Design a Tool for Generating Test Cases using Swarm Intelligence

Shahbaa I. Khaleel **Ashraf Abdulmunim Al Thanoon**
shahbaaibrkh@uomosul.edu.iq
College of Computer Sciences and Mathematics
University of Mosul, Mosul, Iraq

Received on: 24/10/2012 **Accepted on: 30/01/2013**

ABSTRACT

In this research, the tools and techniques of artificial intelligence were studied and employed in software engineering. And that was conducted through using the Particle Swarm Optimization PSO and Cat Swarm Optimization CSO in generating optimal test cases of the software written with C++ language in an automatic way because that enables the corporation which develops the program to save time and costs as well as ensuring the test process quality, which is estimated by 50% of the product cost. In this research, the software engineering tool Generate Test Suite GTS TOOL was constructed and modeled with the aid of the computer, which is used to generate optimal test cases automatically and this tool also support the drawing of the control flowgraphs and paths inside the program and tests each path using CSO and PSO. The proposed tool succeeded in generating optimal test cases for several programs and in a very short time. The average of generating the test cases using PSO was 4 minutes and 1.2 minutes for CSO. Where the performance of the CSO was much better than the performance of PSO.

Keywords: Particle Swarm Optimization PSO , Cat Swarm Optimization CSO, control flowgraphs, test cases.

تصميم أداة لتوليد حالات الاختبار باعتماد ذكاء السرب

شهباء إبراهيم خليل **أشرف عبد المنعم عبد المجيد**
كلية علوم الحاسوب والرياضيات
جامعة الموصل، الموصل، العراق

تاريخ استلام البحث: 2012/10/24 **تاريخ قبول البحث: 2013/01/30**

الملخص

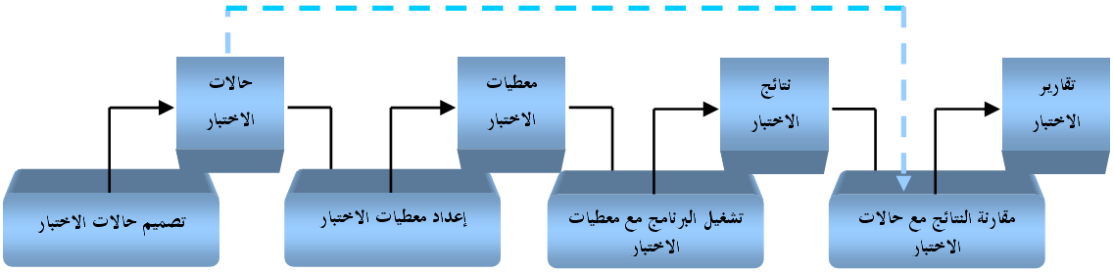
في هذا البحث تمت دراسة آليات الذكاء الاصطناعي وتقنياتها بغية توظيفها في خدمة هندسة البرمجيات، وقد أنجز ذلك من خلال استخدام خوارزمية سرب الطيور Particle swarm optimization PSO وخوارزمية سرب القطط cat swarm optimization CSO في توليد حالات الاختبار المثالية للبرمجيات المكتوبة بلغة C++ وبشكل تلقائي مما سيوفر للمؤسسة المطورة للبرمجيات الوقت والكلفة فضلاً عن ضمان جودة عملية الاختبار التي تقدر عادة بنحو 50% من كلفة إنتاج المنتج. وقد تم في هذا البحث نمذجة أداة هندسة البرمجيات بمساعدة الحاسوب Generate Test Suite GTS TOOL وبناؤها التي تستخدم في توليد حالات اختبار مثالية بشكل تلقائي وتدعم هذه الأداة أيضاً رسم مخطط تدفق السيطرة والمسارات داخل البرنامج فضلاً عن حالة الاختبار لكل

مسار باستخدام، PSO وCSO. وقد نجحت الأداة المقترحة في توليد حالات اختبار مثالية لعدة برامج وبوقت قصير جداً، إذ كان معدل توليد حالات الاختبار باستخدام PSO 4 دقائق والـ CSO 1.2 دقيقة، حيث كان أداء الـ CSO أفضل بكثير من أداء الـ PSO.

الكلمات المفتاحية : أمثلية سرب الطيور، أمثلية سرب القطط، مخطط تدفق السيطرة، حالات الاختبار.

1- المقدمة

أصبحت البرمجيات في العصر الحالي واسعة الانتشار وموجودة في كل مكان، ويتطلب هذا الوجود أو الانتشار اختباراً شاملاً للبرامج التي يتم تطويرها، والاختبار البرمجي هو مهمة استهلاكية ومكلفة، إذ انه يستهلك تقريباً *50%* من مصادر تطوير النظام البرمجي.وتصنف تقنيات الاختبار التقليدية إلى صنفين: اختبار الصندوق الأبيض واختبار الصندوق الأسود. في تقنيات اختبار الصندوق الأسود تتولد حالات الاختبار من متطلبات رسمية وغير رسمية. في حين يمثل اختبار الصندوق الأبيض طريقة لاختبار الوظائف الخارجية من الشفرة بواسطة فحص واختبار الشفرة التي تحقق الوظيفة الخارجية [2]. ويمكن للاختبار أن يحصل يدويا أو أوتوماتيكيا باستخدام أدوات الاختبار. وقد وجد بان الاختبار البرمجي الأوتوماتيكي هو أفضل من الاختبار اليدوي. وان عدداً قليلاً جدا من أدوات توليد بيانات الاختبار يكون متوفرا تجاريا في الوقت الحالي. ويعد الاختبار التطويري منهجية صلدة لتوليد بيانات اختبار بجودة عالية وبشكل أوتوماتيكي [2]. يوضح الشكل (1) الأنموذج العام لعملية الاختبار [1].



الشكل (1). الأنموذج العام لعملية الاختبار

2- الأعمال السابقة

في عام *2000* قدم الباحثان *Nguyen Tran Sy, Yves Deville* طريقة جديدة لتوليد بيانات الاختبار أوتوماتيكيا للبرامج الضرورية الحاوية على متغيرات صحيحة ، منطقية وعشرية. والأسلوب مبني على أساس تقنيات التماسك *consistency techniques* التي تدمج متغيرات صحيحة وعشرية[5]. قام الباحثون كل من *Joachim Wegener , Stefan Wappler , Andreas Windisch* عام *2007* بعمل مقارنة تجريبية بين عمل الخوارزمية الجينية و*particle swarm optimization (PSO)* في اختبار البرمجيات *software testing [7]*. في هذا البحث تم تقديم تقنية عامة وجديدة من قبل *Abdelaziz M. Khamis , Moheb R . Girgis , Ahmed S. G hiduk* عام *2007* تدمج بين مفهوم مجاميع الامتداد *spanning sets* والخوارزمية الجينية *GA* لتوليد بيانات اختبار بشكل أوتوماتيكي لتغطية مجاميع الامتداد[8]. قدم *Ankur Pachauri and Gursaran* عام *2012* تعريف صيغة زيادة بديلة *alternate maximization formulation* وتمت مقارنتها مع صيغة النقصان *minimization formulation* وتم استخدام *GA* و*PSO* كتقنيات بحث وإضافة للعوامل الاعتيادية[10] .

3- الدوافع لإجراء عملية الاختبار

يمكن تحسين أداء العمليات عن طريق فهم واستيعاب الأسباب الجذرية لحدوث الخلل والأخطاء المكتشفة في المشاريع وهذا بدوره يجب أن يمنع تكرار حدوث الخلل وبالتالي تحسين جودة الأنظمة المستقبلية [3]. الاختبار الفعال قبل نشر المنتج يحقق ثلاثة منافع رئيسة وهي:

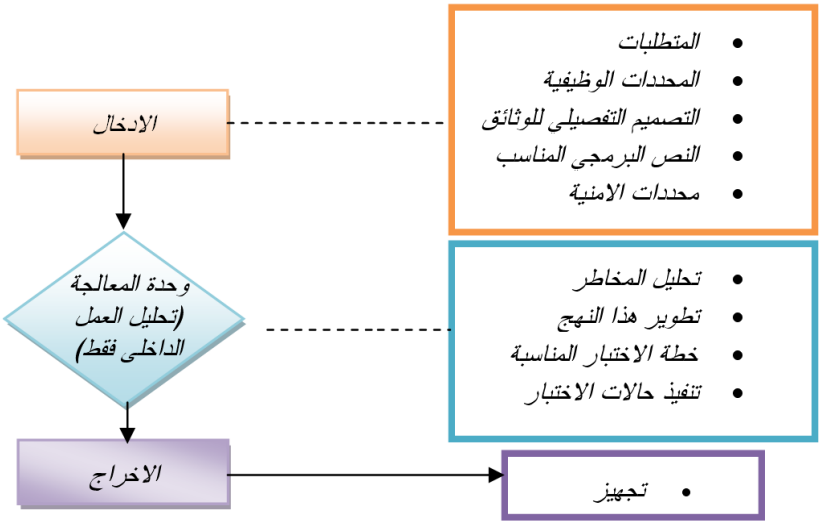
- اكتشاف الخلل والأخطاء قبل نشر التطبيق يسمح بإصلاحها قبل أن تؤثر على الأعمال مما يقلص من تعطيل وإرباك العمل بسبب فشل البرنامج أو حدوث الأخطاء ويقلص من كلفة إصلاح الخلل.
- يمكن من تخمين أو تقدير مدى بقاء الخلل في البرنامج واستخدام تلك التخمينات لتقرير متى يلبي البرنامج مستوى أو معيار الوثوقية لنشر المنتج.
- نتائج الاختبار تساعد على تحديد مواطن القوة ومواطن الضعف أو النقص في عمليات التطوير وعمل تحسينات للعملية التي تحسن من أداء البرنامج الناتج [3].

4- اختبار الصندوق الأبيض

يرتكز اختبار الصندوق الأبيض على تحليل الهيكل الداخلي للبرنامج. واختبار الصندوق الأبيض هو عملية إعطاء مدخل إلى النظام وفحص كيفية قيام الإدخال بتوليد الإخراج المطلوب[4]. هذا الاختبار يأخذ في الحسبان الآلية الداخلية للنظام أو المكونات. الشكل (2) يوضح تقدم العمل في اختبار الصندوق الأبيض ويبين الجدول (1) التقنيات لهذا الاختبار[6].

الجدول (1). تقنيات اختبار الصندوق الأبيض

White Box Testing	*اختبارات الصندوق الأبيض*
Loop Testing	*اختبار الدارة البرمجية*
Branch Testing	*الاختبار الفرعي*
Data Flow Testing	*اختبار تدفق البيانات*
Control Flow Testing	*اختبار تدفق السيطرة*
Basis Path Testing	*اختبار المسار الأساسي*



الشكل (2). تقدم العمل في اختبار الصندوق الأبيض

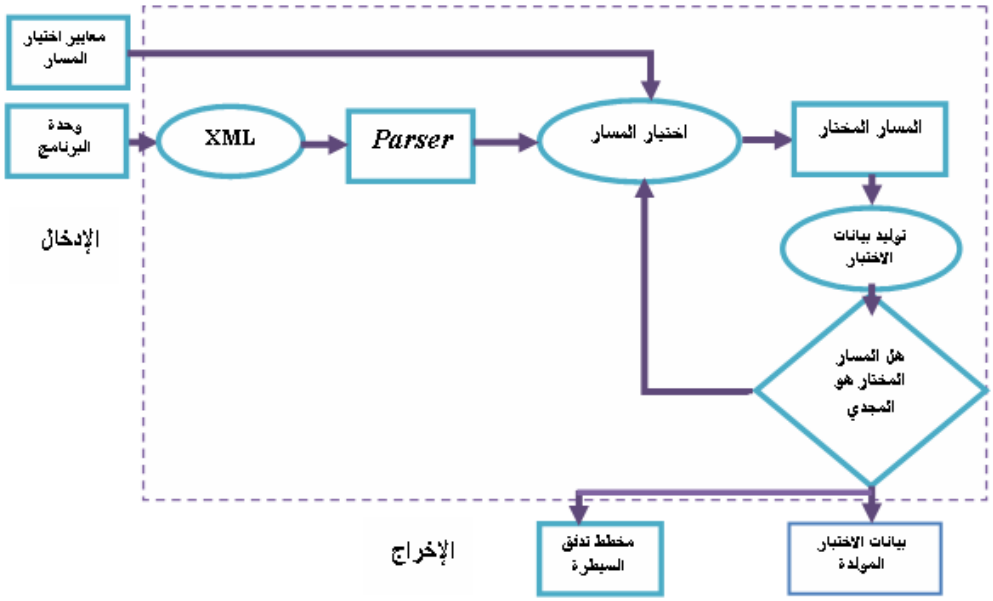
يكون اختبار الصندوق الأبيض أكثر كلفة وأغلى بكثير من اختبار الصندوق الأسود. ويتطلب إجراؤه توليد الشفرة المصدرية قبل التخطيط لإجراء الاختبار ويتطلب جهدا وعملا اكبر بكثير في تعيين بيانات الإدخال المناسبة وتحديد فيما إذا كانت البرمجيات صحيحة أو غير صحيحة [7].

5- اختبار البرمجيات التلقائي أو الآلي

الاختبار التلقائي يستخدم البرمجيات لتوليد حالات الاختبار وتنفيذها ومقارنة النتائج ويسجل الأخطاء من دون التدخل البشري. وقد جذب الاختبار التلقائي الكثير من الانتباه إليه في الوقت الحاضر كوسيلة لتقليص كلفة الاختبار واكتشاف أخطاء أكثر وتوفير الوقت الثمين. وبكلمات ابسط هو "كتابة شفرة لاختبار شفرة". معظم عمليات الاختبار التلقائية تُنفذ باستخدام أدوات اختبار خاصة وتتطلب مهارات أكثر تخصصاً من المهارات التي تتطلبها الاختبارات اليدوية [8]. الاختبار الشامل يمكن تنفيذه فقط بواسطة جعل عملية الاختبار أوتوماتيكيا. المنافع المستحصلة منه هي تقليص الوقت المستهلك وتقليص الجهود وكلفة الاختبار البرمجي. وتتكون أدوات الاختبار التلقائي بشكل عام من منفذ الأداة و مهمات الاختبار ومولد بيانات الاختبار [9] .

6- التكوين التلقائي لحالات الاختبار

هناك أسلوب واحد لتوليد حالات الاختبار وهو الاختيار العشوائي لبيانات الاختبار إلى أن يتم الاقتناع بالمعيار المرغوب به أي الحصول على أفضل حالة اختبار. وهذا ما ينتج عنه الكثير من حالات الاختبار الفائضة عن الحاجة لأن الكثير من حالات الاختبار سوف تتمرس أو تفحص نفس المسارات [10]. لتوليد حالات الاختبار تلقائيا يجب تقرير ما يلي حالة الاختبار، مخطط الحالة لرسم المسار الأولي، المسار الثانوي، مسار الانتقال، الحد أو الحافة، نطاق المسار [11]. الغرض من إنشاء مخطط تدفق السيطرة *CFG* هو تمثيل مخططي مفصل لوحدة البرنامج. الفكرة من وراء رسم الـ *CFG* هو كي يتم رؤية أو تصور كل المسارات في وحدة البرنامج. إذا حدث تشغيل عملية توليد الاختبار أوتوماتيكيا فان إحدى المترجمات يمكن تعديلها لإنتاج الـ *CFG* [12] . والشكل (3) يبين عملية توليد حالات الاختبار.



الشكل (3). عملية توليد حالة الاختبار

الإدخالات في عملية توليد الاختبار

- الشفرة المصدرية لوحدة البرنامج
- معيار اختيار المسار : الحالة ، الفرع.

إنشاء XML

- تحويل البرنامج إلى XML.
- تحليل المسارات التي تم الحصول عليها من تقرير xml.

اختيار المسارات

- يجري اختيار المسارات من الـ *CFG* ليلبي معيار اختيار المسار ويجري انجازه بوساطة ملاحظة تركيبة الـ *CFG*.

اختبار قابلية التنفيذ للمسار

- الفكرة من وراء فحص قابلية التنفيذ لمسار جرى اختياره هو لمقابلة أو ملاءمة معيار اختيار المسار. وهناك نوعان من المسارات:

  - مسار قابل للتنفيذ: مدخلاته موجودة ولهذا فان المسار يتم تنفيذه.
  - مسار غير قابل للتنفيذ: ليس لديه مدخلات لتنفيذ المسار.

7- تنفيذ حالة الاختبار

الخطوة التالية في عملية الاختبار هي تنفيذه. إذ يجري تكوين برنامج متكامل لاختبار النظام بعد الانتهاء من مرحلة بناء حالات الاختبار. مواصفات حالة الاختبار تحدد فقط مجموعة من حالات الاختبار للوحدة التي سيجرى اختبارها. لكن تنفيذ حالات الاختبار تتطلب وحدات متكاملة لإقامة البيئة كما هو محدد في خطة الاختبار ومواصفات حالة الاختبار[10].

8- ذكاء السرب

منذ أكثر من خمسين عاماً اثبت علماء البيولوجيا وجود أنواع متعددة من أشكال الذكاء منبثقة من مجتمع الحشرات والأسماك والطيور أو الثدييات. داخل كثيب النمل وأسراب النمل الأبيض ومستعمرات النحل وأسراب الطيور والأسماك لا يمتلك الفرد الواحد القدرة العصبية الضرورية. على الرغم من ذلك فان التفاعل المجرد البسيط بين عدد كبير من المخلوقات البسيطة يمكن أن يقود إلى نشوء ذكاء متفاعل ومتأقلم مع البيئة المحيطة. وفي مجتمعات الحشرات يكون النظام بأكمله منظماً في أنموذج لامركزي. الكثير من الوحدات المستقلة بذاتها التي تملك سلوكاً بسيطاً احتمالياً نسبياً يجري توزيعها في البيئة. كل وحدة مزودة فحسب بالمعلومات المحلية. ولا تمتلك هذه الوحدات أي تمثيل أو معرفة واضحة بالتركيبة الشاملة التي من المفترض أن تقوم بإنتاجها أو تطويرها. ولا أية خطة على الإطلاق. أي بكلمات أخرى إن المهمة الشاملة ليست مبرمجة بشكل واضح من خلال الأفراد ولكنها تنبثق بعد نجاح عدد كبير من التفاعلات الأحادية بين الأفراد أو بين الأفراد والبيئة. هذا الأنموذج من الذكاء الجماعي الذي يتم بناؤه العديد من الكيانات المنفردة البسيطة كان هو الملهم لنظام جديد في علوم الكمبيوتر وهو ذكاء السرب[13].

1-8 خوارزمية سرب الطيور *PSO*

إن اختيار عوامل الـ *PSO* الجيدة بطريقة سهلة جرى تقديمها من قبل *Pedersen et al 2000* والذي قام أيضاً بعدد كبير من التجارب مع مسائل الأمثلية المتنوعة والإعدادات. وقد سميت توليف عوامل الـ*PSO* بالأمثلية المتحول *Meta – optimization* لأن هناك طريقة أمثلية أخرى بأسلوب التغطية لتوليف عوامل الـ *PSO* [14]. وقد طورت خوارزمية سرب الطيور من قبل *Kennedy1995* و *Eberhart* وهي تحاكي السلوك الاجتماعي لأسراب الطيور والأسماك والطرائق التي تستخدمها هذه الأسراب لإيجاد المأوى والمسكن ومصادر الغذاء أو موطن آخر ملائم لها.وتفترض تقنية الـ *PSO* الأساسية إن فضاء البحث هو *d-dimensional* [5].

- كل عضو يُسمى بالجسيم، وكل جسيم *i-th particle* يمثله متجه موقعي *d-dimensional* ويُوصف كالآتي $X_i = [\, x_{i1}, x_{i2}, \ldots\ldots x_{1d} \,]$.
- مجموعة جسيمات n في السرب تُسمى الأفراد.
- أفضل موقع سابق للجسيم (الموقع الذي يعطي أفضل قيمة لياقة) يُسمى الجسيم الأفضل ويُوصف: $PB_i = [pb_{i1}, pb_{i2}, \ldots\ldots\ldots\ldots, pb_{id}]$
- أفضل موقع من بين أفضل مواقع الجسيم المتحققة حتى الآن يُسمى الشامل الأفضل *Global Best* ويُوصف: $GB_I = [gb_{i1}, gb_{i2}, \ldots\ldots\ldots\ldots, gb_{id}]$
- نسبة تغير الموقع لكل جسيم يُسمى سرعة الجسيم ويُوصف: $V_I = [v_{i1}, v_{i2}, \ldots\ldots\ldots\ldots, v_{id}]$

عند التكرار K ، فان السرعة لـ *d-dimensional* لـ *i-th particle* يتم تحديثه باستخدام المعادلة (1) [15].

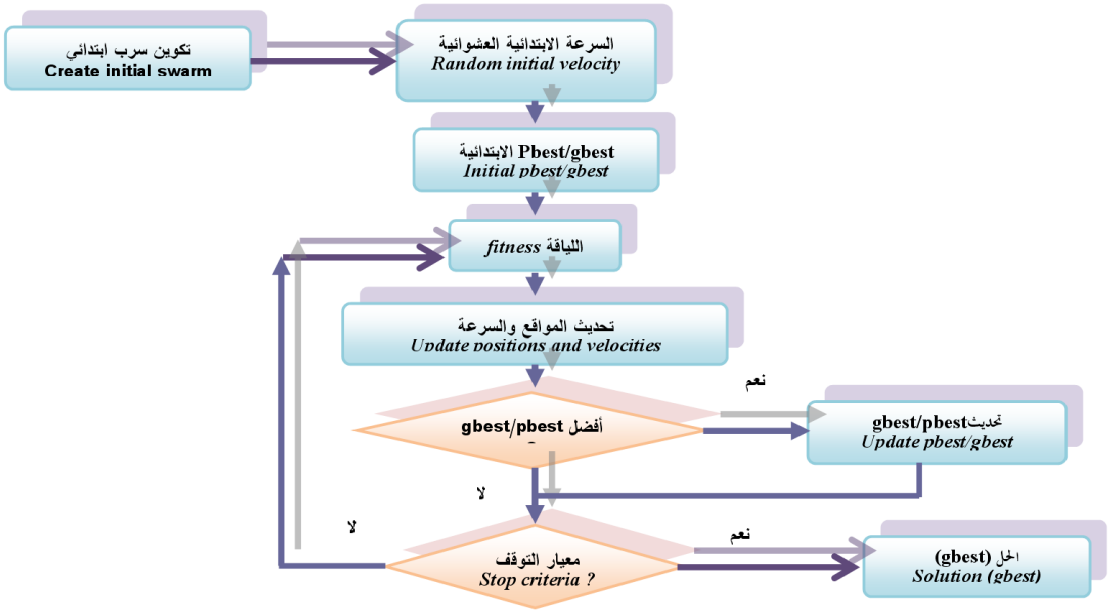
$$V_{id}(K+1) = wv_{id}(k) + c_1 r_1 (pb_{id}(k) - x_{id}(k)) + c_2 r_2 (gb_{id}(k) - x_{id}(k)) \quad \ldots(1)$$

حيث $i=1,2\ldots.$ و n هو حجم الأفراد ، حيث w هو عامل القصور الذاتي *inertia weight* ، r_1 وr_2 هما الأرقام العشوائية التي تستخدم للحفاظ على تنوع الأفراد ويتم توزيعها بانتظام بين [1,0] لبعد الـ *j-th* لجسيم الـ *i-th*، الـ c_1 هو عدد صحيح موجب يُسمى معامل مكون التمييز الذاتي، الـ c_2 هو أيضاً ثابت موجب يُسمى معامل المكون الاجتماعي من المعادلة (1). يحدد الجسيم إلى أين يتحرك الحركة التالية اعتماداً على خبرته الشخصية التي هي ذاكرة أفضل موقع سابق وخبرة الجسيم الأكثر نجاحاً في السرب [14].

- موقع *i-particle* يتم تحديثه بواسطة المعادلة (2) [15]:

$$x_{id}(k+1) = x_{id}(k) + V_{id}(k+1) \quad \ldots(2)$$

المتغير w ينظم عملية الاستبدال بين قابليات الاستكشاف الشاملة العالمية " الواسعة النطاق " للسرب وقابليات الاستكشاف المحلية "القريبة " للسرب. وزن القصور الذاتي الكبير يسهل الاستكشاف الشامل في حين الصغير يميل إلى تسهيل الاستكشاف المحلي الشامل. القيمة المناسبة لوزن القصور الذاتي عادةً تقدم موازنة بين قابليات الاستكشاف العالمية والمحلية وبالنتيجة تؤدي إلى تقليص عدد التكرارات المطلوبة لتحديد موقع الحل الأفضل. في البداية وزن القصور الذاتي هو مجموعة من الثوابت. وهكذا فان القيمة الأولية هي بحدود 1,2 وتتقلص بالتدريج باتجاه الصفر الذي يعد قيمة جيدة لـ w. المتغيرات $c_1 = c_2 = 2$ يمكن أن تُحدد كقيم قياسية. ويشير عدد من نتائج التجارب إلى أن $c_1 = c_2 = 1.49$ قد يقدم نتائج أفضل، قيمة حجم السرب قد تكون 20. المعادلة (1) تصف كيفية تحديث السرعة ديناميكيا. المعادلة (2) تحديث موقع الجسيمات الطائرة [16].



الشكل (4). خوارزمية سرب الطيور

2-8 خوارزمية سرب القطط

في العام *2006* قدمت خوارزمية أساسية في ذكاء السرب من قبل *Che et. al*. وقد عُرفت بخوارزمية سرب القطط أو الـ *CSO*. وكما يتبين من اسمها بشكل واضح فان هذه الخوارزمية تستند إلى سلوك القطط. واعتماداً على مراقبة القطط فقد قسم سلوكها بشكل عام على قسمين. هذان السلوكان هما السلوك الساكن اليقظ وسلوك الحركة "المطاردة" النشط. وعلى الرغم من أنها تبدو كسولة وتقضي معظم وقتها في الراحة ولكن من خلال المراقبة والملاحظة أكثر دقةً تبين أنها تراقب محيطها "ما حولها" بتحمس بانتظار الحركة التالية التي سوف تتخذها[17]. خوارزمية سرب القطط *CSO* هي إحدى خوارزميات الأمثلية الاستدلالية الجديدة المرتكزة على ذكاء السرب. وقد أظهرت الأبحاث السابقة أن هذه الخوارزمية لها أداء أفضل إذا ما قورنت بخوارزميات الأمثلية الاستدلالية الأخرى: كخوارزمية سرب الطيور *PSO* والـ *PSO* ذو الوزن *weighted – pso* في حالات تصغير الدالة [18] . في حالات تحسين الدالة فان الـ *CSO* هي الخوارزمية الأفضل في إيجاد أفضل حل شامل. وبالمقارنة مع الخوارزميات التجريبية الأخرى مثل الـ *PSO* والـ *weighted-PSO* ، فان الـ *CSO* عادةً تحقق نتائج أفضل. ولكن في بعض الأحيان وفي عدد من الحالات فان الـ *CSO* الخالصة تستغرق وقتاً طويلاً في إيجاد الحل المناسب. ولهذا فان هناك حاجة إلى معالج ذي سرعة عالية للحصول على نتائج معقولة [19]. الهدف في هذا البحث هو تقديم إصدار جديد *CSO* لتحسين الأداء وتحقيق مقاربة أفضل في اقل تكرار لتوليد بيانات الاختبار.

1-2-8 وصف نواة الـ *CSO*

من اجل دمج هذين الأسلوبين في خوارزمية يجب تعريف نسبة الامتزاج *MR* الذي يملي أوامر اتحاد أسلوب البحث مع أسلوب تعقب الأثر. ويقرر *MR* كم من القطط التي سوف يجري تحريكها بعملية أسلوب البحث. فعلى سبيل المثال، إذا كان حجم الأفراد هو *50* والـ *MR* مساوي لـ*0.7* فانه يجب أن يكون هناك *50×0.7=35*

قطاً متحركاً إلى أسلوب البحث و15 قطاً متحركاً إلى أسلوب التعقب في هذا التكرار. اذن عملية الـ *CSO* يمكن تلخيصها كالآتي[19]:

أولاً – إنشاء *N* من القطط وتحديد قيم ابتدائية للموقع، والسرع المؤشر لكل قط.

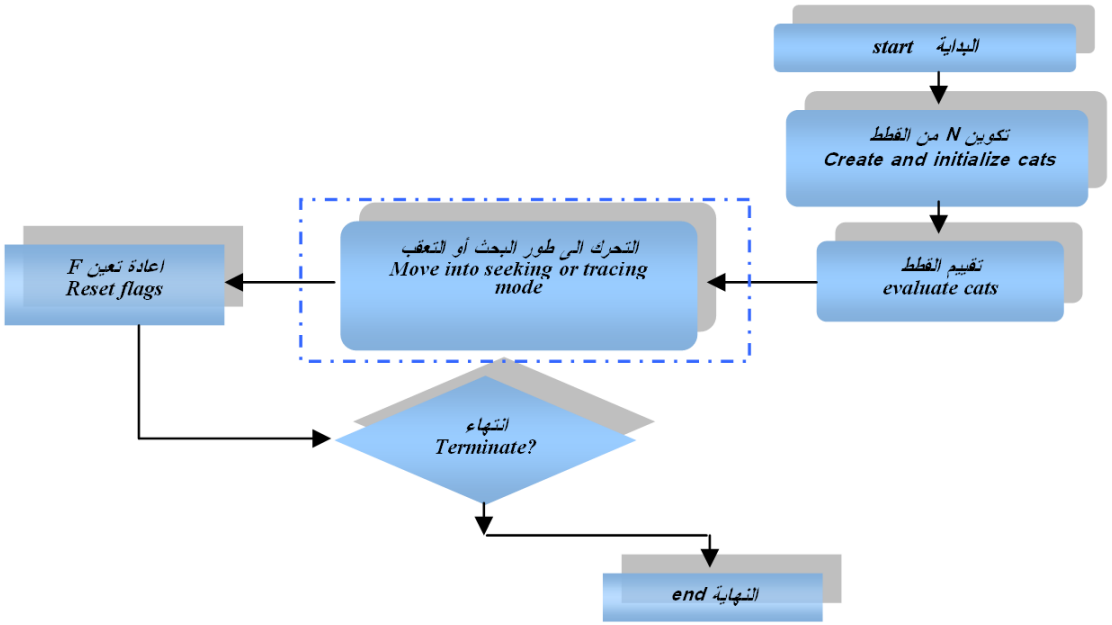
ثانياً – تخمين قيمة اللياقة لكل قطة على حسب دالة اللياقة والاحتفاظ بأفضل قط في الذاكرة.

ثالثاً – يجري تطبيق القط في مجال البحث أو طور التعقب على حسب مؤشر *flag* القطة.

رابعاً – بعد الانتهاء من العملية يجري التقاط عدد القطط ووضعهم في طور البحث أو طور التعقب وفقاً لقيمة *MR*.

خامساً– التحقق من حالة الانتهاء فإذا كانت مرضية ينتهي البرنامج وإلا فالرجوع إلى الخطوة الثانية.

إجمالي عملية الـ *cso* يتم توضيحه في الشكل (5) إذ يبين وصف نواة *CSO* [17]:



الشكل (5). نواة *CSO*

2-2-8 وصف لخوارزمية سرب القطط

خوارزمية سرب القطط تحاكي السلوك الطبيعي للقطط. تكون القطط دائما متيقظة وبطيئة الحركة. وهذا السلوك يمثل طور البحث. فعندما يتم التحسس بوجود فريسة فان القط يطارد بسرعة كبيرة جدا. هذا السلوك أي المطاردة بسرعة عالية يمثل طور التعقب . وقد جرت صياغة هذين الطورين رياضياً لحل مسائل الأمثلية. وقد استخدمت مواقع القطط لتمثل مجموعة الحل. وكل قط له موقع وسرعة لكل بعد وقيمة لياقة. فضلا عن مؤشر يستخدم لتحديد فيما إذا كان القط في طور البحث أم في طور التعقب [20]. وقد اتضح مما سبق فان الـ *CSO* له طوران ثانويان: طور البحث وطور تعقب الأثر. لدمج هذين الطورين في خوارزمية تعرف نسبة امتزاج *MR* التي تحدد اتحاد طور البحث مع طور التعقب. وقد جرى التعليق على تلك القطط التي هي مستيقظة وتقضي غالبية وقتها في الراحة وتراقب البيئة المحيطة بها. فإذا قررت التحرك فان الحركة ستكون بطيئة وحذرة. هذا السلوك يمثل طور البحث [21].

فيما يأتي خطوات خوارزمية سرب القطط:

الخطوة الأولى: تكوين $N=5$ $(P_{min} < P_1,P_2,P_3,P_4$ and $P_5 < P_{max})$ في العملية.

الخطوة الثانية: نشر القطط عشوائياً في أبعاد *M-dimensional* فضاء الحل وإعطاء القيم عشوائياً التي تكون في نطاق الحد الأعلى للسرعة إلى سرع كل قط . ثم التقاط عدد من القطط عشوائياً ووضعهم في طور التعقب وبنسبة $MR=20\%$ والقطط الآخر 80% توضع في طور البحث.

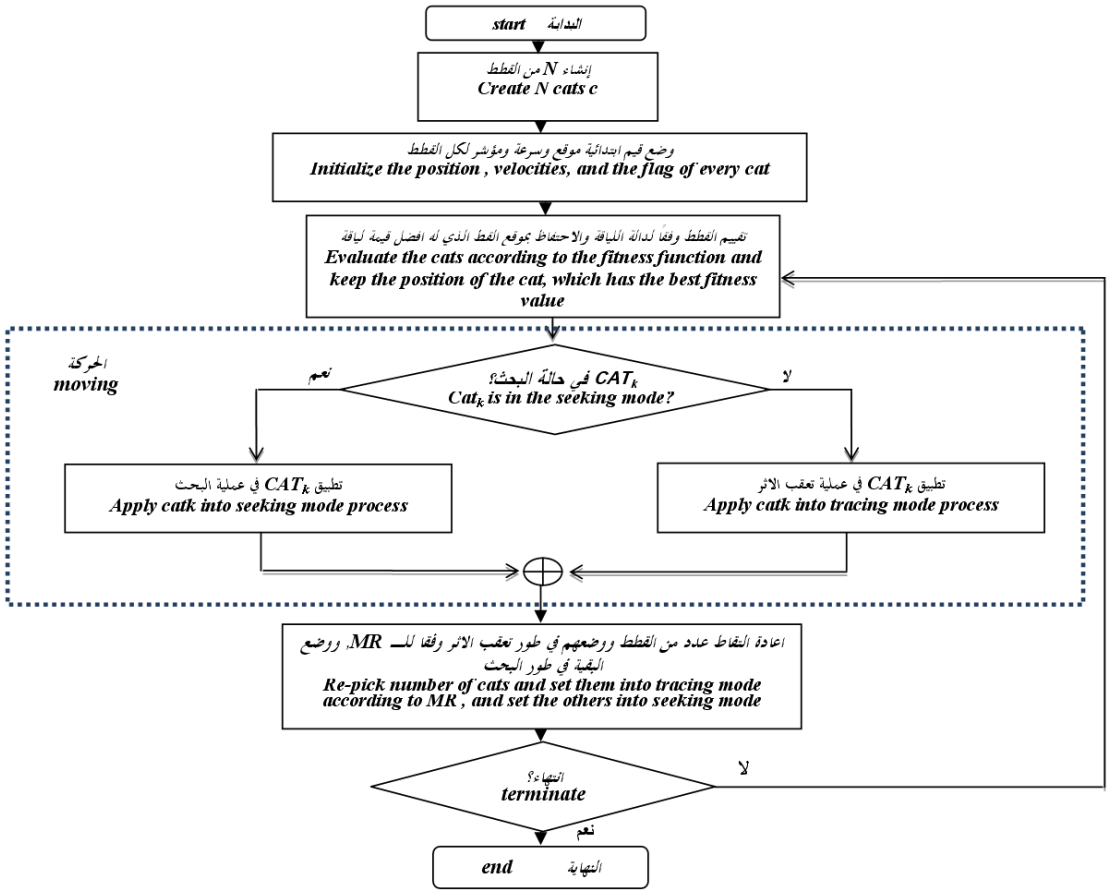
الخطوة الثالثة: حساب قيمة اللياقة لكل قط بتطبيق مواقع القطط في دالة اللياقة التي تمثل معايير الهدف والاحتفاظ بأفضل قط في الذاكرة. هنا يحتاج تذكر موقع أفضل قط P_{best} لأنه يمثل الحل الأفضل حتى الآن.

الخطوة الرابعة: تحريك القطط وفقاً لمؤشراتها، فإذا كان القط CAT_k في طور البحث فسيجعل القط في عملية البحث وإلا يتم تطبيقه في عملية التعقب.

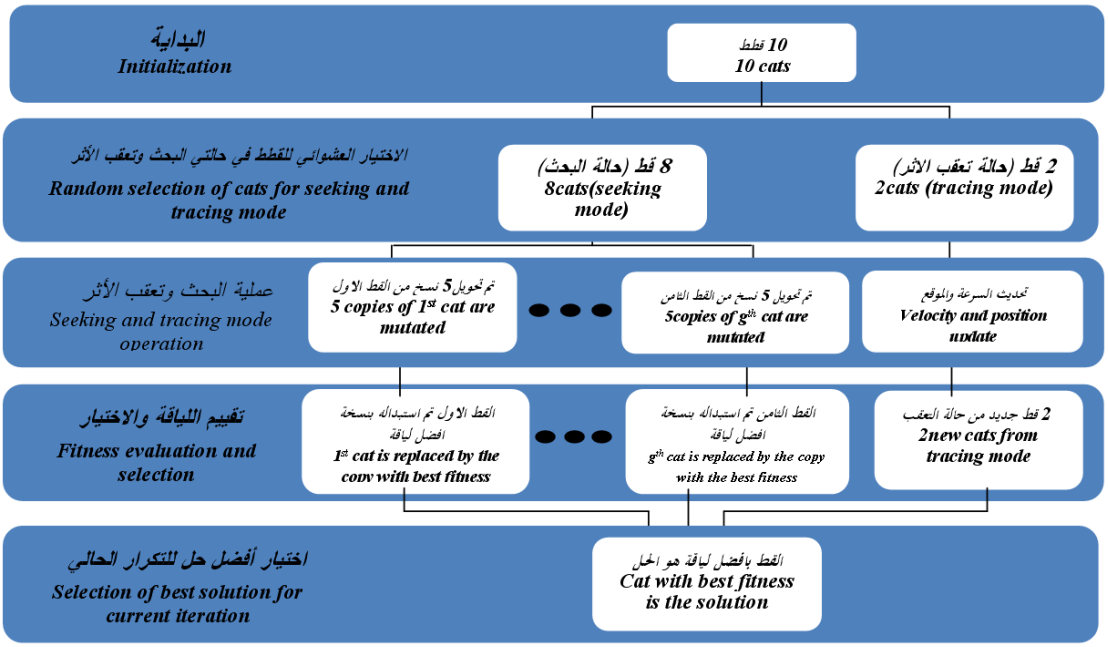
الخطوة الخامسة: يجري التقاط عدد من القطط ووضعهم في طور التعقب وفقاً للـ *MR* ثم وضع بقية القطط في طور البحث.

الخطوة السادسة: التحقق من شروط الانتهاء فإذا كانت مرضية يجري إنهاء البرنامج وإلا فستعاد الخطوات الثالثة إلى الخامسة [21].

والشكل (6) يبين المخطط الانسيابي لخوارزمية سرب القطط *CSO* [21]. والشكل (7) يبين المخطط العام لسلوك سرب القطط بطوريها البحث وتعقب الأثر [22].



الشكل (6). المخطط الانسيابي لخوارزمية الـ *CSO*



الشكل (7). المخطط العام لسلوك خوارزمية سرب القطط

9- آلية استخدام التقنيات الذكائية في توليد حالات الاختبار

إن عملية استخدام تقنيات الذكاء الاصطناعي و توظيفها في مساعدة هندسة البرمجيات يمكن أن يعطي نتائج ذات دقة عالية وكلفة قليلة وسريعة، وقد استخدمت في هذا البحث تقنيات السرب في توليد بيانات الاختبار أوتوماتيكيا التي تحقق تغطية عالية للشفرة وكان العمل في محورين:

1- استخدام خوارزمية سرب الطيور في توليد بيانات الاختبار.

2- استخدام خوارزمية سرب القطط في توليد بيانات الاختبار.

وبداية ينبغي النظر إلى الآليات المشتركة بين هاتين الخوارزميتين (دالة اللياقة) المستخدمة في توليد بيانات الاختبار.

دالة اللياقة: تعد دالة اللياقة من أهم المراحل في خوارزميات السرب لما لها من تأثير عال على سرعة خوارزمية الطيور وخوارزمية القطط ودقتهما، إذ أن دالة اللياقة المقترحة كما في المعادلة الآتية:

$$F(T)=(B(T)+R(t))/2 \quad \ldots(3)$$

الاقتراب من قيمة الحدود: إن فرص الأخطاء تكون غالباً في قيم الحدود لذلك فإن حالات الاختبار القريبة من قيم الحدود يجب أن تعطى أولوية أعلى من تلك الأخرى للاختبار. القيمة القريبة من الحد هي العامل الذي يمثل مدى قرب قيم حالة الاختبار من الحدود. مثلا المسار $\{1,2,4\}$ للبرنامج 1 والذي معادلته المنطقية هي $a > 0 \ \Lambda \ b \leq 0$. وبهذا فإن قيم الحدود لهذا المسار هي $a=0، b=0$. حالات الاختبار ذات قيم a و b القريبة من الصفر تعطى أولوية أكبر من تلك الأبعد عن الصفر. وكما في الاحتمالية فإن هذا العامل يساهم أيضاً في اختيار أي حالة اختبار معينة. وأكثر عامل لأي مجموعة اختبار *Test suit* يرفع احتمالية مجموعة الاختبار ليتم اختيارها للتنفيذ. رياضياً عامل الاقتراب من قيمة الحد B لمجموعة الاختبار T يتم حسابه كالتالي:

$$B(T) = B(t_1) * B(t_2) * \ldots * B(t_n) \quad \ldots(4)$$

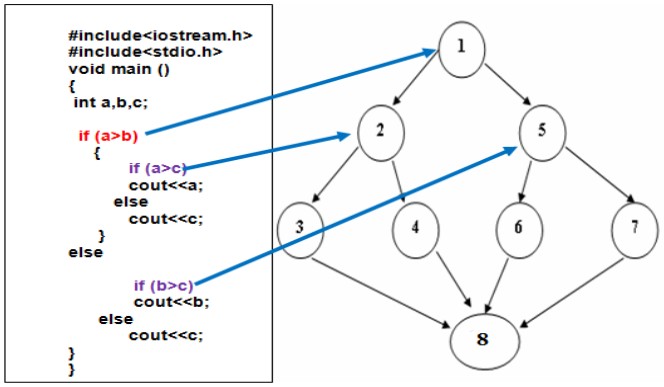
إذ ان $B(t)$ هو عامل الاقتراب من قيمة الحد لحالة الاختبار t [23].

تغطية الفروع

معظم أدوات الاختبار الآلي تستخدم معيار تغطية الفرع لاختيار حالات الاختبار. تغطية الفرع تحسب من نسبة عدد الحافات التي تم تغطيتها من قبل حالات الاختبار مقسوماً على حافات مخطط تدفق السيطرة للبرنامج المراد اختباره، إذ انه كلما زادت هذه النسبة حصل الاقتراب من الوصول إلى حالة الاختبار المثالية التي تغطي جميع المسارات في البرنامج. ويبين مخطط تدفق السيطرة للبرنامج الموضح في الشكل (8) آلية سريان البيانات خلال المسارات. عامل تغطية الفروع لأي مجموعة اختبار T يشار إليه بـ $R(T)$. ولحساب هذا العامل أولا يتم تكوين تدفق السيطرة ومن ثم يطبق على كل حالة اختبار من مجموعة الاختبار وبعدها يحتسب مجموعة الفروع التي جرى تغطيتها في هذه المجموعة. ويتم أخذ الإتحاد*the union* لكل المجاميع المحسوبة من الفروع. وبعدها يحتسب عدد العناصر للمجموعة الناتجة وليكن n ومن ثم تحسب قيمة تغطية الفروع من المعادلة الآتية:

$$R(T)=n/e \quad \ldots(5)$$

إذ أن e تمثل العدد الكلي لحافات مخطط تدفق السيطرة للبرنامج المراد اختباره، وان n تمثل العدد الكلي للحافات التي مرت فيها مجموعة الاختبار على البرنامج[24].



الشكل (8). مخطط تدفق السيطرة للبرنامج

1-9 خوارزمية سرب الطيور في توليد حالات الاختبار

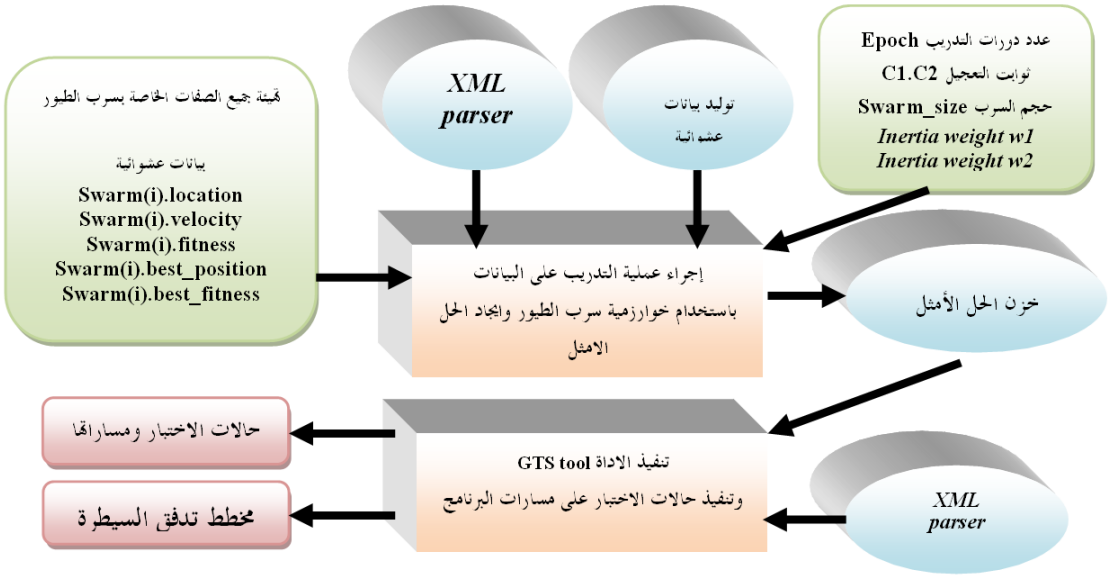
أولا: وحدة التدريب

يجري في هذه الوحدة تدريب الخوارزمية على توليد حالات الاختبار بواسطة تحليل الشفرة البرمجية للبرنامج لإيجاد الحل الأمثل لحالات الاختبار ويكون التدريب بعدد من الدورات.

الإدخـالات: تهيأ في هذه المرحلة جميع الإدخالات الخاصة بالخوارزمية وهي عدد الطيور في السرب *Swarm_size* ، فقد اختير 20 طيراً رقمياً في السرب وعدد دورات التدريب *Epoch* فضلاً عن ثوابت التعجيل c_1 و c_2 والأوزان الخاصة بالمعادلات رقم (1)، (2) فضلا عن إدخال بيانات التدريب الممثلة بمصفوفة أحادية تتكون من 20 طائراً وكل طائر في السرب عبارة عن مصفوفة ثنائية تعبر عن مجموعة الاختبار (التي تضم مجموعة من حالات الاختبار) والشكل (9) يمثل مخطط صندوقي لتنفيذ مرحلة التدريب والاختبار لخوارزمية سرب الطيور، والجدول (2) يبين مدخلات خوارزمية سرب الطيور.

الجدول (2). يبين مدخلات خوارزمية سرب الطيور

Parameter	*Value*
No. Of particles	*20*
Inertia weight w1	*0.9*
Inertia weight w2	*0.6*
Velocity bound	*-32.768 –32.768*
Boundary condition	*1*
Termination	*1000*
c1	*2.1*
c2	*2.1*



الشكل (9). المخطط الصندوقي لتنفيذ مرحلة التدريب والاختبار لخوارزمية سرب الطيور

المعالجة: لكل طير رقمي i معلومات خاصة بهِ في بداية المعالجة تهيأ كل هذه المعلومات ببيانات عشوائية هي:

- موقع الطير في فضاء البحث.
- سرعة الطير الخاصة في فضاء البحث.
- قيمة دالة اللياقة الخاصة بالطير الرقمي $swarm(i).fitness$ وأفضل قيمة لدالة اللياقة حصل عليها هذا الطير $swarm(i).best\_fitness$.
- أفضل موقع حصل عليه الطير في السرب أي أفضل موقع اقترب فيه الطير من الحل الأمثل في فضاء البحث.

يتم إيجاد دالة اللياقة الخاصة بالطير الرقمي التي يجري مقارنتها مع أفضل دالة لياقة حصل عليها الطير $swarm(i).best\_fitness$ في الأجيال السابقة، فإذا كانت القيمة الحالية أفضل فتستبدل القيمة ألسابقة بالقيمة الجديدة، فضلاً عن استبدال قيمة موقع الطير بالموقع الجديد صاحب أفضل دالة لياقة، أما إذا لم تكن دالة اللياقة أفضل فلا تستبدل القيم، بل تبقى القيم القديمة نفسها. وبعد تكرار العمليات السابقة لكل الطيور الرقمية في السرب يجري إيجاد أفضل موقع حصل عليه أفضل طائر بالنسبة للسرب، حيث تتم مقارنة $swarm(i).best\_fitness$ لكل السرب والحصول على الموقع الأفضل والأقرب من الحل الأمثل، بعد ذلك تعدل صفات عناصر السرب

جميعها من ناحية الموقع والسرعة حسب أفضل طير رقمي إذ تُطبَّق المعادلة رقم (1) وتحسب سرعة الطير الرقمي اعتماداً على أفضل طير في السرب و على ثوابت التعجيل مضروبة في عدد عشوائي، إذ يستخدم الإيعاز *Rand* لتوليده مباشرة في المعادلة. والمعادلة رقم (2) يجري فيها تحديث موقع الطير في السرب. تُكرر العمليات السابقة جميعها لحين تحقق شرط التوقف بالوصول للحل الأمثل أو انتهاء العدد المحدد من الدورات *Epoch* ثم يخزن الحل الأمثل في ملف *best_solution.mat*.

المخرجات: أفضل قيم لحالة الاختبار لاستخدامها لاحقاً في عملية اختبار البرنامج ،مسارات البرنامج، رسم مخطط تدفق السيطرة.

ثانيا: وحدة الاختبار

تختبر في هذه الوحدة كفاءة الحل الأمثل الذي استحصل عليه في مرحلة التدريب. الحل الأمثل *gbest.test* الذي يمثل أفضل مجموعة اختبار *Test Suite* التي استحصلت في عملية التدريب والتي تتكون من مصفوفة ثنائية عدد صفوفها يمثل عدد مسارات البرنامج المراد اختباره والتي تسمى حالة الاختبار *Test Case* وعدد أعمدتها يكافئ عدد المتغيرات الموجودة في البرنامج، إذ يستخدم الحل الأمثل (أفضل بيانات اختبار) الذي يمثل أفضل طير في السرب، إذا تمرر على البرنامج للتحقق من مرورها على جميع المسارات ورسم مخطط التدفق الخاص بالبرنامج .

2-9 خوارزمية سرب القطط في توليد حالات الاختبار

أولا: وحدة التدريب

يحدث في هذه الوحدة تدريب الخوارزمية على توليد حالات الاختبار عن طريق تحليل الشفرة البرمجية للبرنامج لإيجاد الحل الأمثل لحالات الاختبار ويكون التدريب بعدد من الدورات.

الإدخالات: يتم في هذه المرحلة تهيئة جميع الإدخالات الخاصة بالخوارزمية وهي عدد القطط في السرب *Swarm_size* فقد اختير 20 قطا رقمياً في السرب وعدد دورات التدريب *Epoch* فضلاً عن ثوابت التعجيل c_1 و c_2 والأوزان الخاصة بالمعادلات رقم (6)، (7) الآتيتين:

$$V_{k.d} = V_{k.d} + rl \,.\, cl\,(X_{best.d} - X_{k.d}) \quad \ldots(6)$$

$$X_{k.d} = X_{k.d} + V_{k.d} \quad \ldots(7)$$

حيث أن: $d = 1, 2, \ldots\ldots, M$

$X_{best,d}$: موقع أفضل قط وأفضل قيمة لياقة.

$X_{k,d}$: الموقع الصحيح للـ cat_k.

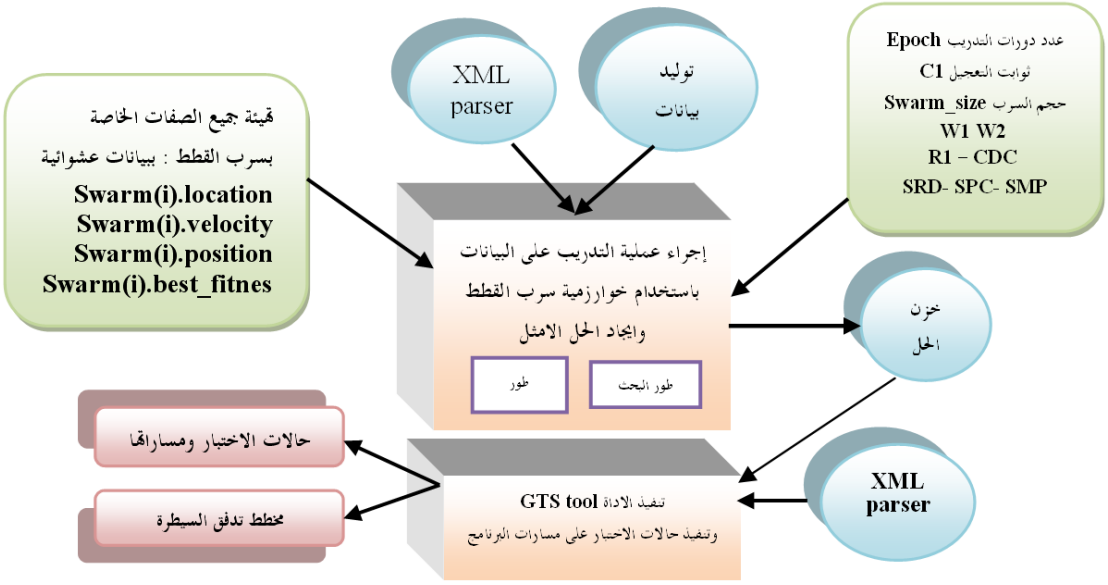
r_1 : متغير عشوائي يعود إلى [1 , 0].

c_1 : ثابت، الذي يُحدد بـ 2 في التجارب.

فضلا عن إدخال بيانات التدريب الممثلة بمصفوفة أحادية تتكون من 20 قطا وكل قط في السرب يمثل مصفوفة ثنائية تمثل بدورها مجموعة الاختبار(التي تضم مجموعة من حالات الاختبار). والشكل (10) يمثل المخطط الصندوقي لتنفيذ مرحلة التدريب والاختبار لخوارزمية سرب القطط.

الجدول (3). يبين مدخلات خوارزمية سرب القطط

Parameter	*Value or range*
No. Of CAT	*20*
SMP	*5*
SPC	*T – F*
SRD	*[0-1]*
CDC	*1*
MR	*RANDOM*
C1	*2.1*
R1	*[0,1]*
W1	*0.6*
W2	*0.9*



الشكل (10). المخطط الصندوقي لتنفيذ مرحلة التدريب والاختبار لخوارزمية سرب القطط

المعالجة: لكل قط رقمي i معلومات خاصة بهِ في بداية المعالجة وتهيأ هذه المعلومات كلها في بيانات عشوائية هي:

- موقع القط في فضاء البحث.
- سرعة القط الخاصة في فضاء البحث.
- إذ تقسم هذه البيانات في كل دورة على طورين هما :

أولاً: طور البحث

ينشأ لكل قط في هذا الطور مجموعة من النسخ وعددها خمسة، وهناك متغير يسمى *SRD* اعتمادا على هذا المتغير يحدد فيما إذا كان القط الحالي هو ضمن النسخ الجديدة أم لا، ويتم حساب دالة اللياقة لكل نسخة من القط، بعدها تحتسب احتمالية بقاء كل نسخة من الجيل حسب المعادلة (8) فالنسخة ذات الاحتمالية الأعلى تمتلك فرصة اكبر للبقاء والانتقال إلى الجيل القادم، وتجرى هذه العملية على جميع القطط الموجودة في هذا الطور، يلي ذلك تحديد أفضل عنصر إذ يتم إيجاد أفضل موقع حصل عليه أفضل قط نسبة للسرب، حيث تتم مقارنة $swarm(i).best\_fitness$ لكل السرب والحصول على الموقع الأفضل والأقرب من الحل الأمثل.

ثانياً: طور التتبع

$$p_i = \frac{|FSl - FSb|}{FSmax - FSmin}, , , , , , , , \text{where } 0 < i < j \quad \ldots(8)$$

If a minimization fitness function is used,

$$FS_b = FS_{min}$$

Else

$$FSb = FSmax$$

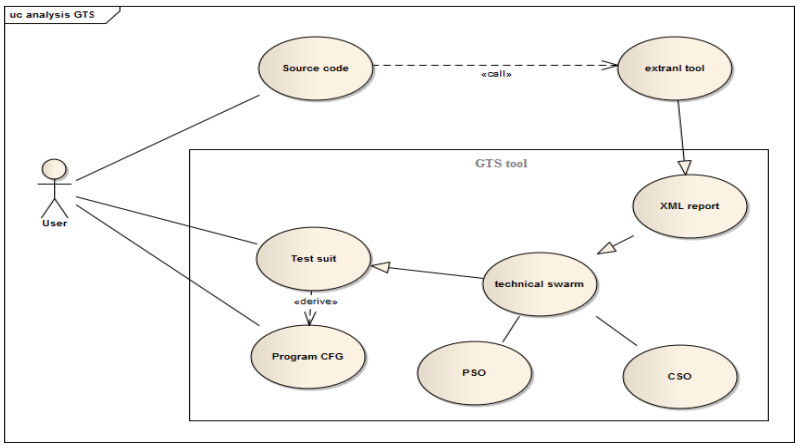
يؤخذ في هذا الطور كل قط من القطط الخاصة بهذه المرحلة على حدة ويحدث سرعته في مجال البحث وفق المعادلة (6) اعتماداً على أفضل قط في السرب و ثوابت التعجيل مضروبة في عدد عشوائي، ويستخدم الإيعاز *Rand* لتوليده مباشرة في المعادلة. واعتماداً على هذه القيمة يحدث الموقع الحالي لهذا القط اعتماداً على المعادلة (7). تُكرر العمليات السابقة لحين تحقق شرط التوقف بالوصول للحل الأمثل أو انتهاء العدد المحدد من الدورات *Epoch* ثم يخزن الحل الأمثل في ملف *best_solution.mat*.

المخرجات: أفضل قيم لحالة الاختبار لاستخدامها لاحقاً في عملية اختبار البرنامج ،مسارات البرنامج، رسم مخطط تدفق السيطرة.

ثانيا: وحدة الاختبار: يحدث في هذه الوحدة اختبار كفاءة الحل الأمثل الذي جرى الحصول عليه في مرحلة التدريب. الحل الأمثل gbest.test الذي يمثل أفضل مجموعة اختبار *Test Suite* التي استحصلت عليها في عملية التدريب والتي تتكون من مصفوفة ثنائية عدد صفوفها يمثل عدد مسارات البرنامج المراد اختباره والتي تسمى حالة الاختبار *Test Case* وعدد أعمدتها يكافئ عدد متغيرات الموجودة في البرنامج، اذ يستخدم الحل الأمثل (أفضل بيانات اختبار) الذي يمثل أفضل طير في السرب، وتمرر على البرنامج للتحقق من مرورها على جميع المسارات ورسم مخطط التدفق الخاص بالبرنامج .

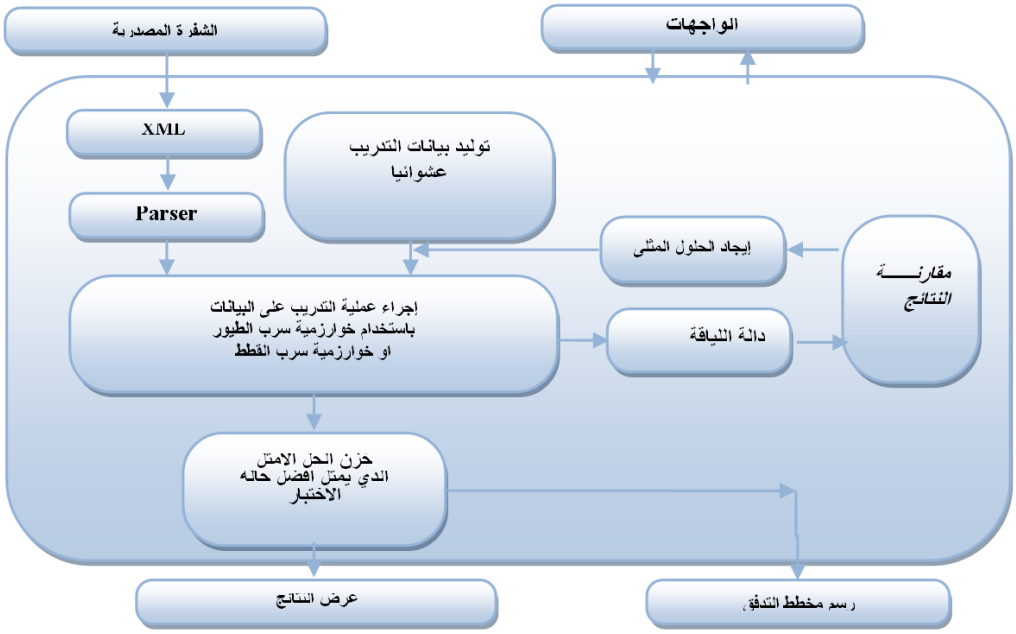
10- تحليل الأداة المقترحة

قبل البدء بعملية التحليل التفصيلية للأداة المقترحة *GTS tool* يجب البدء بوصفها بشكل عام وبطريقة تصف وجهة نظر المستخدم النهائي للأداة التي تعبر عن الكيفية التي ينبغي أن يكون فيها شكل النظام النهائي، من ناحية المستخدم. يمثل الشكل (11) المتطلبات الوظيفية للأداة من ناحية المستخدم النهائي، إذ ان المستخدم يتعامل مع الشفرة البرمجية المصدرية للبرنامج المراد اختباره وبيانات حالات الاختبار المولدة باستخدام تقنيات السرب التي تستخدم لفحص الشفرة البرمجية المصدرية، وأيضاً توفر الأداة مخطط رسومي لتدفق السيطرة لبيانات البرنامج *CFG*، وتعتمد عملية توليد حالات الاختبار على تقنيات ذكائية وهي تقنية سرب الطيور، وتقنية سرب القطط.



الشكل (11). تحليل الأداة من وجهة نظر المستخدم النهائي

إن أنموذج الأداة يستخلص المسارات للبرنامج المراد اختباره بلغه *C++* من تقرير بهيئة *xml* وهو إخراج البرنامج التجاري *(Code to XML)*. حيث يجري الحصول على بيانات الاختبار من تحليل المسارات التي جرى الحصول عليها من تقرير *xml*. وبشكل عام فان المعمارية العامة للأداة المقترحة كما موضحة في الشكل (12) الذي يبين الأجزاء الرئيسة المكونة للأداة المقترحة.



الشكل (12). معمارية الأداة المقترحة

11- النتائج

تتضمن هذه الفقرة كيفية التمثيل العملي *Practical Implementation* للأداة المقترحة بوصفها تنفيذاً للبرامج والنتائج التي جرى الحصول عليها فضلاً عن مناقشة هذه النتائج. وتعد هذه الفقرة استكمالاً لمرحلة بناء الأداة المقترحة، وذلك لغرض إتمام مرحلة بناء الأداة. حيث تعرض نتائج اختبار الأداة المقترحة *GTS*، وذلك من خلال الخطوات آلاتية:

1. اختيار بيانات الاختبار المناسبة لمدخلات الأداة التي تتكون من ملف بصيغة *XML* يحوي على تحليل الشفرة البرمجية للبرنامج المراد اختباره وهو إخراج الأداة الـ*Code To XML*.

2. عرض نتائج توليد حالات الاختبار وتحليل الشفرة البرمجية من خلال توضيح المسارات ورسم مخطط تدفق السيطرة للبرنامج المراد اختباره وذلك باستخدام إحدى التقنيات الذكائية الآتية:

- توليد حالات الاختبار باستخدام خوارزمية سرب الطيور.
- توليد حالات الاختبار باستخدام خوارزمية سرب القطط.

جرى اختبار الأداة على أكثر من برنامج مكتوباً بلغة $C++$، ولعرض النتائج فقد جرى اختيار الشفرة البرمجية المصدرية للبرنامج الأول المتضمن حالة *IF* وكما موضح بالشكل (13). إذ حولت الشفرة البرمجية في الشكل السابق إلى صيغة الـ *XML* الموضحة في الشكل (14) وتعد هذه الصيغة إدخال للأداة المقترحة.

```
#include<iostream.h>
#include<stdio.h>
void main ( int a , int b,int c)
{
        if (a>b)
        {
                if (a>c)
                        cout<<"a";
                else
                        cout<<"c";
        }
        else
        {
                if(b>c)
                        cout<<"b";
                else
                        cout<<"c";
        }
}
```

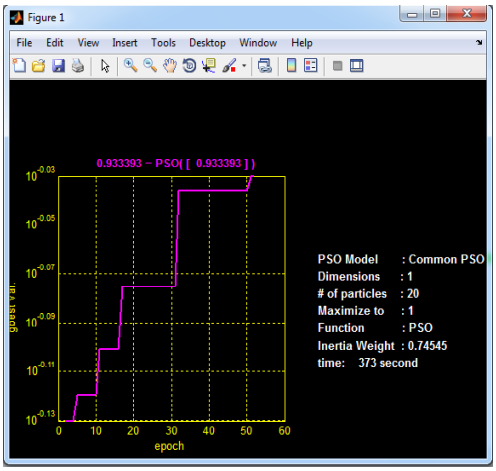
الشكل (13). الشفرة البرمجية المصدرية للبرنامج الأول

```
<?xml version="1.0" encoding="GB2312"?>
- <Function EndPos="211" BegPos="75" Text="void main">
  - <IF EndPos="208" BegPos="79" Text="(a>b)">
    - <YES EndPos="144" BegPos="90" Text="Yes">
      - <IF EndPos="140" BegPos="95" Text="(a>c)">
        - <YES EndPos="117" BegPos="108" Text="Yes">
            <Code EndPos="117" BegPos="108" Text="cout<<"a";"/>
          </YES>
        - <NO EndPos="140" BegPos="131" Text="No">
            <Code EndPos="140" BegPos="131" Text="cout<<"c";"/>
          </NO>
        </IF>
      </YES>
    - <NO EndPos="208" BegPos="155" Text="No">
      - <IF EndPos="204" BegPos="160" Text="(b>c)">
        - <YES EndPos="181" BegPos="172" Text="Yes">
            <Code EndPos="181" BegPos="172" Text="cout<<"b";"/>
          </YES>
        - <NO EndPos="204" BegPos="195" Text="No">
            <Code EndPos="204" BegPos="195" Text="cout<<"c";"/>
          </NO>
        </IF>
      </NO>
    </IF>
  </Function>
```

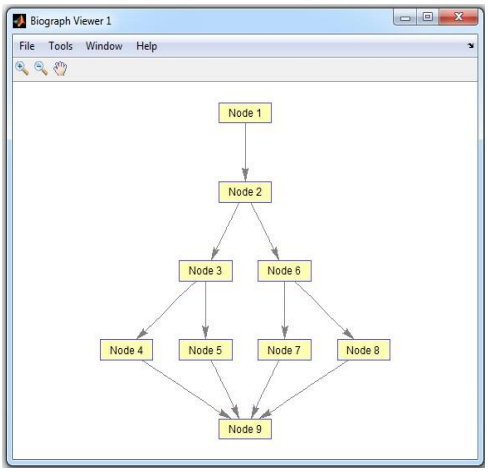
الشكل (14). البرنامج الأول بصيغة الـ *XML*

- **نتائج تنفيذ الأداة المقترحة باستخدام خوارزمية سرب الطيور PSO للبرنامج الأول**

بعد أن يُحدد البرنامج المراد توليد حالات الاختبار المثالية الخاصة به، وتُحدد الخوارزمية المستخدمة لتوليد حالات الاختبار، فان على المستخدم تحديد معلمات الخوارزمية المناسبة لعرض التحكم في أدائها وهي عدد عناصر السرب وعدد التكرارات. وإن الأداة تدعم قيم افتراضية، في حالة عدم إدخال المستخدم لأية قيم جديدة. وقد جرى توليد حالات الاختبار للبرنامج في الشكل (13) وكانت قيم عدد عناصر السرب *20* وعدد التكرارات *1000*، والشكل (15) يوضح واجهة منحني وصول خوارزمية *PSO* إلى الهدف.



الشكل (15). منحني وصول خوارزمية POS إلى الهدف للبرنامج الأول



الشكل (16). مخطط التحكم في التدفق للبرنامج الأول

وكانت نتائج التنفيذ كما يأتي: الوقت الذي استغرقته الأداة في الوصول إلى حالات الاختبار هو 373 ثانية، عدد التكرارات 52، نسبة الدقة 0.93، مخطط تدفق السيطرة للبرنامج كما موضح في الشكل (16)، حالات الاختبار المثالية مع مسارات البرنامج الموضحة في الجدول (4).

الجدول (4). حالات الاختبار المثالية والمسارات الخاصة بها للبرنامج الأول باستخدام خوارزمية سرب الطيور

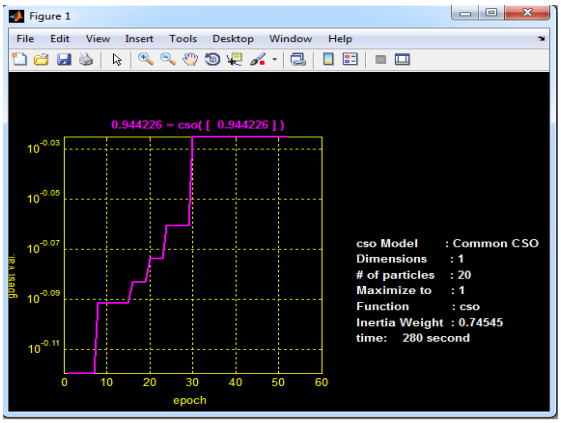
الطريق المتبع لحالات الاختبار المولدة	حالات الاختبار المولدة
{1-2-3-4-9}	*{-15677 , -16231 , -16073}*
{1-2-6-7-9}	*{10523 , 11854 , 11034 }*
{1-2-6-8-9}	*{ 32767 , 32767 , 32767}*
{1-2-3-5-9}	*{ 32758 , 31692 , 32767 }*

- **نتائج تنفيذ الأداة المقترحة باستخدام خوارزمية سرب القطط *CSO* للبرنامج الأول**

بعد تحديد الخوارزمية لتوليد حالات الاختبار المستخدمة على أنها خوارزمية سرب القطط، فإن على المستخدم تحديد معلمات الخوارزمية المناسبة وهي كما يأتي: عدد عناصر السرب، عدد التكرارات، نسبة عدد العناصر لحالة التعقب، نسبة عدد العناصر لحالة البحث. والأداة أيضاً تدعم قيم افتراضية لهذه الخوارزمية، في حالة عدم إدخال المستخدم لأي قيم جديدة. وقد جرى توليد حالات اختبار للبرنامج في الشكل (13) وكانت قيم عدد عناصر السرب 20 وعدد التكرارات 1000، ونسبة عدد عناصر البحث 60% ونسبة عدد عناصر التعقب 40% والشكل(17) يوضح منحني وصول خوارزمية *CSO* إلى الهدف.

الجدول (5). حالات الاختبار المثالية والمسارات الخاصة بها للبرنامج الأول باستخدام خوارزمية سرب القطط

الطريق المتبع للحالات الاختبار المولدة	حالات الاختبار المولدة
{1-2-6-8-9}	*{-8 , 43 , 64}*
{1-2-3-4-9}	*{108 , -56 , -12 }*
{1-2-3-5-9}	*{ 74 , -17 , 106 }*
{1-2-6-7-9}	*{ -139 , 142 , 75 }*

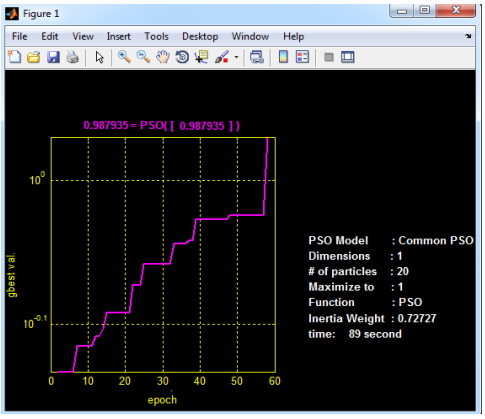


الشكل (17). منحني وصول خوارزمية *COS* إلى الهدف للبرنامج الأول

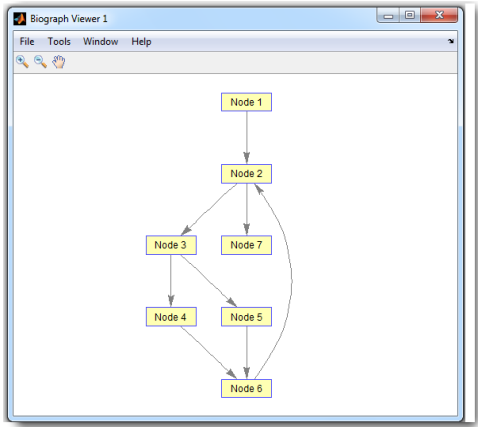
وكانت نتائج التنفيذ كما يأتي: الوقت الذي استغرقته الأداة في الوصل إلى حالات الاختبار هو 280 ثانية، عدد التكرارات 52، نسبة الدقة 0.94، مخطط تدفق السيطرة للبرنامج كما موضح في الشكل (16). وحالات الاختبار المثالية مع مسارات البرنامج موضحة في الجدول (5).

وكذلك جرى اختبار الأداة على برنامج آخر مكتوباً بلغة *C++* أيضاً المتضمن حالة *FOR* إذ حولت الشفرة البرمجية إلى صيغة الـ *XML* وتعد هذه الصيغة إدخال للأداة المقترحة.

- **نتائج تنفيذ الأداة المقترحة باستخدام خوارزمية سرب الطيور PSO للبرنامج الثاني**



الشكل (18). منحني وصول خوارزمية *POS* إلى الهدف للبرنامج الثاني



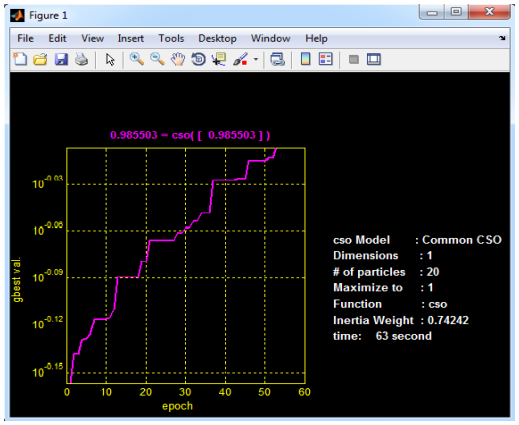
الشكل (19). مخطط التحكم في التدفق للبرنامج الثاني

وكانت نتائج التنفيذ كما يأتي: الوقت الذي استغرقته الأداة في الوصول إلى حالات الاختبار هو 89 ثانية، عدد تكرارات 58، نسبة الدقة 0.98، مخطط تدفق السيطرة للبرنامج كما موضح في الشكل (19). حالات الاختبار المثالية مع المسارات البرنامج الموضحة في الجدول (7).

الجدول (7). حالات الاختبار المثالية والمسارات الخاصة بها للبرنامج الثاني باستخدام خوارزمية سرب الطيور

الطريق المتبع للحالات الاختبار المولدة	حالات الاختبار المولدة
{1-2-3-4-6-2-7}	{ 2 , 3 }
{1-2-3-5-6-2-7}	{ 5 , 1 }
{1-2-7}	{ 6 , 11 }

- **نتائج تنفيذ الأداة المقترحة باستخدام خوارزمية سرب القطط CSO للبرنامج الثاني**



الشكل (20). منحني وصول خوارزمية *COS* إلى الهدف للبرنامج الثاني

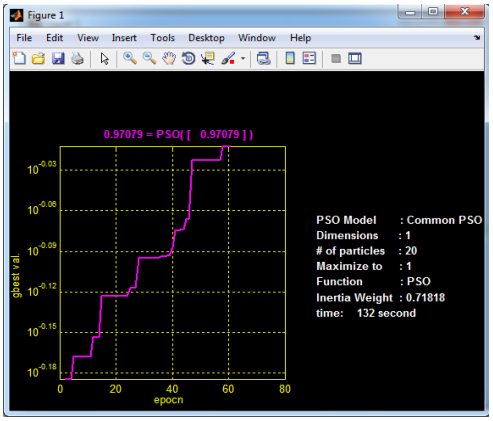
الجدول (8). حالات الاختبار المثالية والمسارات الخاصة بها للبرنامج الثاني باستخدام خوارزمية سرب القطط

الطريق المتبع لحالات الاختبار المولدة	حالات الاختبار المولدة
{1-2-7}	{ 3 , 5 }
{1-2-3-4-6-2-7}	{ 2 , 3 }
{1-2-3-5-6-2-7}	{ 4 , 0 }

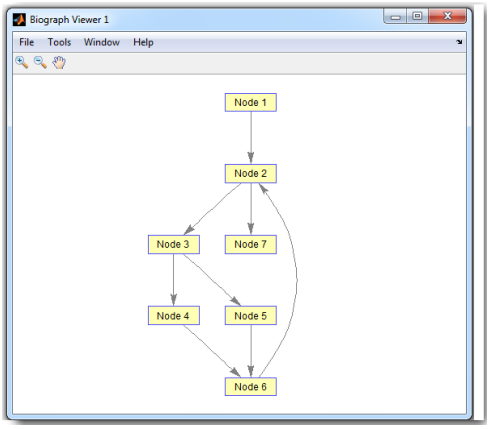
وكانت نتائج التنفيذ كما يأتي: الوقت الذي استغرقته الأداة في الوصول إلى حالات الاختبار هو 63 ثانية، عدد التكرارات 53، نسبة الدقة 0.98، مخطط تدفق السيطرة للبرنامج كما موضح في الشكل (19)، حالات الاختبار المثالية مع مسارات البرنامج الموضحة في الجدول (8).

وقد تم تنفيذ اختبار الأداة على برنامجاً آخر يتضمن حالة *WHILE* إذ حولت الشفرة البرمجية له إلى صيغة الـ *XML* وهذه الصيغة تعد إدخال للأداة المقترحة.

- **نتائج تنفيذ الأداة المقترحة باستخدام خوارزمية سرب الطيور PSO للبرنامج الثالث**



الشكل (21). منحني وصل خوارزمية POS إلى الهدف للبرنامج الثالث



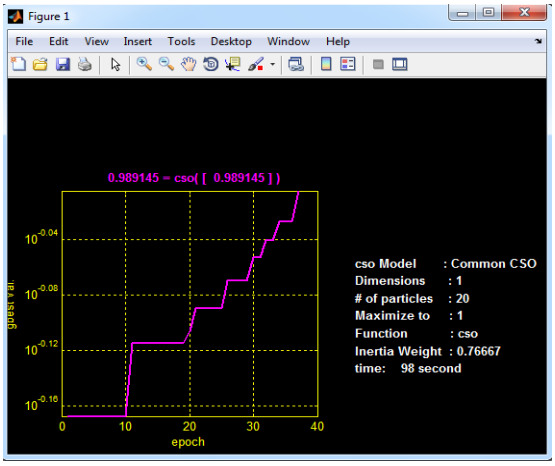
الشكل (22). مخطط التحكم في التدفق للبرنامج الثالث

وكانت نتائج التنفيذ كما يأتي: الوقت الذي استغرقته الأداة في الوصول إلى حالات الاختبار هو 132 ثانية، عدد التكرارات 61، نسبة الدقة 0.97، مخطط تدفق السيطرة للبرنامج كما موضح في الشكل (22)، حالات الاختبار المثالية مع مسارات البرنامج الموضحة في الجدول (9).

الجدول (9). حالات الاختبار المثالية والمسارات الخاصة بها للبرنامج الثالث باستخدام خوارزمية سرب الطيور

الطريق المتبع للحالات الاختبار المولدة	حالات الاختبار المولدة
{1-2-3-4-6-2-7}	{ 8 , 59 }
{1-2-3-5-6-2-7}	{13 , 30 }
{1-2-7}	{ 88 , 9 }

- **نتائج تنفيذ الأداة المقترحة باستخدام خوارزمية سرب القطط CSO للبرنامج الثالث**



الشكل (23). منحني وصل خوارزمية COS إلى الهدف للبرنامج الثالث

الجدول (10). حالات الاختبار المثالية والمسارات الخاصة بها للبرنامج الثالث باستخدام خوارزمية سرب القطط

الطريق المتبع للحالات الاختبار المولدة	حالات الاختبار المولدة
{1-2-3-4-6-2-7}	{ 22 , 71 }
{1-2-3-5-6-2-7}	{19 , 40 }
{1-2-7}	{ 31 , 8 }

وكانت نتائج التنفيذ كما يأتي: الوقت الذي استغرقته الأداة في الوصول إلى حالات الاختبار هو 98 ثانية. عدد التكرارات 45 ونسبة الدقة 0.99. مخطط تدفق السيطرة للبرنامج موضح في الشكل (22). حالات الاختبار المثالية مع المسارات البرنامج موضحة في الجدول (10).

12- مقارنة نتائج الخوارزميتان في توليد حالات الاختبار

لغرض بناء مقارنة دقيقة بين الخوارزميتان، فقد جرى فحص أداء الأداة المقترحة على ثلاثة برامج مختلفة، البرنامج الأول المتضمن حالة if والثاني المتضمن حالة for والثالث حالة while. الجدول رقم (11) يبين نتائج توليد حالات الاختبار للبرامج الثلاثة من ناحية السرعة والدقة والأداء.

الجدول (11). نتائج توليد حالات الاختبار للبرامج الثلاثة من ناحية السرعة والدقة والأداء باستخدام خوارزمية سرب الطيور وخوارزمية سرب القطط

الخاصية	خوارزمية سرب الطيور			خوارزمية سرب القطط		
اسم البرنامج	P1	P2	P3	P1	P2	P3
الوقت المستغرق بالثواني	373	89	132	280	63	98
التكرارات	52	58	61	52	53	45
الدقة	0.93	0.98	0.97	0.94	0.98	0.99

كما يتضح من الجدول أعلاه إن توليد حالات الاختبار باستخدام خوارزمية سرب القطط كانت الأفضل لجميع البرامج المختارة في البحث اعتماداً على المقاييس المحسوبة.

13- الاستنتاجات

من خلال بناء أداة لتوليد حالات الاختبار *GTS tool* التي صممت في هذا البحث والتي تتضمن توليد حالات اختبار مثالية باستخدام التقنيات الذكائية، وتنفيذ خوارزميات الأداة المقترحة في البحث والنتائج التي جرى الحصول عليها من التنفيذ العملي للأداة المقترحة، استحصلت استنتاجات عديدة تتلخص كما يأتي:

- ❖ استنتج أن توليد حالات الاختبار بشكل تلقائي يوفر لفريق تطوير البرمجيات مزايا عديدة:
  - يقلل من الوقت المستغرق لتوليد حالات اختبار مثالية لفحص البرمجيات.
  - يقلل من الجهد والأعباء على الفريق البرمجي أثناء فحص البرمجيات.
  - يقلل الكلفة على المؤسسة المطورة للبرمجيات، إذ أن عملية الاختبار تكلف أكثر من 50% من الكلفة الكلية لتطوير البرمجيات.
  - استخدام الأداة المقترحة *GTS* في توليد حالات الاختبار بشكل تلقائي ساعد في تقليل عدد من مشكلات توليد حالات الاختبار بشكل يدوي.
- ❖ ان استخدام التقنيات الذكائية لمساعدة هندسة البرمجيات يعطي دفعة قوية وسرعة ودقة متناهيتين في دعم مراحل هندسة البرمجيات ولاسيما في مرحلة فحص البرمجيات.
- ❖ استنتج أن تطبيق خوارزمية سرب الطيور وخوارزمية سرب القطط في توليد حالات اختبار مثالية تشمل جميع الاحتمالات بأقل عدد من حالات الاختبار وبدقة عالية وسرعة عالية جدا تقدر بالدقائق، يمكن ان يكون ذو فائدة كبير في هندسة البرمجيات.
- ❖ إن استخدام خوارزمية سرب الطيور في توليد حالات الاختبار ومن خلال التطبيق العملي على عدة برامج أعطت أفضل النتائج وبوقت قصير بمعدل 4 دقائق وبدقة عالية تصل إلى 97%.

- يتبين عند استخدام خوارزمية سرب القطط في توليد حالات الاختبار ومن خلال التطبيق العملي على عدة برامج أعطى نتائج أفضل من طريقة سرب الطيور وبوقت قصير بمعدل 1.15 دقائق وبدقة عالية تصل إلى 99%.
- وأخيرا لوحظ من خلال المقارنة التي أجريت بين خوارزمية سرب الطيور وخوارزمية سرب القطط في الجدول (11)، أن خوارزمية سرب القطط كانت أفضل من ناحية السرعة والدقة في الوصول إلى حالات الاختبار المثالية مقارنة مع خوارزمية سرب الطيور.

المصادر

[1] زاهر الحاج حسين، 2006، "هندسة البرمجيات ثنائية الهندسة والإدارة"، شعاع للنشر والعلوم.

[2] Sangeeta Sabharwal, Ritu Sibal, Chayanika Sharma, 2011, "A Genetic Algorithm based approach for prioritization of test case scenarios in static testing", IEEE, International Conference on Computer & Communication Technology (ICCCT).

[3] Ioan Mihnea, Radu, 2008, "Testing: First Step Towards Software Quality", journal of applied Quantitative Methods.

[4] Mohd. Ehmer Khan, 2010, "Different Forms of Software Testing Techniques for Finding Errors", IJCSI International Journal of Computer Science Issues.

[5] Mohd. Ehmer Khan. 2011, "Different Approaches to White Box Testing Technique for Finding Errors", International Journal of Software Engineering and Its Applications.

[6] Sowmya Padmanabhan, 2004, "Domain Testing: Divide and Conquer", Master Thesis, in Computer Sciences, Florida Institute of Technology, Melbourne, Florida

[7] Gustaf Brannstrom, 2012, "Automated software testing for cross-platform systems", Master's Thesis in Computing Science, Umea University Department of Computing Science, SWEDEN.

[8] Naveen Jayachandran, 2005,"Understanding ROI Metrics for Software Test Automation", Master thesis, in Computer Science Department of Computer Science and Engineering College of Engineering, University of South Florida

[9] Harmen - Hinrich Sthamer, 1995, "The Automatic Generation of Software Test Data Using Genetic Algorithms", Doctor of Philosophy thesis, University of Glamorgan / Prifvsgol Morgannwg.

[10] Pankaj Jalote, 2008, "A Concise Introduction to Software Engineering", Springer, Science+Business Media.

[11] Ranjita Swain, Vikas Panthi, Prafulla Kumar Behera, Durga Prasad Mohapatra, 2012, "Automatic Test case Generation From UML State Chart Diagram", International Journal of Computer Applications.

[12] Chen Mingsong, Qiu Xiaokang, Li Xuandong, 2006, "Automatic Test Case Generation for UML Activity Diagrams", ACM, Shanghai, China.

[13] Xiaohui Cui, Thomas E. Potok, 2006, "Swarm Intelligence in Text Document Clustering", Computational Sciences and Engineering Division Oak Ridge National Laboratory.

[14] Dr.R.Umarani, V.Selvi, 2010, "Particle Swarm Optimization-Evolution, Overview and Applications", International Journal of Engineering Science and Technology.

[15] Sanjay Singla, Dharminder Kumar, H M Rai, Priti Singla, 2011,"A Hybrid PSO Approach to Automate Test Data Generation for Data Flow Coverage with

Dominance Concepts", International Journal of Advanced Science and Technology.

[16] Ajith Abraham, Crina Grosan, Vitorino Ramos, 2006, "Swarm Intelligence in Data Mining", Springer-Verlag Berlin Heidelberg.

[17] Noel D. Bacarisas, John Paul T. Yusiong, 2011, The Effect of Varying the Fitness Function on the Efficiency of the Cat Swarm Optimization algorithm in Solving the Graph Coloring Problem", Annals, Computer Science Series.

[18] Budi Santosa, Mirsa Kencana Ningrum, 2009, "Cat Swarm Optimization for Clustering", IEEE, International Conference of Soft Computing and Pattern Recognition.

[19] Maysam Orouskhani, Mohammad Mansouri, Mohammad Teshnehlab, 2011, "Average-Inertia Weighted Cat Swarm Optimization", Springer-Verlag Berlin Heidelberg.

[20] Pyari Mohan Pradhan, Ganapati Panda, 2012, " Solving multiobjective problems using cat swarm optimization", Elsevier Ltd. Expert Systems with Applications 2956–2964.

[21] Jong-Ching Hwang, Jung-Chin Chen, J.S. Pan, Yi-Chao Huang, 2009, "CSO and PSO to Solve Optimal Contract Capacity for High Tension Customers", Electrical Engineering Department, National Kaohsiung University of Applied Sciences, Kaohsiung, Taiwan.

[22] Ganapati Panda, "Cat Swarm Optimization: Theory and Application to Direct and Inverse Modeling", School of Electrical Sciences, Indian Institute of Technology Bhubaneswar.

[23] Harsimran Singh, 2004, "Automatic Generation of Software Test Cases using Genetic Algorithms", Master thesis In Software Engineering, Computer Science & Engineering Department Thapar Institute of Engineering & Technology, Patiala.

[24] Bogdan Korel, 1990, "Software Test Data Generation", IEEE, Computer Sociaty and Association for Computing Machinery.