши оптимално – колкото ясно да са ни дефинирани критериите, има възможност никой от потребителите да не влезе в тази група. Също така алгоритъм с пълно изчерпване би бил немислимо бавен. При такива проблеми е добре да се използва метаевристичен алгоритъм, който търси не оптимално, а задоволително решение.

Имплементираният и документиран тук алгоритъм, се състои от две части:

- Генериране на решения
- Обработка на решенията

За всяка от частите се грижат два отделни вида пчелички (нишки) – скаути и работнички. Всички операции се изпълняват паралелно и независимо. Това допринася за ускорението и ефективността на решението.

Скаутите отговарят за генерирането на решенията. Те обхождат наличните ресурси (Resource обектите) от т.н. „околна среда“ (Environment обекта). Решенията се записват в опашка. В реалния свят тази опашка представлява трети вид пчелички - „наблюдатели“. Те дават сигнали на пчелите работнички, кои територии да обработват.

В решението се използва опашка, защото най-добре се вписва за нуждите на алгоритъма без да прави реализацията по – сложна с допълнителен вид пчелички.

Третия вид пчелички са работничките – те обработват ресурсите, които са им показани от наблюдателите. В алгоритъма това е реализирано, чрез търсене в определения ресурс от опашката.

3. Имплементация

Имплементацията на алгоритъма се състои от два основни пакета:

- `abc` – В този пакет се намират всички класове и интерфейси, свързани с имплементацията на самият алгоритъм. Архитектурата му е направена така, че да може да бъде използван като библиотека.
- `example` – Тук са разположени класовете, които показват примерно използване на алгоритъма.

3.1. `abc`

В пакета `environment` се намират класовете, които представляват околната среда и ресурсите, които са обработвани от пчеличките.

Тук можем да намерим и специална имплементация на интерфейса `Set`, пригодена за нуждите на алгоритъма.

В пакета `criteria` се намират класовете и интерфейсите, които представляват различните критерии за пчеличките. За да се използва алгоритъма, трябва да се имплементират класове от тип `Criteria`, в които да се опише логиката за обработване на ресурсите.

В пакета `hive` са разположени класовете и интерфейсите свързани с “кошера” и пчеличките, които са основните действащи лица в алгоритъма.

Тук се намират класовете `EmployeeBee` и `ScoutBee`, които представляват имплементация на пчеличките работници и пчеличките скаути.

Програмно погледнато, всяка пчеличка е самостоятелна нишка, за това тези класове са наследници на класа `Thread`.

Класът `Hive` е основният клас, който навързва всички останали класове. На него се подава обект от тип `Environment` и се задават броят пчелички, който да го обработват.

3.2. `example`

В пакета `criteria` се намират примерни имплементации на класа `Criteria`.

В пакета `generator` са разположени класове генератори за `Environment` и `Resource` обекти, нужни за примера.

4. Експериментални резултати

Тук ще бъдат представени направените от нас експериментални резултати, като за целта ще бъдат използвани две машини, описани по – надолу в тази точка. Експерименталните резултати са направени с цел изследването на имплементирания от нас алгоритъм.

3.1 Изчисляване на ускорението

Ускорението показва предимството на паралелното решение на конкретен проблем, спрямо неговата последователна реализация.

Ускорението при n – процесорна система се определя, като отношението на времето за последователно изпълнение T_s , към времето за паралелно изпълнение T_{par} на един и същ проблем.

$$S_n = T_s / T_{par}$$

3.2 Изчисляване на ефективността

Изчисляването на ефективността на алгоритъма се определя като отношение на ускорението към броя на процесите, които участват в паралелното решаване на конкретен проблем:

$$E_n = S_n / n$$

3.3. Експериментални резултати на машина Dell Inspiron n5110

Характеристики на машината:

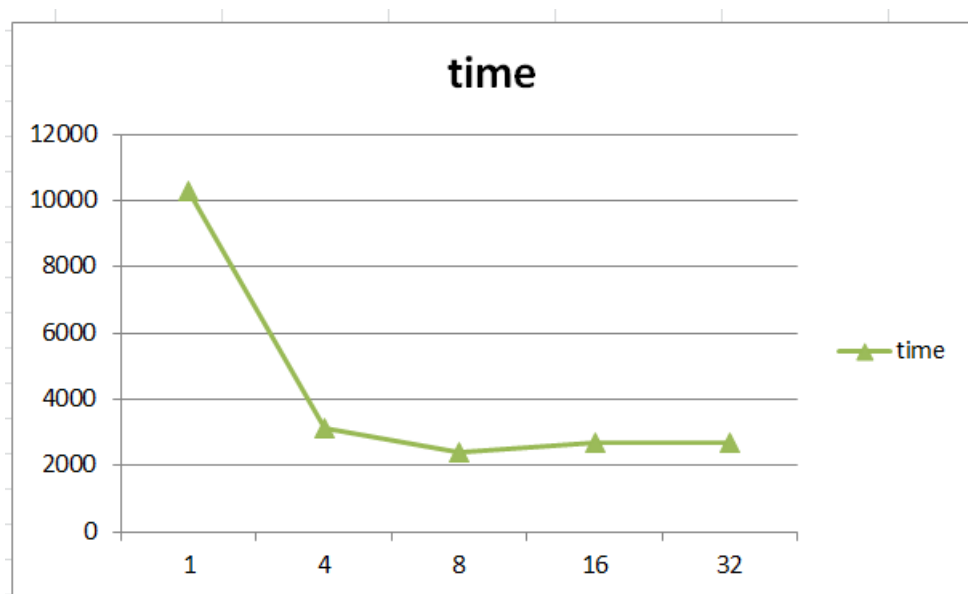
Chipset	Mobile Intel® 6 Series Express Chipset (HM67)
Processor	Intel® Core™ i7-2670QM (2.0GHz/6MB cache) Processor 2.2 GHz Turbo upto 3.1GHz, Qual Core, 8 Threads, 6M Cache)

Направени са опити с различен брой пчели работнички и скаути. Опитните резултати са в таблица 1.

scouts count	1	4	8	16	32
employees count	1	4	8	16	32
time	10279	3152	2428	2672	2711
speed	1	3,261104	4,233526	3,846931	3,79159
efficiency	1	0,815276	0,529191	0,240433	0,118487

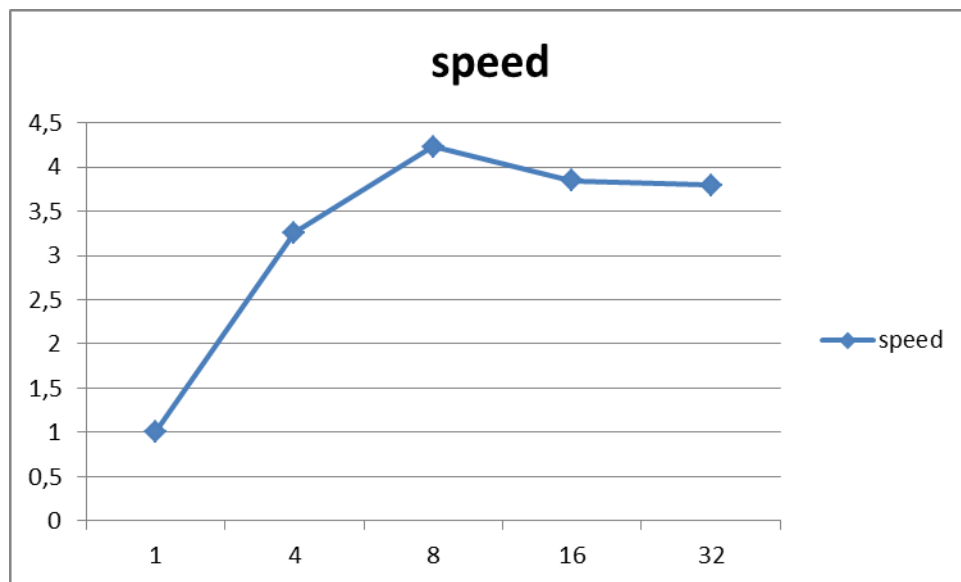
т а б л и ц а 1

На фигура 1 са показани опитните резултати в графичен вид.



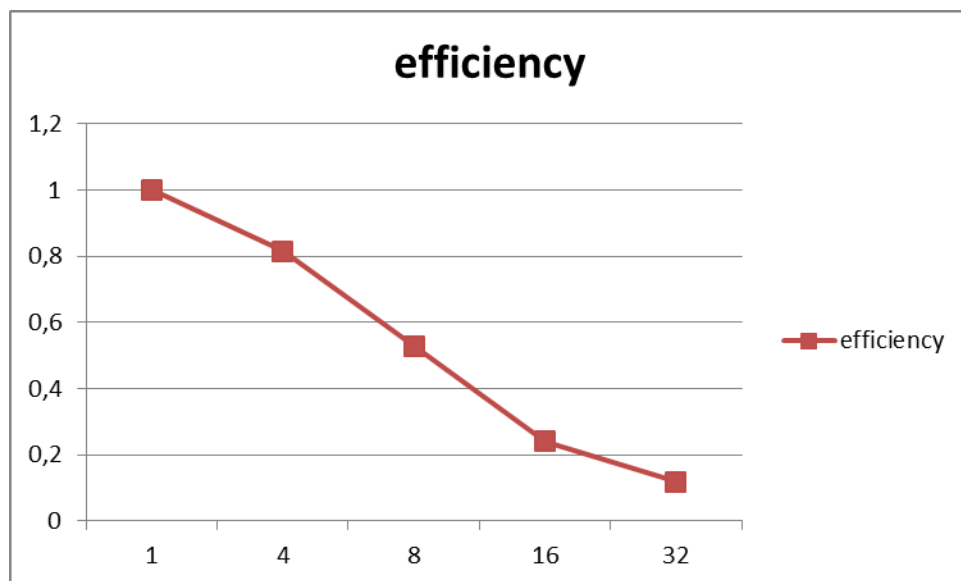
ф и г у р а 1 В р е м е з а и з п ъ л н е н и е

На фигура 2 е показано ускорението при различен брой пчели.



ф и г у р а 2 У с к о р е н и е

На фигура 3 е показана по графичен начин ефективността на алгоритъма.



ф и г у р а 3 Е ф е к т и в н о с т

3.4. Експериментални резултати на машина Asus N56V

Характеристики на машината:

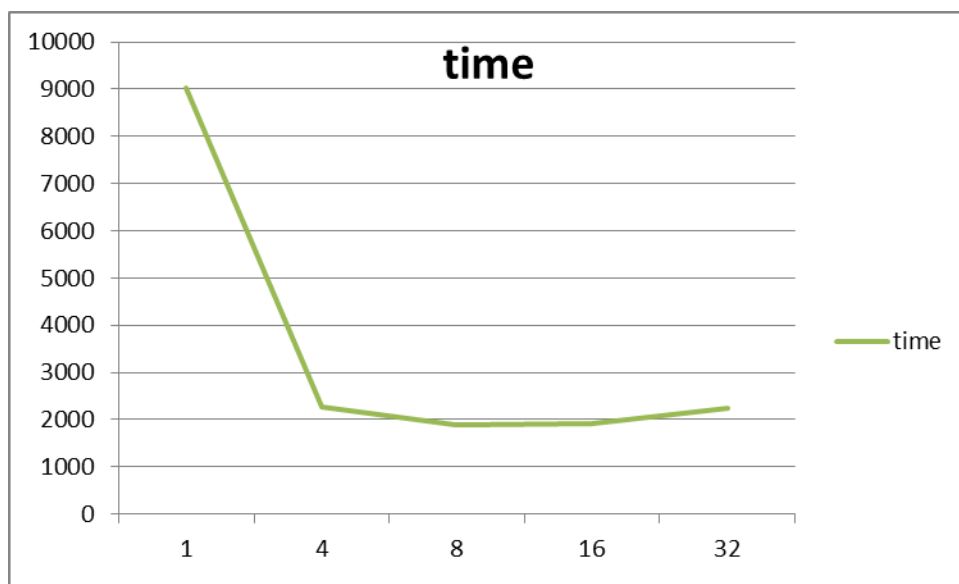
Chipset	Mobile Intel® 6 Series Express Chipset (HM67)
Processor	Intel® Core™ i7-3610QM (2.3GHz/6MB cache) Processor 2.3 GHz Turbo upto 3.3GHz, Qual Core, 8 Threads, 6M Cache)

Направени са опити с различен брой пчели работнички и скаути. Опитните резултати са в таблица 2.

scouts count	1	4	8	16	32
employees count	1	4	8	16	32
time	9016	2276	1876	1923	2234
speed	1	3,961336	4,80597	4,688508	4,03581

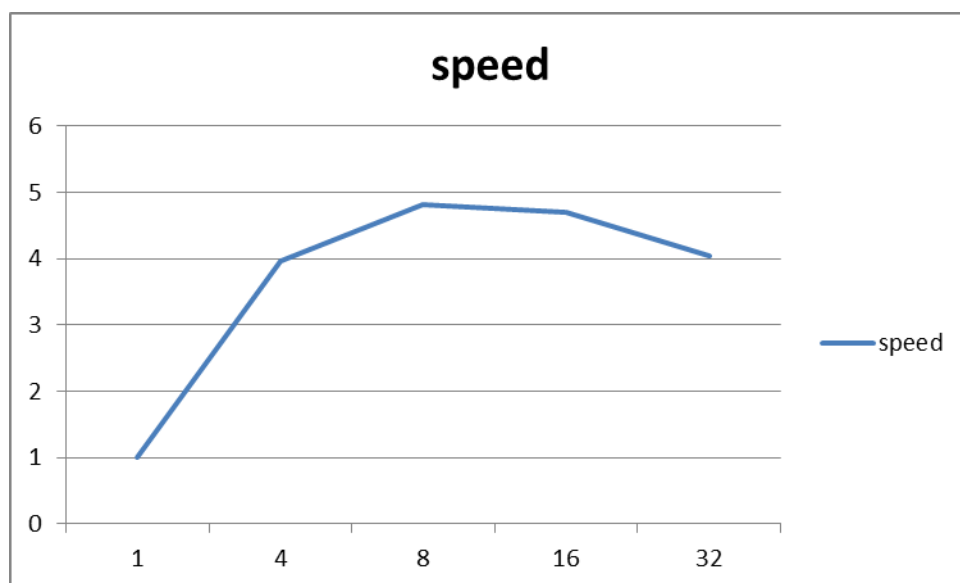
Т а б л и ц а 2

На фигура 4 са показани опитните резултати в графичен вид.



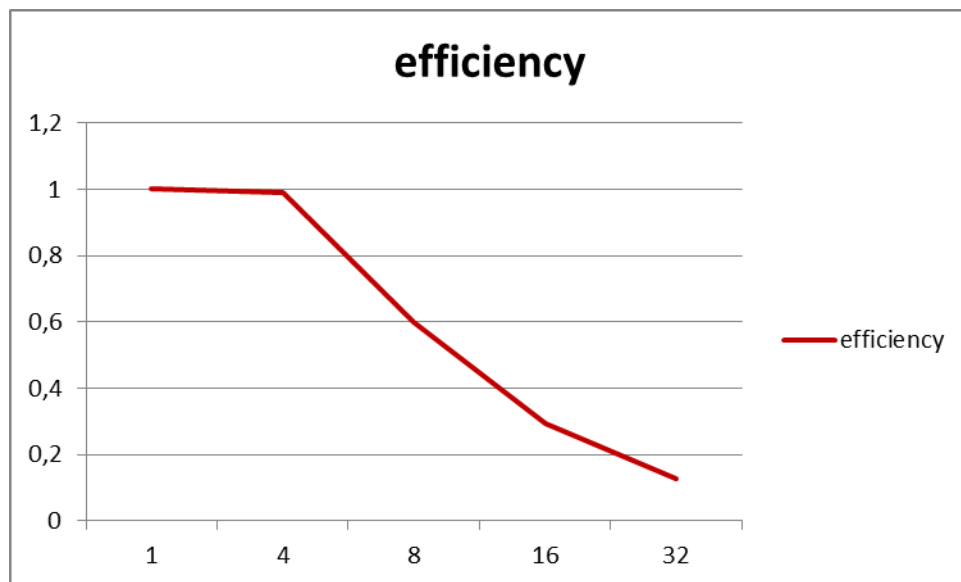
Ф и г у р а 4 В р е м е з а и з п ъ л н е н и е

На фигура 5 е показано ускорението при различен брой пчели.



Ф и г у р а 5 У с к о р е н и е

На фигура 6 е показана по графичен начин ефективността на алгоритъма.



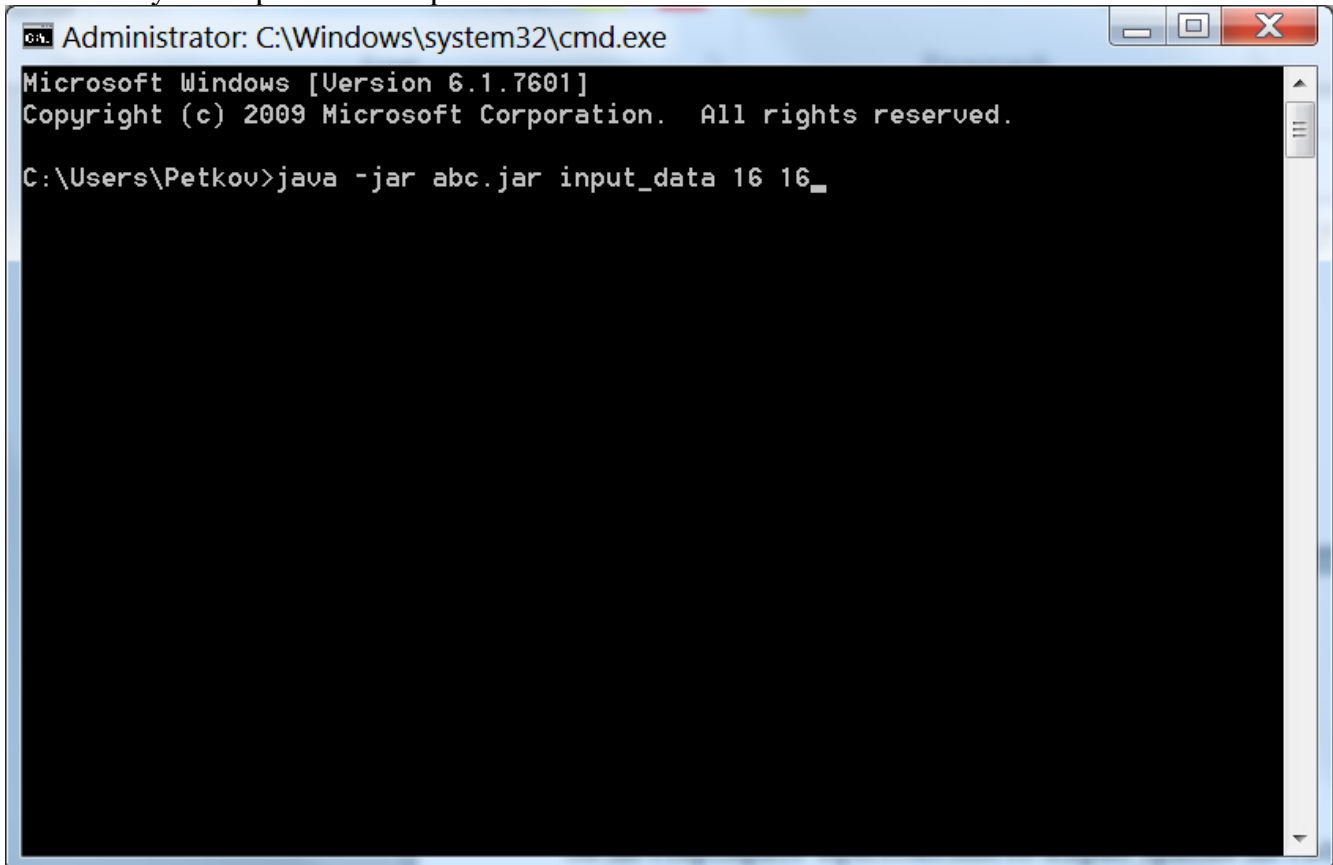
Ф и г у р а 6 Е ф е к т и в н о с т

5. Заключение

От направените експерименти можем да заключим, че при по-добър паралелизъм имаме по-голяма ускорение и по-добра ефективност. Алгоритъма Artificial Bee Colony може да се приложи успешно при проблеми, свързани с търсене в голямо количество информация с цял ускорение на процеса.

6. Ръководство на потребителя

За да стартирате приложението първо трябва да създадете изпълним `jar` файл. Това лесно става през `eclipse` среда за разработка. За самото стартиране е нужно да напишем `java -jar` за стартиране на `java` изпълними файлове. След това трябва да подадем името на `jar` файла. Като параметри трябва да се подадат също пътят до файла с данни, броят пчели скаути и броят пчели работници.



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Petkov>java -jar abc.jar input_data 16 16_
```

7. Приложение

```
package bg.metaheuristic.abc.collection.set;

/**
 * Provides the functionality to return an element
 *
 *
 * @param <T>
 */
public interface ElementProvider<T> {
    public T get();
}

package bg.metaheuristic.abc.collection.set;

import java.util.HashSet;
import java.util.Iterator;

import bg.metaheuristic.abc.environment.resource.Resource;

/**
 * Implementation of HashSet that provides the functionality to return a random
 * element from the Set in synchronous way.
 *
 */
public class SynchronizedHashSetProvider extends HashSet<Resource> implements
    ElementProvider<Resource> {

    private static final long serialVersionUID = 1L;

    /**
     * Implementation of HashSet that provides the functionality to return a
     * random element from the Set in synchronous way.
     */
    public SynchronizedHashSetProvider() {
    }

    /**
     * Implementation of HashSet that provides the functionality to return a
     * random element from the Set in synchronous way.
     *
     * @param initialCapacity
     *         The initial capacity of the set
     */
    public SynchronizedHashSetProvider(final int initialCapacity) {
        super(initialCapacity);
    }

    /**
     * Returns a random element from the Set. The method is synchronized on the
     * current instance.
     */
}
```



```

@Override
public Resource get() {
    Resource element = null;

    synchronized (this) {
        final Iterator<Resource> iter = iterator();
        element = iter.next();
        iter.remove();
    }

    return element;
}
}

```

```

package bg.metaheuristic.abc.criteria;

```

```

import bg.metaheuristic.abc.environment.resource.Resource;

```

```

/**
 * Abstract class that is used to pass an acceptance criteria to the bees
 *
 *
 */

```

```

public abstract class Criteria {

    /**
     * Process the given data so the bee can decide what to do
     *
     * @param resource
     * @return
     */
    public abstract boolean process(final Resource resource);
}

```

```

package bg.metaheuristic.abc.environment;

```

```

import bg.metaheuristic.abc.environment.resource.Resource;

```

```

/**
 * Abstract class representing the environment where all the resources are
 * located. Here is stored the data that the bees are going to process
 *
 *
 */

```

```

public abstract class Environment {

    /**
     * Return a single resource from the environment
     *
     * @return
     */
    public abstract Resource getResource();

    /**

```

```

        * Adds a new resource to the current environment
        *
        * @param resource
        */
    public abstract void addResource(final Resource resource);
}

package bg.metaheuristic.abc.environment;

import bg.metaheuristic.abc.collection.set.SynchronizedHashSetProvider;
import bg.metaheuristic.abc.environment.resource.Resource;

/**
 * Simple implementation of the Environment that uses HashSet collection for
 * storing its resources
 *
 */
public class HashSetEnvironment extends Environment {

    private SynchronizedHashSetProvider resources;

    public HashSetEnvironment() {
        resources = new SynchronizedHashSetProvider();
    }

    @Override
    public Resource getResource() {
        Resource resource = null;

        synchronized (resources) {
            if (!resources.isEmpty()) {
                resource = resources.get();
            }
        }

        return resource;
    }

    /**
     * Adds the given resource to the environment
     *
     * @param resource
     */
    @Override
    public void addResource(final Resource resource) {
        if (resource != null) {
            resources.add(resource);
        }
    }
}

```

```
package bg.metaheuristic.abc.environment.resource;
```

```
/**
 * A basic representation of a resource
 *
 */
public abstract class Resource {

}
```

```
package bg.metaheuristic.abc.hive;
```

```
import java.util.HashSet;
import java.util.Set;
```

```
import bg.metaheuristic.abc.criteria.Criteria;
import bg.metaheuristic.abc.environment.Environment;
import bg.metaheuristic.abc.environment.resource.Resource;
import bg.metaheuristic.abc.hive.bee.Bee;
import bg.metaheuristic.abc.hive.bee.EmployeeBee;
import bg.metaheuristic.abc.hive.bee.ScoutBee;
import bg.metaheuristic.abc.hive.queue.OnLookersQueue;
import bg.metaheuristic.abc.util.Constants;
import bg.metaheuristic.log.Log;
```

```
/**
 * Class representing the bee hive.
 *
 */
```

```
public class Hive {

    private Set<Bee> employees;
    private Set<Bee> scouts;
    private OnLookersQueue onlookersQueue;
    private Environment environment;
    private Set<Resource> result;

    public Hive(final int employeesCount, final int scoutsCount,
                final Environment environment, final Criteria scoutCriteria,
                final Criteria employeeCriteria) {

        this.onlookersQueue = new OnLookersQueue();
        this.result = new HashSet<Resource>();
        this.environment = environment;

        initEmployeeBees(employeesCount, employeeCriteria);
        initScouts(scoutsCount, scoutCriteria);
    }

    /**
     * Fires up the show! The scout bees are send to search resources, employee
     * bees are working on the currently found resource, everything works like a
     * charm :))
     */
}
```

```

public void start() {
    sendScoutsToInvestigate();
    wakeUpEmployees();
    waitToFinish();
}

public void putResult(final Resource resource) {
    synchronized (result) {
        result.add(resource);
    }
}

/**
 * Creates the employee bees
 *
 * @param count
 * @param criteria
 */
private void initEmployeeBees(final int count, final Criteria criteria) {
    Log.info(Constants.LOG_RULE_THICK);
    Log.info("Init scout bees...");

    employees = new HashSet<Bee>(count);
    for (int i = 0; i < count; i++) {
        employees.add(new EmployeeBee("Employee_" + i, criteria, this));
    }

    Log.info("Done!");
}

/**
 * Creates the scout bees
 *
 * @param count
 * @param criteria
 */
private void initScouts(final int count, final Criteria criteria) {
    Log.info(Constants.LOG_RULE_THICK);
    Log.info("Init employee bees...");

    scouts = new HashSet<Bee>(count);
    for (int i = 0; i < count; i++) {
        scouts.add(new ScoutBee("Scout_" + i, environment, criteria, this));
    }

    Log.info("Done!");
}

/**
 * Stars all scouts and send them to investigate and filter resources
 */
private void sendScoutsToInvestigate() {
    Log.info(Constants.LOG_RULE_THICK);
    Log.info("Send scout bees...");

    for (Bee bee : scouts) {
        bee.start();
    }
}

```

```

    }

    Log.info("Done!");
}

/**
 * Wake up all the employee bees that are waiting for a propriate resource
 * to be processed
 */
private void wakeUpEmployees() {
    Log.info(Constants.LOG_RULE_THICK);
    Log.info("Wake up employee bees...");

    for (Bee bee : employees) {
        bee.start();
    }

    Log.info("Done!");
}

/**
 * Tell the current thread to wait all the bees to finish their work
 */
private void waitToFinish() {
    try {
        for (Bee scout : scouts) {
            scout.join();
        }

        for (Bee employee : employees) {
            employee.join();
        }
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    Log.info(Constants.LOG_RULE_THICK);
    for (Resource resource : result) {
        Log.info(Constants.LOG_RULE_THIN);
        Log.info(resource.toString());
    }

    Log.info(Constants.LOG_RULE_THICK);
    Log.info("Count : " + result.size());
}

/* Getters & Setters */

public OnlookersQueue getOnlookersQueue() {
    return onlookersQueue;
}
}

```

```

package bg.metaheuristic.abc.hive.bee;

import bg.metaheuristic.abc.criteria.Criteria;
import bg.metaheuristic.abc.hive.Hive;
import bg.metaheuristic.log.Log;

/**
 * This is an abstract class that represents a basic <b>Bee</b>
 * package bg.metaheuristic.abc.hive.bee;

import bg.metaheuristic.abc.criteria.Criteria;
import bg.metaheuristic.abc.environment.resource.Resource;
import bg.metaheuristic.abc.hive.Hive;
import bg.metaheuristic.log.Log;

/**
 * This is an implementation of the Bee class. It represents a worker that is
 * processing a resource
 *
 *
 */
public class EmployeeBee extends Bee {

    public EmployeeBee(final String name, final Criteria criteria,
                       final Hive hive) {
        super(name, criteria, hive);
    }

    @Override
    public void run() {
        Log.debug(getName() + " started!");

        while (true) {
            Resource resource = null;
            try {
                resource = hive.getOnlookersQueue().dequeue();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            if (resource == null) {
                break;
            }

            if (criteria.process(resource)) {
                hive.putResult(resource);
                Log.debug(getName() + " consumed!");
            }

        }

        Log.debug(getName() + " finished!");
    }
}

*
*/

```

```

public abstract class Bee extends Thread {

    protected Criteria criteria;

    protected Hive hive;

    public Bee(final String name, final Criteria criteria, final Hive hive) {
        super(name);

        this.criteria = criteria;
        this.hive = hive;

        Log.info("Bee " + name + " created!");
    }
}

```

```

package bg.metaheuristic.abc.hive.bee;

import bg.metaheuristic.abc.criteria.Criteria;
import bg.metaheuristic.abc.environment.Environment;
import bg.metaheuristic.abc.environment.resource.Resource;
import bg.metaheuristic.abc.hive.Hive;
import bg.metaheuristic.log.Log;

/**
 * This is an implementation of the Bee class. It represents a bee that searches
 * for a good resource
 *
 *
 */
public class ScoutBee extends Bee {

    private Environment environment;

    public ScoutBee(final String name, final Environment environment,
                    final Criteria criteria, final Hive hive) {
        super(name, criteria, hive);
        this.environment = environment;
    }

    @Override
    public void run() {

        Log.debug(getName() + " started");

        while (true) {
            Resource resource = null;

            resource = environment.getResource();

            if (resource != null) {

                if (criteria.process(resource)) {
                    Log.debug(getName() + " enqueue");
                    hive.getOnlookersQueue().enqueue(resource);
                }
            }
        }
    }
}

```

```

        }
    } else {
        hive.getOnlookersQueue().setExhausted();
        break;
    }
}

Log.debug(getName() + " finished!");
};
}

package bg.metaheuristic.abc.hive.queue;

import bg.metaheuristic.abc.environment.resource.Resource;
import bg.metaheuristic.log.Log;

/**
 * Queue representing the onlookers bees
 *
 *
 */
public class OnLookersQueue extends ResourceQueue {

    private boolean isExhausted;

    @Override
    public void enqueue(final Resource element) {
        synchronized (queue) {
            queue.offer(element);
            Log.debug(Thread.currentThread().getName() + " notifies");
            queue.notify();
        }
    }

    @Override
    public Resource dequeue() throws InterruptedException {
        synchronized (queue) {
            if (queue.isEmpty() && !isExhausted) {
                Log.debug(Thread.currentThread().getName()
                    + " starting to wait");
                queue.wait();
            }
            return queue.poll();
        }
    }

    public void setExhausted() {
        synchronized (queue) {
            isExhausted = true;
        }
    }
}

```

```

package bg.metaheuristic.abc.hive.queue;

```



```

import java.util.LinkedList;
import java.util.Queue;

import bg.metaheuristic.abc.environment.resource.Resource;

/**
 * Class representing a Queue of resources. Provides two basic methods for
 * adding and retrieving values to and from the queue
 *
 *
 */
public abstract class ResourceQueue {

    protected Queue<Resource> queue;

    public ResourceQueue() {
        this.queue = new LinkedList<Resource>();
    }

    /**
     * Adds the given resource to the queue
     *
     * @param resource
     */
    public void enqueue(final Resource resource) {
        synchronized (this.queue) {
            this.queue.offer(resource);
        }
    }

    /**
     * Gets the oldest element from the queue
     *
     * @return
     * @throws InterruptedException
     */
    public Resource dequeue() throws InterruptedException {
        synchronized (this.queue) {
            return this.queue.poll();
        }
    }
}

package bg.metaheuristic.abc.util;

/**
 * Holds application common constants
 *
 *
 */
public interface Constants {

    /* Logger */

    String LOG_RULE_THICK = "=====";

```

```

String LOG_RULE_THIN = "-----";

/* Resource generation */

int LIST_MIN_SIZE = 1000;
int LIST_MAX_SIZE = 100000;
int LIST_AVERAGE_SIZE = 10000;

int ENVIRONMENT_MIN_SIZE = 500;
int ENVIRONMENT_MAX_SIZE = 1000;

/* Persisting */

String FIELD_SEPARATOR = "\t";
}

```

```

package bg.metaheuristic.abc.util;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;

import bg.metaheuristic.abc.environment.Environment;
import bg.metaheuristic.abc.environment.HashSetEnvironment;
import bg.metaheuristic.abc.environment.resource.Resource;
import bg.metaheuristic.exmaple.resource.ListResource;

/**
 * Class holding static utility methods
 *
 */
public class Utils {

    /**
     * Private constructor to prevent outer instantiation
     */
    private Utils() {

    }

    public static long now() {
        return System.currentTimeMillis();
    }

    /**
     * Saves the environment to a file
     *
     * @param environment
     * @param filename
     */
}

```

```

public static void saveEnvironment(final Environment environment,
    final String filename) {
    try {
        PrintWriter writer = new PrintWriter(new File(filename));

        ListResource resource = (ListResource) environment.getResource();
        while (resource != null) {
            for (int value : resource.getValues()) {
                writer.print(value);
                writer.print(Constants.FIELD_SEPARATOR);
            }

            writer.println();

            resource = (ListResource) environment.getResource();
        }
        writer.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

/**
 * Loads the environment from a file
 *
 * @param filename
 * @return
 */
public static Environment loadEnvironment(String filename) {

    final Environment environment = new HashSetEnvironment();

    try {
        BufferedReader reader = new BufferedReader(new FileReader(filename));
        String line = null;
        while ((line = reader.readLine()) != null) {
            String[] elements = line.split(Constants.FIELD_SEPARATOR);
            List<Integer> values = new ArrayList<Integer>();
            for (String element : elements) {
                values.add(Integer.parseInt(element));
            }

            final Resource resource = new ListResource(values);
            environment.addResource(resource);
        }
        reader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return environment;
}
}

```

```

package bg.metaheuristic.exmaple.criteria;

import java.util.Collections;
import java.util.List;

import bg.metaheuristic.abc.criteria.Criteria;
import bg.metaheuristic.abc.environment.resource.Resource;
import bg.metaheuristic.exmaple.resource.ListResource;
import bg.metaheuristic.log.Log;

/**
 * Implementation of the Criteria class. Searches in the resource the number
 * <b>7</b>.
 *
 */
public class EmployeeCriteria extends Criteria {

    private static final int SEARCH_VALUE = 12942845;

    @Override
    public boolean process(final Resource resource) {

        final List<Integer> values = ((ListResource) resource).getValues();

        Collections.sort(values);

        boolean result = Collections.binarySearch(values, SEARCH_VALUE) > 0;

        Log.info("Result : " + result);
        return (result);
    }
}

```

```

package bg.metaheuristic.exmaple.criteria;

import bg.metaheuristic.abc.criteria.Criteria;
import bg.metaheuristic.abc.environment.resource.Resource;
import bg.metaheuristic.abc.util.Constants;
import bg.metaheuristic.exmaple.resource.ListResource;

/**
 * Implementation of the Criteria class. Checks if the given resource size is
 * larger than the average size.
 *
 */
public class ScoutCriteria extends Criteria {

    @Override
    public boolean process(Resource resource) {

```

```

        return ((ListResource) resource).getValues().size() >
Constants.LIST_AVERAGE_SIZE;
    }

}

```

```

package bg.metaheuristic.exmaple.generator;

```

```

import java.util.Random;

```

```

/**
 * Generates a dummy values
 *
 *
 * @param <T>
 */

```

```

public abstract class DummyGenerator<T> {

```

```

    /**
     * Generates a random int number between the given bounds
     *
     * @param min
     * @param max
     * @return
     */

```

```

    protected int randomInt(final int min, final int max) {

        final Random randomGenerator = new Random();
        final int randomNumber = randomGenerator.nextInt(max - min);

        return (min + randomNumber);
    }

```

```

    /**
     * Generates a random int number
     *
     * @return
     */

```

```

    protected int randomInt() {
        final Random randomGenerator = new Random();
        return randomGenerator.nextInt();
    }

```

```

    /**
     * Generates a dummy object with random size
     *
     * @return
     */

```

```

    public abstract T generate();

```

```

}

```

```

package bg.metaheuristic.exmaple.generator;

```

```

import bg.metaheuristic.abc.environment.Environment;

```

```

import bg.metaheuristic.abc.environment.HashSetEnvironment;

```

```

import bg.metaheuristic.log.Log;

public class EnvironmentGenerator extends DummyGenerator<Environment> {

    private int environmentSize;

    @Override
    public Environment generate() {

        final Environment environment = new HashSetEnvironment();

        final ResourceGenerator resourceGenerator = new ResourceGenerator();

        Log.info("Size : " + environmentSize);

        for (int i = 0; i < environmentSize; i++) {
            Log.info("Index : " + i);

            environment.addResource(resourceGenerator.generate());
        }

        return environment;
    }

    /* Getters & Setters */

    public int getEnvironmentSize() {
        return environmentSize;
    }

    public void setEnvironmentSize(int environmentSize) {
        this.environmentSize = environmentSize;
    }
}

```

```

package bg.metaheuristic.exmaple.generator;

```

```

import java.util.ArrayList;
import java.util.List;

```

```

import bg.metaheuristic.abc.environment.resource.Resource;
import bg.metaheuristic.abc.util.Constants;
import bg.metaheuristic.exmaple.resource.ListResource;
import bg.metaheuristic.log.Log;

```

```

/**
 * Generates a resource object filled up with dummy values
 *
 *
 */

```

```

public class ResourceGenerator extends DummyGenerator<Resource> {

    private int generateListSize() {
        return randomInt(Constants.LIST_MIN_SIZE, Constants.LIST_MAX_SIZE);
    }
}

```

```

    }

    @Override
    public Resource generate() {

        final int size = generateListSize();
        return generate(size);
    }

    // @Override
    public Resource generate(final int size) {

        Log.info("Size : " + size);

        final List<Integer> list = new ArrayList<Integer>(size);
        final Resource resource = new ListResource(list);

        for (int i = 0; i < size; i++) {
            list.add(randomInt(0, 100000000));
        }

        return resource;
    }
}

```

```

package bg.metaheuristic.exmaple.main;

```

```

import bg.metaheuristic.abc.environment.Environment;
import bg.metaheuristic.abc.util.Constants;
import bg.metaheuristic.abc.util.Utills;
import bg.metaheuristic.exmaple.generator.EnvironmentGenerator;
import bg.metaheuristic.log.Log;

```

```

/**
 * The main class of the application
 *
 *
 */

```

```

public class Generator {

```

```

    /**
     * The generator entry point
     *
     * @param args
     */

```

```

    public static void main(String[] args) {

```

```

        if (args.length == 2) {
            final String filename = args[0];
            final int environmentSize = Integer.parseInt(args[1]);

            Log.info(Constants.LOG_RULE_THICK);
            Log.info("Generating environment with size " + environmentSize);

            final EnvironmentGenerator generator = new EnvironmentGenerator();

```

```

        generator.setEnvironmentSize(environmentSize);
        final Environment environment = generator.generate();

        Log.info("Environment generated!");

        Log.info("Saving to " + filename + "...");
        Utils.saveEnvironment(environment, filename);
        Log.info("Environment saved!");
    } else {
        Log.info("Please provide a single argument that is the filename of the
file to be processed!");
    }
}
}

```

```

package bg.metaheuristic.exmaple.main;

```

```

import bg.metaheuristic.abc.criteria.Criteria;
import bg.metaheuristic.abc.environment.Environment;
import bg.metaheuristic.abc.hive.Hive;
import bg.metaheuristic.abc.util.Utils;
import bg.metaheuristic.exmaple.criteria.EmployeeCriteria;
import bg.metaheuristic.exmaple.criteria.ScoutCriteria;
import bg.metaheuristic.log.Log;

```

```

/**
 * The main class of the application
 *
 *
 */

```

```

public class Main {

```

```

    /**
     * The entry point of the application
     *
     * @param args
     */

```

```

    public static void main(String[] args) {

```

```

        if (args.length == 3) {
            final int scoutCount = Integer.parseInt(args[0]);
            final int employeeCount = Integer.parseInt(args[1]);
            final String filename = args[2];

            final Environment environment = Utils.loadEnvironment(filename);
            final Criteria scoutCriteria = new ScoutCriteria();
            final Criteria employeeCriteria = new EmployeeCriteria();

            final Hive hive = new Hive(employeeCount, scoutCount, environment,
                scoutCriteria, employeeCriteria);

            long start = Utils.now();
            hive.start();
            long end = Utils.now();

```



```

        System.out.println("Time for execution : " + (end - start));
    } else {
        Log.info("Please provide three arguments : scouts couts, employees
count, filename!");
    }
}
}

```

```

package bg.metaheuristic.exmaple.resource;

```

```

import java.util.List;

```

```

import bg.metaheuristic.abc.environment.resource.Resource;

```

```

/**
 * Implementation of the resource class. Holds a bunch of values in a list
 * collection.
 *
 *
 */

```

```

public class ListResource extends Resource {

    private List<Integer> values;

    public ListResource(final List<Integer> values) {
        this.values = values;
    }

    public List<Integer> getValues() {
        return values;
    }

    public void setValues(List<Integer> values) {
        this.values = values;
    }
}

```

```

package bg.metaheuristic.log;

```

```

/**
 * Implementation of the Logger class. Logs everything to the standard output
 *
 */

```

```

public class ConsoleLogger extends Logger {

    public ConsoleLogger(final LogLevel logLevel) {
        super(logLevel);
    }

    /**
     * Logs the message in the standard output
     */
    @Override
    protected void logMethod(final String message, final LogLevel prefix) {

```

```

        System.out.println(prefix + message);
    }
}

```

```
package bg.metaheuristic.log;
```

```
import bg.metaheuristic.log.Logger.LogLevel;
```

```

/**
 * This class provides a public access to all logger methods. This way it is
 * used a single logger for the whole application that is accessed in a static
 * way.
 *
 */
public class Log {
    private static Logger logger = new ConsoleLogger(LogLevel.INFO);

    /**
     * Prevent outer instantiation
     */
    private Log() {

    }

    public static void info(String message) {
        logger.info(message);
    }

    public static void debug(String message) {
        logger.debug(message);
    }

    public static void warn(String message) {
        logger.warn(message);
    }

    public static void error(String message) {
        logger.error(message);
    }

    public static void fatal(String message) {
        logger.fatal(message);
    }
}

```

```
package bg.metaheuristic.log;
```

```

/**
 * Abstract logger class. Holds a couple of methods for logging in a different
 * logging level. Should be extended and the child class should implement the
 * <b>logMethod</b> method. This method is called for logging the messages.

```

```

*
*
*/
public abstract class Logger {
    /**
     * Represent the level of logging.
     *
     */
    protected static enum LogLevel {
        INFO("I : ", 5), DEBUG("D : ", 4), WARN("W : ", 3), ERROR("E : ", 2), FATAL(
            "F : ", 1);

        private String prefix;

        private int weight;

        private LogLevel(final String prefix, final int weight) {
            this.prefix = prefix;
            this.weight = weight;
        }

        /**
         * Checks if the current log level allows logging messages with the
         * given log level.
         *
         * @param logLevel
         *         LogLevel to be checked
         * @return <b>true</b> or <b>false</b>
         */
        public boolean allows(LogLevel logLevel) {
            return (this.weight >= logLevel.weight);
        }

        @Override
        public String toString() {
            return prefix;
        }
    }

    public Logger(final LogLevel logLevel) {
        this.logLevel = logLevel;
    }

    protected LogLevel logLevel;

    public void info(final String message) {
        log(message, LogLevel.INFO);
    }

    public void debug(final String message) {
        log(message, LogLevel.DEBUG);
    }

    public void warn(final String message) {
        log(message, LogLevel.WARN);
    }
}

```

```

    public void error(final String message) {
        log(message, LogLevel.ERROR);
    }

    public void fatal(final String message) {
        log(message, LogLevel.FATAL);
    }

    private void log(final String message, final LogLevel prefix) {
        if (logLevel.allows(prefix)) {
            logMethod(message, prefix);
        }
    }

    /**
     * This method is called when a message is logged. It should be overridden
     * in the child class. If you want to log the messages on a remote server or
     * in the standard output or a file - this is the place where you should do
     * this.
     *
     * @param message
     * @param prefix
     */
    protected abstract void logMethod(final String message,
                                       final LogLevel prefix);

    /* Getters && Setters */

    public LogLevel getLogLevel() {
        return logLevel;
    }
}

```