الجمهورية العربية السورية

جـــــامعة دمــشـــق

كلية الهندسة الميكانيكية و الكهربائية

قسم هندسة الحواسيب والأتمتة

تطوير خوارزمية جدولة مهام في الحوسبة السحابية بالإعتماد على الخوارزميتين SOS-ICA

رسالة أُعدت لنيل درجة الماجستير في هندسة الحواسيب وشبكاتها

إعداد

المهندس بشار محسن إبراهيم

بإشراف

الدكتورة رأفة خازم

أستاذ في قسم هندسة الحواسيب والأتمتة

مشرف مشارك

الدكتورة رؤى ونوس

باحث في المعهد العالي للعلوم التطبيقية والتكنولوجيا

لجنة الحكم

قرار مجلس البحث العلمي والدراسات العليا رقم 3144 المتخذ بالجلسة رقم 19 تاريخ 4/7/2022.

لجنة الحكم تتألف من السادة:

د. رأفة حازم	مدرس في قسم هندسة الحواسيب والأتمتة	عضواً مشرفاً
د. مفيد حداد	مدرس في قسم هندسة الحواسيب والأتمتة	عضواً
د. ياسر ذيبان	مدرس في قسم هندسة الحواسيب والأتمتة	عضواً

إهداء

إلى أمي وأبي ...

كلمة شكر

أتوجه بالشكر إلى الكادر التدريسي في قسم هندسة الحواسيب والأتمتة في كلية الهندسة الميكانيكية والكهربائية في جامعة دمشق لجهودهم وتوجيهاتهم في إغناء هذا البحث، وأخص بالذكر الدكتورة رأفة خازم لجهودها ومتابعتها لإخراج البحث بالشكل الأفضل.

أتوجه بالشكر إلى الدكتورة رؤى ونوس في المعهد العالي للعلوم التطبيقية والتكنولوجيا لمتابعتها وتوجيهاتها الغنية.

الفهرس

قائمة الأشكال

قائمة الجداول

جدول الاختصارات

الاختصار	معنى الاختصار	شرح
ACO	Ant Colony Algorithm	خوارزمية مستعمرة النمل
Cloudlet	Task	المهمة المراد تنفيذها
DSOS	Discrete Symbiotic Organisms Search	البحث المتقطع في الكائنات المتكافلة
ETC	Expected Time to Compute	الزمن المتوقع للتنفيذ
FCFS	First Come First Served	من يصل أولاً يُخدَّم أولاً
ICA	Imperialist Competitive Algorithm	خوارزمية تنافس المستعمِرين
Metaheuristic	Meta-heuristic	فائقة الاستدلال
MIPS	million instructions per second	مليون تعليمة في الثانية
SOS	Symbiotic Organisms Search	خوارزمية البحث في الكائنات المتكافلة
VM	Virtual Machine	آلة افتراضية ضمن النظام السحابي

ملخص

إنَّ الهدف من الجدولة في الحوسبة السحابية هو تحصيص الموارد المتوفرة ليتم تنفيذ المهام المطلوبة خلال أقل زمن تنفيذ واستخدام فعال للموارد، ويكون دور نظام الجدولة هو اتخاذ القرار المناسب الذي يضمن تنفيذ جميع المهام بحيث يكون مدى الاستفادة من الموارد عالياً ويكون زمن التنفيذ أقل مايمكن، ولتحقيق ذلك كان التوجه لاستخدام خوارزميات الأمثلة في هذا البحث.

يقوم هذا البحث بتطبيق خوارزميتَين من خوارزميات الأمثَلة في جدولة مجموعة من المهام على آلات افتراضية متوفرة في نظام سحابي، هاتان الخوارزميتان هما خوارزمية البحث في الكائنات المتكافلة SOS (Symbiotic Organisms Search) وقد يُطلق عليها اسم خوارزمية التكافل البيئي، وخوارزمية التنافس بين المستعمرين ICA (Imperialist Competitive Algorithm) وقد يُطلَق عليها اسم خوارزمية التنافس الإمبريالي، وهما تهدفان للبحث عن حل أمثلي يتمثل بجدولة كل المهام على الآلات الافتراضية المتوفرة بزمن تنفيذ كلي أصغري Makespan علماً بأنَّ هاتين الخوارزميتين تندرجان ضمن الخوارزميات Meta-heuristic فائقة الاستدلال.

تمت نمذجة كل خوارزمية ومحاكاتها باستخدام بيئة المحاكاة cloudsim واستخدام لغة java من خلال بيئة التطوير Netbeans، وبعد ذلك تم اقتراح خوارزمية جديدة تحمل اسم ICA-SOS بدمج الخوارزميتين وذلك باعتماد خوارزمية SOS في مرحلتين من مراحل خوارزمية ICA بهدف الاستفادة من سرعة تقاربها نحو الحل الأمثلي والإبقاء على عملية المنافسة للاستيلاء على المستعمرات الجديدة في خوارزمية ICA، ثم الوصول إلى حل أمثلي أفضل من الحل الذي تعطيه كل منهما على حدة.

بينتْ النتائج – بعد تنجيز الخوارزمية المقترحة وتجريبها– تحسين الحل بنسبة 10% مقارنة بخوارزمية SOS، وبنسبة 25% مقارنة بخوارزمية ICA.

الكلمات المفتاحية: الحوسبة السحابية، جدولة المهام، خوارزمية SOS، خوارزمية ICA.

Abstract

The goal of scheduling in cloud computing is to allocate the available resources to execute required tasks with low execution time and effective use of resources. The role of the scheduling system is to make the appropriate decision for implementation of all tasks with high utilization of resources and low execution time, to achieve this we used optimization algorithms in this research.

This research implements two optimization algorithms in scheduling a set of tasks on virtual machines available in a cloud system. These two algorithms are the SOS (Symbiotic Organisms Search), and the ICA (Imperialist Competitive Algorithm), they aim to search for an optimal solution by scheduling all tasks on virtual machines available with a minimum Makespan, knowing that these algorithms are among the metahuristic algorithms.

Each algorithm was modeled and simulated using the cloudsim simulation environment and using the java language through the development environment Netbeans, and then a new algorithm called ICA-SOS was proposed by integrating the two algorithms by adopting the SOS algorithm in two stages of the ICA algorithm in order to take SOS advantage speed and maintain On the competition to capture new colonies in the ICA algorithm, and thus reach an optimal solution that is better than the solution given by each of them separately.

The results of the new ICA-SOS algorithm showed improvement rates for both algorithms, where the SOS was improved by 10%, and ICA was improved by 25%.

Keywords: cloud computing, task scheduling, Symbiotic Organism Search, SOS, Imperialist Competitive Algorithm, ICA.

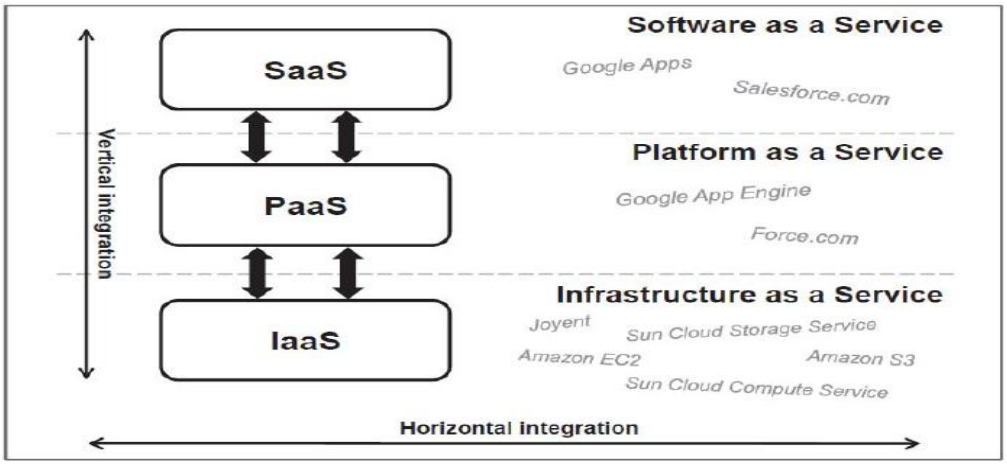
الفصل الأول: المقدمة

1.1 مقدمة

ظهرت العديد من الأبحاث المتعلقة بالبحث في البيانات والاستفادة من الشبكة بالشكل الأمثل، تماشياً مع التطور السريع في مجال الحوسبة واستخدام شبكة الانترنت وكمية البيانات الضخمة المتبادلة بشكل يومي على الشبكة، وبعد ظهور مفهوم الحوسبة السحابية كان لابد من البحث في هذا المجال وتطويره بما يلائم متطلبات العصر، وتعتمد الفكرة الأساسية للحوسبة السحابية على تقديم الخدمات للزبائن عن طريق شبكة الانترنت، ومع زيادة أعداد المستخدمين وزيادة الطلب على الخدمات، زادت معها المشكلات المتعلقة بجدولة هذه الطلبات وتنظيمها على المخدمات للوصول إلى أعلى استفادة وبزمن استجابة أقل مايمكن. وما زال البحث في تطوير خوارزميات للجدولة في الحوسبة السحابية الشغل الشاغل للعديد من الباحثين.

ظهرَ مصطلح الحوسبة السحابية في أعمال وشركات مختلفة قبل أن تقوم شركة Amazon عام 2002 بإطلاق سحابتها الأولى Amazon Web Services وفي عام 2006 أطلقت الشركة سحابتها الثانية وأسمتها AC2 لتقديم الخدمات التجارية على شبكة الانترنت، وفي عام 2009 ظهرت السحابة الأشهر التي نستخدمها بشكل يومي، وهي سحابة Google متمثلة بتطبيقاتها المتعددة التي تقدم خدمات عديدة للمستخدمين.

يعتمد نموذج الحوسبة السحابية على تقديم كل شيء ليكون خدمة Everything as a service وتنقسم هذه الخدمات إلى ثلاث طبقات رئيسية، يوضح الشكل 1-1 طبقات الحوسبة السحابية[11].



الشكل 1-1 طبقات الحوسبة السحابية.

- طبقة البرامج كونها خدمة Software as a service (SAAS).

هي أعلى مستوى في السحابة، تهتم بالتطبيقات المتعلقة بالمستخدم النهائي، مثل أنظمة البريد الالكتروني، تطبيقات إدارة علاقات العميل، البرمجيات المشتركة، أنظمة إدارة سير العمل.

- طبقة المنصّات كونها خدمة Platform as a service (PAAS)

هي الطبقة الثانية من طبقات الحوسبة السحابية، تتألف بشكل أساسي من مكتبات وبرامج وسيطة، وتحديثات وأدوات يحتاجها المطورون في تحديث الطبقة الأولى، وتستفيد هذه الطبقة من طبقة IAAS مثل البيئات الافتراضية التي توفرها تلك الطبقة وكذلك البرمجيات المطورة في المصادر الإفتراضية لطبقة IAAS.

- طبقة البنية التحتية كونها خدمة Infrastructure as a service (IAAS)

هي الطبقة الأخيرة من طبقات الحوسبة السحابية، يشار إليها أحياناً بطبقة الأجهزة كونها خدمة Hardware as a service (HAAS) تتضمن خدمات التخزين والنسخ الإحتياطية وقواعد البيانات والأمن.

توفّر هذه الطبقة البنية التحتية للأجهزة، فبدلاً من شراء المخدمات والبرمجيات يقوم العملاء بشراء هذه المصادر كونها خدمة مستقلة، ويتم حساب التكلفة على أساس المصادر المستخدمة. إن هذه الطبقة تستخدم تكنولوجيا الحوسبة الإفتراضية بشكل كبير Virtualization technology حيث تساعد على توفير الطاقة والتكلفة والمساحة في قواعد البيانات، ولأنها تحوي موارد السحابة سيتم تحصيص هذه الموارد، وجدولة المهام الواجب تنفيذها اعتماداً على خوارزميات الجدولة المدروسة.

1.2 الجدولة في الحوسبة السحابية

يدل مصطلح الجدولة في الحوسبة السحابية على عملية تحصيص الموارد المتاحة لتقوم بتنفيذ المهام المطلوبة، مع مراعاة الكفاءة الأعلى لاستخدام الموارد وبزمن تنفيذ أقل، حيث يتوفر العديد من الموارد في النظام السحابي وبالمقابل هناك العديد من المهام الواجب تنفيذها على هذه الموارد، وهنا يأتي دور نظام الجدولة لاتخاذ القرار المناسب الذي يضمن تنفيذ جميع المهام باستخدام أمثل للموارد وبزمن تنفيذ أقل مايمكن، من هنا كان

لخوارزميات الأمثَلة دور أساسي للوصول إلى هذا الهدف، وكان المجال واسعاً لاستخدام هذه الخوارزميات وتطويرها في مجال الجدولة في الحوسبة السحابية.

1.3 خوارزميات الأمثَلة Optimization Algorithms

تهدف الأمثَلة إلى الوصول إلى الحل الأفضل بين مجموعة من الحلول، وخوارزميات الأمثَلة Optimization Algorithms هي الخوارزميات التي تم بناؤها بهدف الوصول إلى حلول أفضل من الحلول التي تم التوصل إليها وفق المعايير المدروسة؛ كالزمن والأداء والسرعة وغيرها من المعايير. وتُستَخدم في مجالات متنوعة مثل علوم الحاسوب والإقتصاد والإدارة[10].

يمكن تقسيم خوارزميات الأمثَلة إلى قسمين رئيسيين:

خوارزميات قديمة أو تقليدية

هي خوارزميات أمثَلة قديمة متعارف عليها عند أوساط الرياضيين وفي المدارس والجامعات[9]، نذكر منها:

خوارزمية Newton-Raphson [16].

خوارزمية steepest descent [17].

خوارزمية gradient descent [18].

خوارزميات البرمجة الخطية Linear programming (LP)

خوارزميات البرمجة غير الخطية Nonlinear programming (NLP)

خوارزميات حديثة أو غير تقليدية

تنقسم إلى ثلاثة أنواع:

- خوارزميات الأمثَلة فائقة الاستدلال Meta-Heuristic Optimization Algorithms

وهي الخوارزميات التي تعتمد على قيم عشوائية في إنشائها وتسمى أيضاً خوارزميات تحسين عشوائية stochastic algorithms أو خوارزميات ذاتية الإرشاد، أو خوارزميات تطورية evolutionary algorithms (EAs)، وأيضاً الخوارزميات المستمدة من الطبيعة nature-inspired algorithms وتسمى أيضا بالخوارزميات فائقة الاستدلال metahuristic.

- خوارزميات تحسين معززة بالذكاء الإصطناعي AI-based Optimization Algorithms

وهي خوارزميات يدخل في تركيبها شبكات عصبونية ذكية neural networks (ANNs) أو متجهات آلات دعم التمييز support vector machines (SVMs) أو أنظمة المنطق الضبابي fuzzy systems (FSs).

- خوارزميات هجينة Hybrid Optimization Algorithms

يتم تصميم هذه الخوارزميات بدمج خوارزميتين أو أكثر، بهدف إيجاد حلول أفضل، وقد يكون التهجين بين أي نوع من الأنواع السابقة[9].

تم في بحثنا هذا تهجين خوارزميتين من نوع metahuristic هما خوارزمية البحث في الكائنات المتكافلة SOS وخوارزمية التنافس بين المستعمِرين ICA وذلك بالدمج بينهما، والتوصل إلى خوارزمية هجينة ICA-SOS، بحيث أصبح أداء الخوارزمية الهجينة أفضل من أداء كل خوارزمية وحدها في مسألة جدولة المهام على الآلات الافتراضية في نظام سحابي.

1.4 أهمية البحث

تأتي أهمية هذا البحث كونه يعمل على تطوير خوارزمية جدولة بالإعتماد على خوارزميتين لم يتم التطرق لهما سوية في أبحاث سابقة، من حيث زمن تنفيذ المهام على الأجهزة الإفتراضية المتاحة في النظام السحابي، حيث تم حقن إحدى الخوارزميتين ضمن مرحلتين من مراحل الخوارزمية الثانية بهدف الاستفادة من إيجابيات كلتا الخوارزميتين في الخوارزمية الهجينة المقترحة.

1.5 الهدف من البحث

إن الهدف الذي يسعى هذا البحث للتوصل إليه هو تطوير خوارزمية جدولة جديدة في البيئة السحابية، يكون فيها زمن تنفيذ جدولة المهام أقل من كلتا الخوارزميتين ICA, SOS.

1.6 المسألة التي يناقشها البحث

إن مسألة تقليل وقت تنفيذ المهام على الأجهزة الافتراضية في البيئة السحابية أمر ضروري لتوفير الوقت والجهد وللإستفادة من الموارد المتاحة بالشكل الأمثل، ومن هنا كان اختيار الخوارزميات فائقة الاستدلال Metaheuristic لحل مسألة الجدولة هو الاختيار الأنسب للوصول إلى حل أمثلي، نناقش في هذا البحث مسألة جدولة عدة مهام على عدة أجهزة إفتراضية متاحة في النظام السحابي، بحيث يتم تنفيذ جميع المهام وبوقت تنفيذ كلي makespan أصغري، ولقد أعطت كل خوارزمية وحدها زمن تنفيذ أمثلي لتنفيذ جميع المهام، وكان الحل الأمثلي الذي أعطته خوارزمية SOS أفضل من الحل الذي أعطته خوارزمية ICA بعد عدد من التكرارات، لكن لم يكن أيٌّ من الحلين هو الحل الأوحد والأفضل لجدولة عدة مهام على عدة آلات افتراضية متاحة.

نناقش في هذا البحث كيفية التوصل إلى حل أمثلي أفضل من الحل الذي توصلت إليه كل من الخوارزميتين، وذلك بدمج الخوارزميتين للإستفادة من ميزات كل منهما.

1.7 الدراسات المرجعية

تطرقت دراسات متعددة لموضوع جدولة المهام في الحوسبة السحابية، واعتمد بعضها على الدمج بين خوارزميتين للتوصل لنتائج أفضل.

تم في الورقة البحثية [7] دراسة موضوع الجدولة بالاعتماد على الدمج بين خوارزمية مستعمرة النمل ACO وخوارزمية SOS حيث تمت معالجة مشكلة ضعف فعالية جدولة المهام عند استخدام خوارزمية النمل وحدها، بالاستفادة من سرعة تقارب خوارزمية SOS.

كما تمّ في الورقة البحثية [8] دراسة موضوع جدولة المهام في الحوسبة السحابية لحل مشكلة ضعف فعالية جدولة المهام باستخدام خوارزمية ACO وحدها حيث تم دمج خوارزمية ICA للاستفادة من عملية التنافس بين الإمبرياليين للوصول لحل أمثلي أفضل من الحل الذي تم التوصل إليه باستخدام خوارزمية ACO وحدها.

أما الورقة البحثية [2] فقد تم استخدام خوارزمية SOS وحدها ونمذجتها لحل مسألة الجدولة في الحوسبة السحابية، وذلك للاستفادة من سرعة تقاربها من الحل الأمثلي في حالة حجم المهام الكبير.

كما تمَّ في الورقة البحثية [5] دراسة استخدام خوارزمية ICA لحل مسألة جدولة المهام في الحوسبة السحابية والاستفادة من عملية المنافسة بين الإمبرياليين.

استعرضنا في هذا الفصل مقدمة عن الجدولة في النظام السحابي وأهمية خوارزميات الأمثَلة واستخدامها في عدة مجالات بهدف تحسين الحلول والتوصل لنتائج أفضل؛ كما أشرنا لأربع دراسات مرجعية تطرقت لموضوع جدولة المهام في النظام السحابي واستخدمت خوارزميات الأمثَلة لتحقيق ذلك.

نستعرض في الفصل الثاني من هذا البحث شرح كل من خوارزمية SOS وخوارزمية ICA، وطريقة نمذجة كل خوارزمية كي تلائم عملية جدولة المهام في البيئة السحابية، وكذلك نشرح طريقة دمج الخوارزميتين والوصول للخوارزمية الهجينة ICA-SOS.

نستعرض في الفصل الثالث شرحاً عن طريقة بناء المشروع على بيئة المحاكاة cloudsim ضمن بيئة التطوير Netbeans والنتائج التي توصلنا إليها بتنفيذ كل من الخوارزميات ومقارنة النتائج.

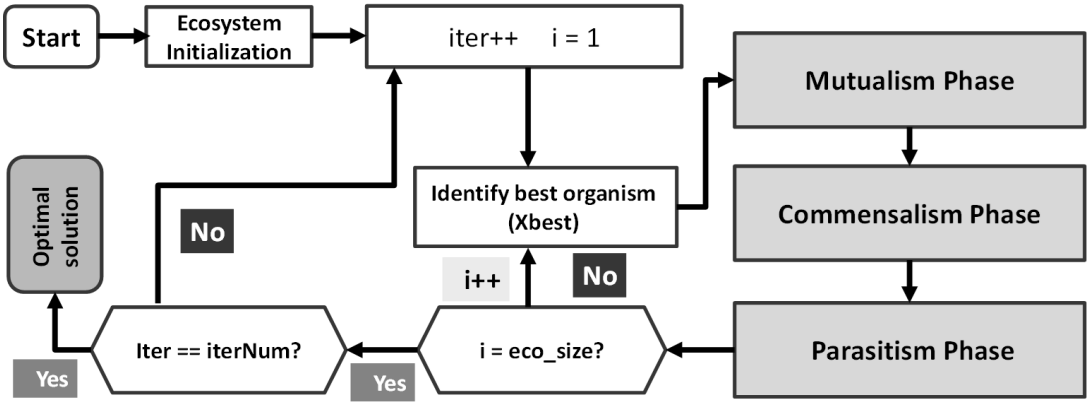
الفصل الثاني: الدراسة النظرية

2.1 خوارزمية البحث في الكائنات المتكافلة SOS

تعتمد خوارزمية SOS على محاكاة سلوك الكائنات الحية في الطبيعة، وهي من الخوارزميات فائقة الاستدلال metaheuristic تُستخدم لحل مشكلات التحسين العددي، وتبدأ الخوارزمية بمجموعة أولية تسمى النظام البيئي.

يتم إنشاء مجموعة من الكائنات في النظام البيئي الأولي بشكل عشوائي في مساحة البحث، يمثل كل كائن حي حلاً مرشحاً واحداً للمشكلة المقابلة، ويرتبط كل كائن حي في النظام البيئي بقيمة تكيف معينة fitness value التي تعكس درجة التكيف مع الهدف المنشود، ويتم في هذه الخوارزمية التحكم في توليد الحلول الجديدة من خلال محاكاة التفاعل بين كائنين في النظام البيئي، وتمر الخوارزمية بثلاث مراحل هي مرحلة التبادل Mutualism Phase ومرحلة التعايش Commensalism Phase ومرحلة التطفل Parasitism Phase تشبه نموذج التكافل البيئي في العالم الحقيقي[1].

يوضح الشكل 2-1 مخطط خوارزمية SOS.



الشكل 2-1مراحل خوارزمية SOS

يدلّ i في الشكل 2-1 لعدد الكائنات في النظام البيئي، ويتم تحديده كدخل للخوارزمية، ويدل iter على عدد التكرارات، ويتم تحديده كدخل للخوارزمية، بحيث يتم في كل تكرار المرور على كل الكائنات في النظام البيئي ecosystem.

مرحلة التبادل Mutualism phase

تتلخص هذه المرحلة بعلاقة مستفيد – مستفيد، حيث يتم التفاعل بين كائنين في النظام البيئي وينتج عن هذا التفاعل منفعة متبادلة للطرفين، مثل العلاقة بين الأزهار والنحل في الطبيعة، حيث تستفيد الأزهار من النحل بأنه يساعدها في عملية التلقيح بنقل حبات الطلع من زهرة لأخرى، وأيضاً يستفيد النحل من هذه العلاقة بأنه يتغذى على الأزهار ويقوم بإنتاج العسل.

بفرض Xi هو كائن حي ضمن النظام البيئي، يتم اختيار كائن حي آخر بشكل عشوائي Xj للتفاعل مع Xi، ينخرط كلا الكائنين في علاقة متبادلة بهدف زيادة ميزة البقاء المتبادل. يمثل كل من Xi و Xj حلاً مرشحاً يتم الاستبدال بهِ حلاً آخر بالإعتماد على المعادلات التالية التي تحسب الحلول المرشحة الجديدة بناءً على التعايش المتبادل بين الكائنين:

$$X_{inew} = X_i + rand(0,1) * (X_{best} - MutualVector * BF1) \quad (1)$$

$$X_{jnew} = X_j + rand(0,1) * (X_{best} - MutualVector * BF2) \quad (2)$$

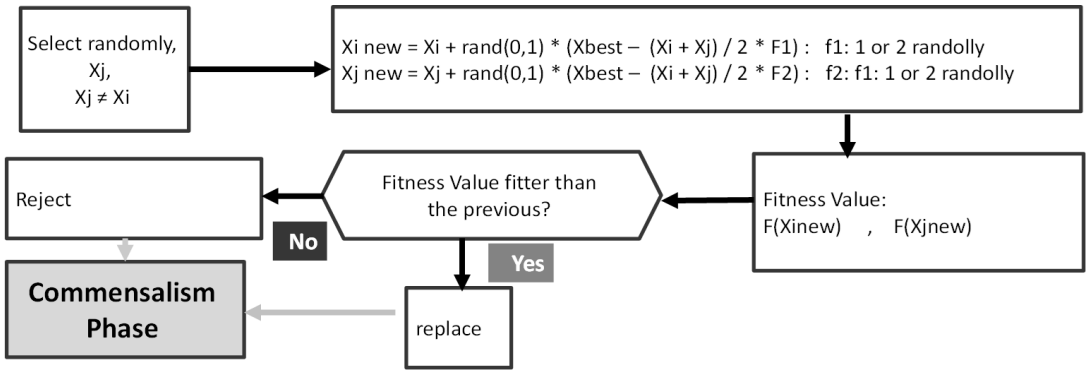
$$MutualVector = \frac{X_i + X_j}{2} \quad (3)$$

إذا كانت درجة تكيف X_{inew} أعلى من درجة تكيف X_i يتم الاستبدال بقيمة X_i قيمةً جديدةً X_{inew} وإلا تبقى القيمة السابقة ل X_i كما هي، وكذلك الأمر بالنسبة للكائن X_j [1].

تُحدد التوابع BF1 و BF2 فيما إذا كان الكائن الحي يستفيد جزئياً أو كلياً من التفاعل حيث الاستفادة متبادلة، تكون قيمة كل منهما إما 1 أو 2، يتم اختيار هذه القيمة بشكل عشوائي من أجل كل كائن حي، فهذه التوابع تعتمد على كمية الفائدة التي يحصل عليها كل من الكائنين [1].

يمثل Xbest الحل المرشح الأفضل الذي يمتلك أعلى درجة تكيف، ويمثل الهدف الذي تسعى بقية الكائنات للوصول إلى درجة تكيفه، حيث تحاول بقية الكائنات في النظام البيئي زيادة درجة تكيفها للبقاء على قيد الحياة، وفقاً لنظرية داروين "ستسود الكائنات الصالحة فقط" [1].

يوضح الشكل 2-2 مرحلة التبادل:



الشكل 2-2 مرحلة التبادل في خوارزمية SOS

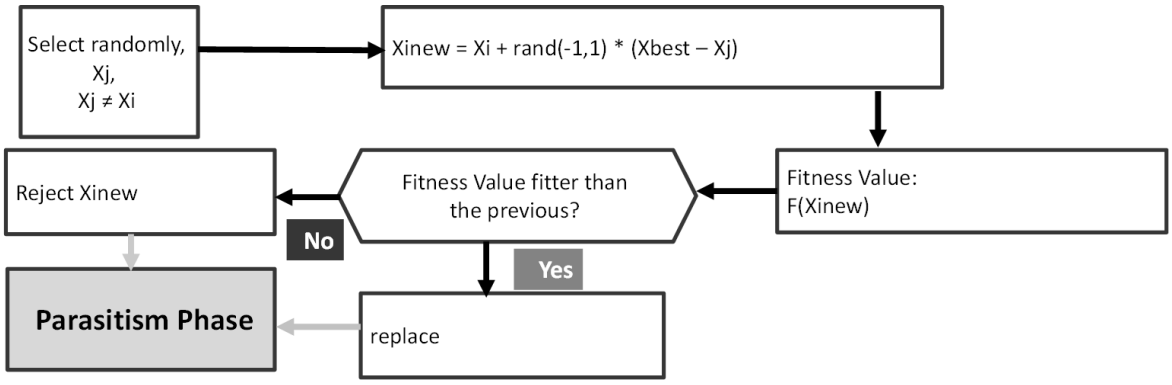
مرحلة التعايش Commensalism phase

تتلخص هذه المرحلة بعلاقة مستفيد – محايد، حيث يستفيد أحد الطرفين من العلاقة، بينما لايتأثر الطرف الآخر، مثال العلاقة بين أسماك ريمورا وأسماك القرش، حيث تتغذى أسماك ريمورا على بقايا طعام سمكة القرش فهي بذلك تستفيد وتحافظ على بقائها، بينما سمكة القرش لاتتأثر بذلك ولاتشعر بوجود سمكة الريمورا، تمثل المعادلة التالية حساب كائن جديد بالاستفادة من كائن آخر.

$$X_{inew} = X_i + rand(-1,1) * (X_{best} - X_j) \quad (4)$$

يمثل الجزء (Xbest-Xj) الفائدة التي يقدمها الكائن Xj لمساعدة Xi على البقاء، إذا كانت درجة تكيف Xinew أعلى من درجة تكيف Xi يتم استبدال قيمة Xi بالقيمة الجديدة Xinew وإلا تبقى القيمة السابقة كما هي [1].

يوضح الشكل 2-3 مرحلة التعايش:



الشكل 2-3 مرحلة التعايش في خوارزمية SOS

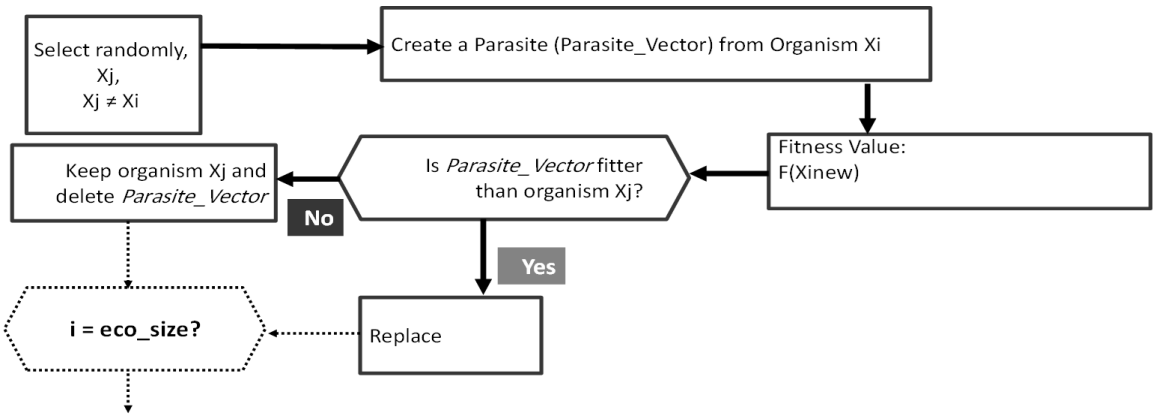
مرحلة التطفل Parasitism phase

تمثل هذه المرحلة علاقة مستفيد – متضرر، حيث يستفيد أحد الطرفين من العلاقة بينما يتضرر الطرف الآخر، مثال على التطفل هو طفيلي البلازموديوم، الذي يستخدم علاقته ببعوضة الأنوفيلة للانتقال بين المضيفين من البشر. بينما ينمو الطفيل ويتكاثر داخل جسم الإنسان، فإن مضيفه البشري يعاني من الملاريا وقد يموت نتيجة لذلك [1].

يتم إعطاء الكائن Xi دوراً مشابهاً لبعوضة الأنوفيلة من خلال إنشاء طفيلي اصطناعي يتم إنشاؤه في مساحة البحث عن طريق تكرار الكائن Xi، ثم تعديل الأبعاد المختارة عشوائياً باستخدام رقم عشوائي، ثم يتم اختيار الكائن Xj بشكل عشوائي من النظام البيئي ويعمل كمضيف لناقل الطفيلي، يحاول الطفيلي استبدال Xj في النظام البيئي، إذا كان لدى الطفيلي قيمة تكيف أفضل فسوف يقتل الكائن Xj ويتولى موقعه في النظام البيئي، أما إذا كانت قيمة تكيف Xj أفضل، فسيتمتع Xj بحصانة من الطفيلي ويبقى Xj في مكانه ويتم إهمال الطفيلي.

يتم توليد الطفيلي بالإعتماد على الكائن Xi من خلال توليد رقم عشوائي وضربه بقيمة Xi ثم اختيار كائن آخر Xj بشكل عشوائي ثم حساب درجة التكيف لكلا الكائنين واختيار الكائن الذي يحقق درجة تكيف أعلى وإهمال الكائن الآخر.

يوضح الشكل 2-4 مرحلة التطفل.



الشكل 2-4 مرحلة التطفل في خوارزمية SOS

يتم إعادة المراحل السابقة إلى أن ينتهي عدد التكرارات وتنتهي الخوارزمية، مع العلم أن دخل الخوارزمية هو عدد التكرارات وعدد الكائنات.

يتم الحصول في نهاية الخوارزمية على كائن يحقق نسبة تكيف أعلى، يمثل هذا الكائن الحل الأمثلي الذي استطاعت الخوارزمية إيجاده.

2.2 استخدام خوارزمية SOS في جدولة المهام في الحوسبة السحابية

توصيف المسألة:

يراد تنفيذ n مهمة (Cloudlet) على m جهاز افتراضي Vm بحيث يكون زمن التنفيذ الكلي Makespan أصغر مايمكن.

بفرض أن n=5 ، m=4 وأن الشكل العام للحل (الكائن الحي organism) هو

(C0,V2) (C1,V1) (C2,V3) (C3,V0) (C4,V3)

إن الحل الذي يتم البحث عنه هو تشكيلة من الإسنادات، يتم إسناد كل مهمة ci إلى جهاز افتراضي ما vj

بفرض أن الأرقام المعرّفة للمهام هي 0,1,2,3,4 وأن الأرقام المعرفة للأجهزة الافتراضية هي 0,1,2,3

إن التشكيلة التالية هي حل مقترح: (4,3) (3,0) (2,3) (1,1) (0,2)

وهذا يعني:

إسناد المهمة رقم 0 للجهاز الافتراضي رقم 2

إسناد المهمة رقم 1 للجهاز الافتراضي رقم 1

إسناد المهمة رقم 2 للجهاز الافتراضي رقم 3

إسناد المهمة رقم 3 للجهاز الافتراضي رقم 0

إسناد المهمة رقم 4 للجهاز الافتراضي رقم 3

كما يوضح الجدول 2-1

الجدول 2-1 إسناد خمس مهام لأربعة أجهزة إفتراضية

Cloudlets	0	1	2	3	4
VMs	2	1	3	0	3

إنَّ زمن التنفيذ الكلي Makespan هو مجموع أزمنة التنفيذ لكل ثنائية في الحل المقترح (الكائن الحي)، بفرض أن عدد الثنائيات k يكون زمن التنفيذ الكلي للكائن الحي:

Makespan = $\sum_{i=1}^{k} ETCi$ (5)

يمثل الحد ETC الزمن المتوقع لتنفيذ مهمة (Cloudlet) على جهاز افتراضي (VM)

$$ETC(i,j) = \frac{Length\ of\ cloudlet(i)}{MIPS\ of\ VM(j)} \quad (\text{second} * 10^{-6}) \quad (6)$$

طول المهمة Length of cloudlet يمثل عدد التعليمات التي تحتاجها المهمة ليتم تنفيذها، وهو مقدار يتم التعبير عنه بعدد صحيح موجب، أما MIPS يمثل عدد التعليمات التي يستطيع الجهاز الافتراضي VM تنفيذها مضروباً بمليون، (Million Instructions Per Second).

إن الحل النهائي الذي تبحث عنه الخوارزمية هو مجموعة إسنادات نمثلها بمجموعة ثنائيات (كائن حي) تحقق أعلى نسبة تكيف في النظام البيئي، أي التي تحقق أصغر زمن تنفيذ Makespan، وذلك بعد انتهاء عدد التكرارات الذي يتم تحديده عند البدء كدخل للخوارزمية.

إنَّ تبديل كائن بكائن آخر يعني تبديل الجهاز الافتراضي المقترح لتنفيذ مهمة ما بجهاز افتراضي آخر مقترح لتنفيذ المهمة نفسها في كائن آخر، في كل عمليات التبديل والتعديل على الكائن الحي(الحل المقترح)، مايتم تعديله هو الجهاز الافتراضي الذي سينفذ المهمة، أي المسقط الثاني من الثنائية فقط.

مثال:

الكائن1: (4,2) (3,0) (2,3) (1,2) (0,1)

الكائن2: (4,1) (3,1) (2,3) (1,3) (0,2)

بفرض أن زمن التنفيذ للكائن الأول هو 0.325 ms وزمن التنفيذ للكائن الثاني هو 0.299 ms ،إن زمن التنفيذ للكائن الثاني أفضل، لذلك يتم إهمال الكائن الأول من فضاء الحلول ويُستَبدَل بالكائن الثاني.

يصبح الكائن1 بعد التعديل:

(0,1=>2) (1,2=>3) (2,3=>3)(3,0=>1) (4,2=>1)

- بفرض تم تهيئة النظام البيئي بعشرة كائنات، يراد جدولة خمس مهام على أربعة أجهزة افتراضية، فضاء الحلول المرشح يكون مكوناً من عشرة أسطر تمثل عدد الكائنات(يتم تحديدها كدخل للخوارزمية) وخمسة أعمدة تمثل عدد المهام المراد تنفيذها على الآلات الافتراضية المتاحة، يتم توزيع الآلات الافتراضية في بداية الخوارزمية بشكل تراتبي على المهام، وهذا موضح في الشكل 2-5:

```
#1# Initial Population: size = 10
 Each organism represents a candidate solution as pairs of cloudlet and Vm (CloudletId, VmID):
 (0 , 0)   (1 , 1)   (2 , 2)   (3 , 3)   (4 , 0)   >>> Organism: [0] >> Makespan = 24.411193192335475
 (0 , 1)   (1 , 2)   (2 , 3)   (3 , 0)   (4 , 1)   >>> Organism: [1] >> Makespan = 24.278708048104097
 (0 , 2)   (1 , 3)   (2 , 0)   (3 , 1)   (4 , 2)   >>> Organism: [2] >> Makespan = 24.14339952156154
 (0 , 3)   (1 , 0)   (2 , 1)   (3 , 2)   (4 , 3)   >>> Organism: [3] >> Makespan = 24.005216294853305
 (0 , 0)   (1 , 1)   (2 , 2)   (3 , 3)   (4 , 0)   >>> Organism: [4] >> Makespan = 24.411193192335475
 (0 , 1)   (1 , 2)   (2 , 3)   (3 , 0)   (4 , 1)   >>> Organism: [5] >> Makespan = 24.278708048104097
 (0 , 2)   (1 , 3)   (2 , 0)   (3 , 1)   (4 , 2)   >>> Organism: [6] >> Makespan = 24.14339952156154
 (0 , 3)   (1 , 0)   (2 , 1)   (3 , 2)   (4 , 3)   >>> Organism: [7] >> Makespan = 24.005216294853305
 (0 , 0)   (1 , 1)   (2 , 2)   (3 , 3)   (4 , 0)   >>> Organism: [8] >> Makespan = 24.411193192335475
 (0 , 1)   (1 , 2)   (2 , 3)   (3 , 0)   (4 , 1)   >>> Organism: [9] >> Makespan = 24.278708048104097
 xBest : 24.005216294853305 to organism: 3
```

الشكل 2-5 شكل فضاء الحلول الأولي لعشرة كائنات

يتم تحديد أفضل كائن حي، وهو أول كائن يحقق أفضل قيمة للتابع المراد تحسينه وهو زمن التنفيذ الكلي Makespan في الحالة السابقة xbest = organism3

بعد تحديد أفضل كائن يتم تنفيذ مرحلة التبادل حيث يتم البدء بأول كائن organism0 ثم يتم اختيار كائن آخر عشوائياً (سطر آخر من فضاء الحلول)، ثم بتطبيق معادلات مرحلة التبادل (1) و (2) يتم حساب كائنين جديدين، كما يلي:

من المعادلات (1) و(2) و (3) تكون معادلتا مرحلة التبادل للكائنين Xi و Xj :

$$X_{inew} = X_i + rand(0,1) * \left(X_{best} - \frac{X_i + X_j}{2} * F_1\right) \quad (7)$$

$$X_{jnew} = X_j + rand(0,1) * \left(X_{best} - \frac{X_i + X_j}{2} * F_2\right) \quad (8)$$

بفرض:

$$X_i = (0,2)\ (1,1)\ (2,3)\ (3,0)\ (4,3)\ , X_j = (0,1)\ (1,2)\ (2,3)\ (3,3)\ (4,1)$$

$$X_{best} = (0,0)\ (1,1)\ (2,1)\ (3,0)\ (4,3)\ , rand(0,1) = 0.6, F_1 = 1$$

بحيث:

X_{best}: يتم حسابه وهو الكائن الذي يحقق أفضل قيمة لزمن التنفيذ.

X_i: هو الكائن الذي سيتم عليه تطبيق معادلات التبادل والتعايش والتطفل، يبدأ بأول كائن وينتهي بآخر كائن في النظام البيئي.

X_j: هو كائن مختلف عن الكائن X_i يتم اختياره عشوائياً من فضاء الحلول.

يتم تطبيق المعادلتين (7) و (8) على الكائنين X_i و X_j كما يلي:

حساب X_{inew}:

يتم أولاً حساب المقدار S1 استناداً إلى المعادلة (7) وتبديل كل من X_i, X_j, X_{best} بأرقام الأجهزة الافتراضية في كل منها(المسقط الثاني من كل ثنائية في الكائن).

$$S1 = \begin{bmatrix}2\\1\\3\\0\\3\end{bmatrix} + 0.6 * \left[\begin{bmatrix}0\\1\\1\\0\\3\end{bmatrix} - \frac{1}{2} * \begin{bmatrix}2+1\\1+2\\3+3\\0+3\\3+1\end{bmatrix} * 1\right] = \begin{bmatrix}1.1\\0.7\\1.8\\-0.9\\3.6\end{bmatrix}$$

ثم يتم مقاربة S1 ليمثل أعداداً صحيحة موجبة تمثل أرقام أجهزة افتراضية، لذلك يتم حساب القيمة المطلقة abs للمقدار السابق ثم التقريب إلى أقرب عدد صحيح من الأعلى ceil ثم باقي القسمة mod على عدد الأجهزة الافتراضية (في هذا المثال عددها أربعة، وتم ترقيم الأجهزة الافتراضية ابتداء من الصفر).

$$S1 = \begin{bmatrix}1.1\\0.7\\1.8\\-0.9\\3.6\end{bmatrix} => abs => \begin{bmatrix}1.1\\0.7\\1.8\\0.9\\3.6\end{bmatrix} => ceil => \begin{bmatrix}2\\1\\2\\1\\4\end{bmatrix} => mod\ 4\ \begin{bmatrix}2\\1\\2\\1\\0\end{bmatrix} => X_{inew}$$

يكون بالنتيجة الكائن الجديد Xinew هو الكائن ذو الإسنادات الجديدة التي تم حسابها من المقدار S1

$$X_{inew} = (0,2)\ (1,1)(2,2)(3,1)(4,0)$$

ثم يتم حساب Xjnew بالطريقة نفسها، وبعدها يتم حساب قيمة زمن التنفيذ لكل كائن: حسب المعادلة (5)

إذا كان $\left(Makespan(X_{inew}) < Makespan(X_i)\right)$ يتم استبدال Xi بالكائن الجديد Xinew وكذلك بالنسبة للكائن Xj.

يتم بعد ذلك الانتقال لمرحلة التعايش مع الأخذ بالاعتبار قيمة Xi الجديدة، حيث يتم اختيار كائن عشوائي آخر مختلف عن الكائن Xi (الذي من المحتمل أن يكون تم تعديله إلى كائن أفضل) ويتم تطبيق معادلة مرحلة التعايش (4) بالطريقة نفسها التي تم شرحها في مرحلة التبادل، ينتج كائن جديد، يتم حساب قيمة الزمن Makespan للكائن الجديد، إذا كانت قيمة الزمن أفضل(أقل) يتم الاستبدال بالكائن الجديد.

يتم بعد ذلك الانتقال لمرحلة التطفل، حيث يتم إنشاء كائن جديد بالإعتماد على الكائن Xi من خلال توليد رقم عشوائي وضربه بقيمة Xi ثم اختيار كائن آخر عشوائياً Xj ثم حساب قيمة زمن التنفيذ Makespan واختيار الكائن الذي يحقق زمن تنفيذ أفضل وإهمال الكائن الآخر.

يتم بعد ذلك الانتقال إلى الكائن التالي في النظام البيئي، وحساب قيمة أفضل كائن يحقق أصغر قيمة لزمن التنفيذ، ثم تطبيق المراحل الثلاث السابقة بالطريقة نفسها، وهكذا إلى أن يتم المرور على جميع الكائنات في النظام البيئي، وبذلك يكون قد تم تطبيق الخوارزمية بتكرار واحد، ويتم الانتقال إلى التكرار التالي، ثم تطبيق مراحل الخوارزمية على جميع الكائنات، وهكذا إلى انتهاء عدد التكرارات.

الحل الأمثلي الذي تصل إليه الخوارزمية هو الكائن الحي الذي يحقق أفضل قيمة لزمن التنفيذ عند توقف الخوارزمية.

اعتماداً على شرح مراحل الخوارزمية وطريقة نمذجتها في جدولة المهام. نكتب الرماز المعبّر عن مراحل خوارزمية SOS بالشكل التالي:

Iter: integer // iteration number: input
N: integer // organisms number: input

iteration_number ← *0*
xbest ←*0*
Do
iteration_number ←*iteration_number +1*
i←*0*
Do
i←*i +1*
For j =1 to N
$if\ Makespan(x_j) < Makespan(x_{best}) then$
$x_{best} \leftarrow x_j$
End if
End for
//mutualism phase
Randomly select x_j *with* $i \neq j$
$F_1 \leftarrow$ *1 or 2* , $F_2 \leftarrow$ *1or 2*
$r_1 \leftarrow$ *rand(0,1)* , $r_2 \leftarrow$ *rand(0,1)*

$$x_{inew} = x_i + r_1 * \left(x_{best} - \frac{x_i + x_j}{2} * F_1\right)$$

$$x_{jnew} = x_j + r_2 * \left(x_{best} - \frac{X_i + X_j}{2} * F_2\right)$$

$if\ Makespan(x_{inew}) < Makespan(x_i)\ then$
$x_i \leftarrow x_{inew}$
End if
$if\ Makespan(x_{jnew}) < Makespan(x_j) then$
$x_j \leftarrow x_{jnew}$
End if
//commensalism phase
Randomly select x_j *with* $i \neq j$
$r_1 \leftarrow$ *rand(-1,1)*

$$x_{inew} = x_i + r_1 * (x_{best} - x_j)$$

$if\ Makespan(x_{inew}) < Makespan(x_i)\ then$
$x_i \leftarrow x_{inew}$
End if
//parasitism phase
Randomly select x_j *with* $i \neq j$
Create a parasite vector x_p *from* x_i *using random number*
$if\ Makespan(x_p) < Makespan(x_j)\ then$
$x_j \leftarrow x_p$
End if
While i <= N
While iteration_number <= iter

2.3 خوارزمية التنافس بين المستعمرين Imperialist Competitive Algorithm

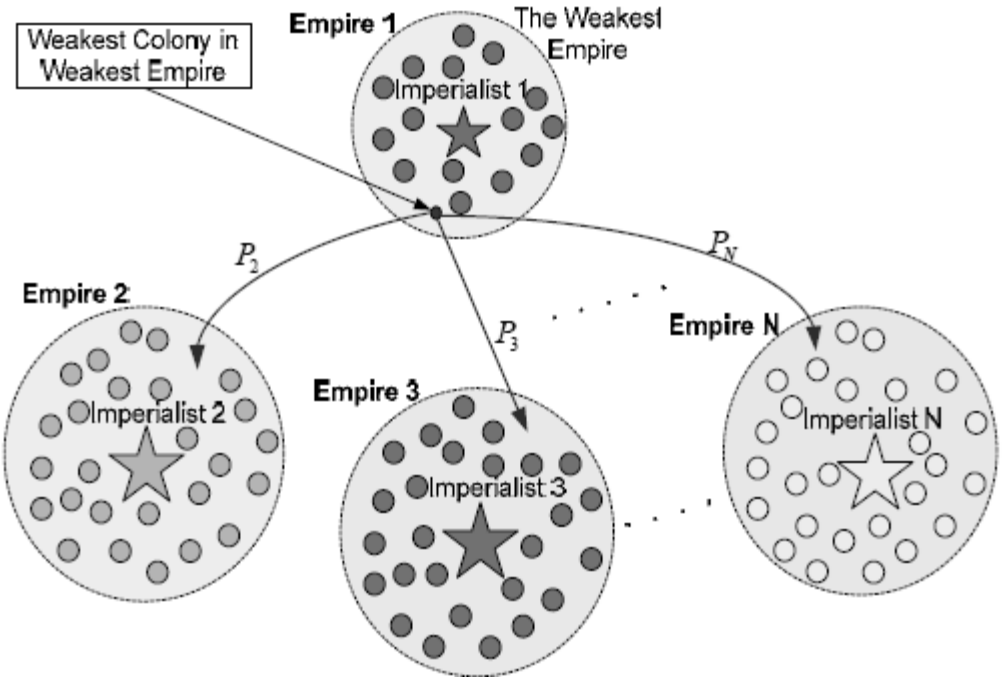
تم اقتراح هذه الخوارزمية لحل مشاكل الأمثلة العددية [3] وهي مستوحاة من التنافس الإمبريالي، تبدأ الخوارزمية بمجموعة أولية، يطلق على كل عنصر فيها اسم الدولة Country التي تنقسم إلى نوعين، المستعمرات Colonies والإمبرياليين imperialists اللذين يشكلان معاً إمبراطوريات empires.

تعتمد فكرة الخوارزمية على المنافسة الإمبريالية بين الإمبراطوريات Imperialistic competition للوصول بالنهاية إلى إمبراطورية واحدة قوية تكون قد سيطرت في مراحل سابقة على جميع مستعمرات الإمبراطوريات الضعيفة.

تبدأ الخوارزمية بتعريف فضاء الحلول population الذي يمثل مجموعة الدول countries بحيث تنقسم الدول إلى إمبرياليين imperialists ومستعمرات Colonies يتم تصنيف الدولة كإمبريالية حسب أفضليتها وطاقتها، وتكون أفضل الدول هي إمبريالية imperialist وماتبقى تصنف كمستعمَرة Colony حيث كل مجموعة مستعمَرات تتبع لإمبريالية ما، يمثل الشكل 2-6 الإمبريالي بنجمة والمستعمرة بدائرة، ويمثل كل من الإمبريالي والمستعمرَة دولة، أما الإمبراطورية فهي مجموعة مستعمَرات وإمبريالي واحد.

يتم تقسيم المستعمرات على الإمبرياليين وفقاً لطاقتها، تشكل المستعمرات والإمبرياليين ما يسمى إمبراطورية empire، ولكل إمبراطورية طاقة كلية تعتمد على طاقة الدولة الإمبريالية وطاقة مستعمراتها.

وبعد تقسيم جميع المستعمرات على الإمبرياليين، تبدأ هذه المستعمرات بالتحرك نحو دولتهم الإمبريالية ذات الصلة، ثم تبدأ المنافسة الإمبريالية imperialistic competition بين كل الإمبراطوريات، بحيث كل إمبراطورية لاتتمكن من زيادة طاقتها (أو على الأقل تمنع نقصان طاقتها) سيتم استبعادها من المنافسة [3]، وستؤدي المنافسة الإمبريالية تدريجياً إلى زيادة قوة الإمبراطوريات القوية ونقصان قوة الإمبراطوريات الأضعف بحيث تفقد الإمبراطوريات الضعيفة قوتها وتنهار، وبالنهاية ينتج إمبراطورية واحدة يكون للإمبريالي فيها الطاقة الأكبر وهو يمثل الحل الذي تجده الخوارزمية.

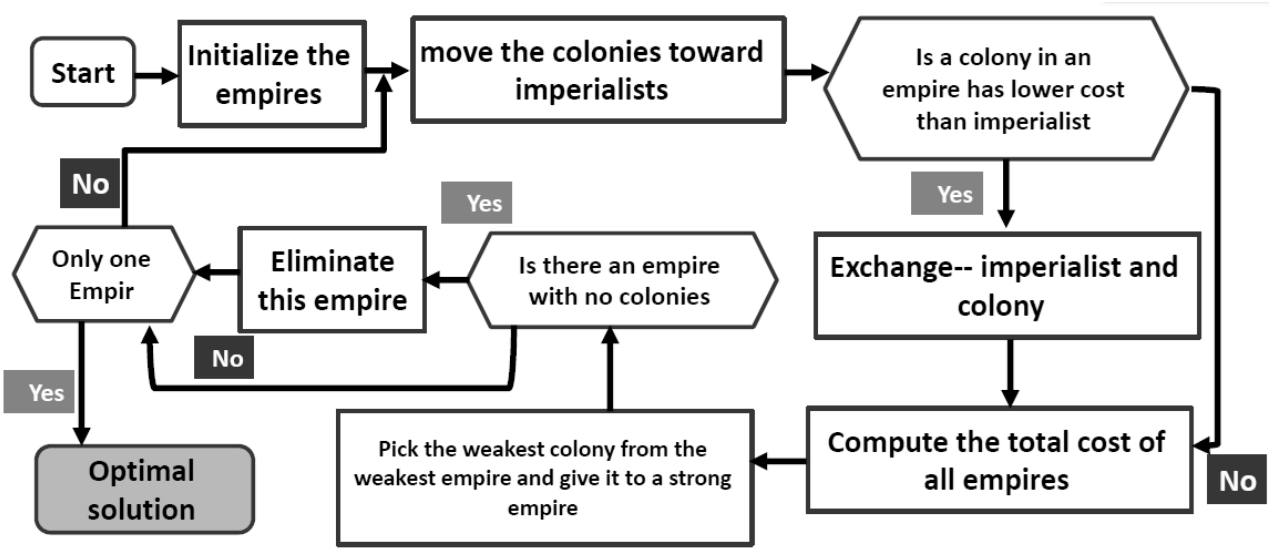


الشكل 2-6 الإمبرياليون والمستعمرات التابعة لهم والإمبراطوريات

من نقاط ضعف هذه الخوارزمية تقاربها السريع من local optima الذي يمثل حلاً مرشحاً لكن ليس هو الحل النهائي global optima لكنها تستطيع الخروج منه بعد عدة تكرارات في مرحلة التنافس الإمبريالي، وقد يؤدي ذلك إلى استهلاك وقت للوصول إلى إمبراطورية واحدة.

يُعدُّ الوصول إلى الحل الأمثلي المحلي local optima فخاً أو مصيدةً traps لأغلب خوارزميات البحث التقليدية، وبهذا تتغلب خوارزميات البحث ال metaheuristic بقدرتها على الهروب من الأفخاخ بسبب اعتمادها على قيم عشوائية وعلى عدد من التكرارات [4].

يوضح الشكل 2-7 مراحل خوارزمية ICA [4].



الشكل 2- 7 مخطط خوارزمية ICA

2.3.1 مراحل خوارزمية ICA

- **تهيئة الإمبراطوريات وتوزيع المستعمرات عليها**

يتم تشكيل إمبراطوريات عددها N_{imp} تتألف كل إمبراطورية من عدة دول، العدد الكلي للدول هو N_{pop} وهو يمثل حجم فضاء الحلول الأولي، يُعدُّ كل من N_{imp} و N_{pop} محددات دخل الخوارزمية.

تكون الدولة إما إمبريالي imperialist وإما مستعمرة colony وتمتلك كل إمبراطورية إمبريالي واحد وعدة مستعمرات تتبع لهذا الإمبريالي، بالتالي عدد الإمبرياليين يساوي عدد الإمبراطوريات N_{imp} أما عدد المستعمرات الكلي N_{col} يساوي الفرق بين عدد الدول وعدد الإمبرياليين:

$$N_{col} = N_{pop} - N_{imp}$$

يتم توزيع المستعمرات على الإمبرياليين استناداً إلى طاقة الإمبريالي:

$$C_n = c_n - max_i\{c_i\} \quad (9)$$

حيث c_n هو قيمة تابع الكلفة للإمبريالي n أما C_n فهو قيمة تابع الكلفة للإمبريالي n بعد التقييس.

إنَّ تابع الكلفة في مسألة الجدولة في هذا البحث هو التابع الذي يحسب زمن التنفيذ الكلي Makespan للإمبريالي n، كما توضح المعادلة:

$$Makespan_n = makespan_n - max_i\{makespan_i\} \quad (10)$$

حيث يتم حساب زمن التنفيذ لكل مستعمرة من المستعمرات التابعة للإمبريالي n ثم حساب القيمة العظمى max لقيم أزمنة المستعمرات، ثم طرحها من قيمة زمن التنفيذ للإمبريالي n.

$$p_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right| \quad (11)$$

تمثل p_n طاقة الإمبريالي التي سيتم على أساسها حساب عدد المستعمرات التابعة لهذا الإمبريالي كما يلي:

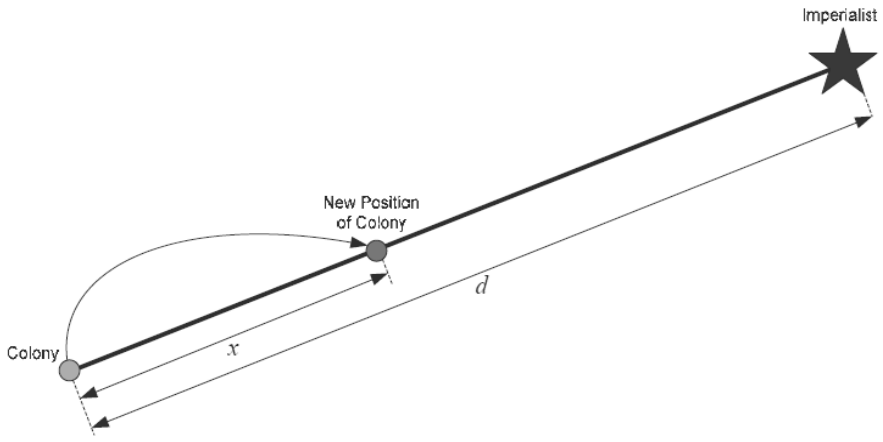
$$N.C_n = round\{p_n.N_{col}\} \quad (12)$$

تمثل $N.C_n$ عدد المستعمرات الأولي في الإمبراطورية n ذات الإمبريالي n

تمثل N_{col} عدد المستعمرات الكلي.

- **تحرك المستعمرات باتجاه الإمبريالي (Assimilating)**

تبدأ المستعمرات في كل إمبراطورية بالتحرك نحو الإمبريالي في هذه الإمبراطورية كما يوضح الشكل 2-8، قد تتحرك المستعمرة إلى موضع جديد تكون قيمة الكلفة عند هذا الموضع أقل(أفضل) من قيمة الكلفة للإمبريالي[3].



الشكل 2-8 تحرك المستعمرة نحو الإمبريالي

يتم نمذجة هذه الحركة رياضياً باختيار المتحول العشوائي x الذي يتم حسابه باستخدام تابع توزيع عشوائي وليكن U حيث

$$x \sim U(0, \beta * d)$$

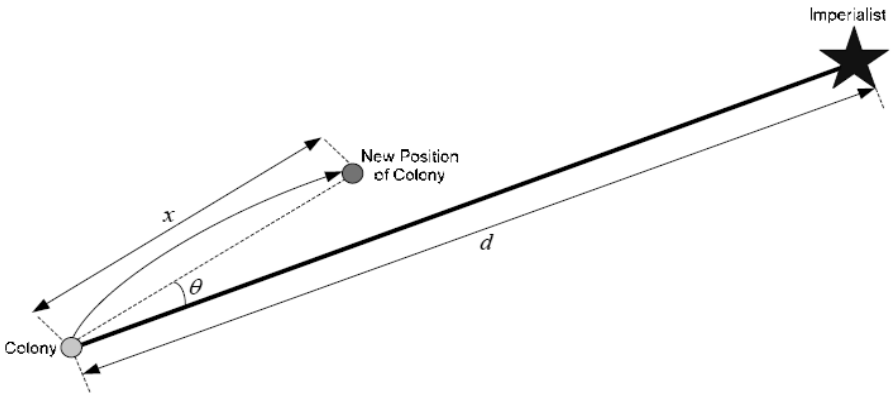
تمثل d المسافة بين المستعمرة والإمبريالي.

تمثل β مقدار أكبر من الواحد (يساوي تقريباً 2).

قد تتحرك المستعمرة باتجاه الإمبريالي وفق زاوية θ يتم توليدها وفق تابع توزيع عشوائي

$$\theta \sim U(-\gamma, \gamma)$$

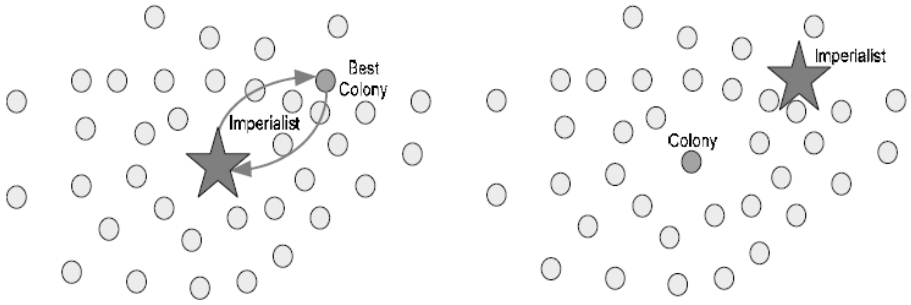
حيث $\gamma \sim \frac{\pi}{4}$ كما يوضح الشكل 2-9.



الشكل 2-9 تحرك المستعمرة نحو الإمبريالي باتجاه عشوائي

- **تبديل مواقع المستعمرة والإمبريالي (Exchanging)**

عندما تتحرك المستعمرة إلى موضع جديد قيمة تابع الكلفة عنده أقل(أفضل) من قيمة تابع الكلفة للإمبريالي، يتم تبديل مواضع المستعمرة والإمبريالي، وتصبح المستعمرة إمبريالياً، ويصبح الإمبريالي القديم مستعمرةً كما يوضح الشكل 2-10، وتتابع الخوارزمية خطواتها مع الإمبريالي الجديد [3].



الشكل 2-10 تبديل مواضع المستعمرة والإمبريالي

- **حساب الطاقة الكلية للإمبراطورية**

طاقة الإمبراطورية تتأثر بشكل رئيسي بطاقة الإمبريالي، وتؤدي بقية المستعمرات دوراً صغيراً في حساب طاقة الإمبراطورية [3] كما توضح المعادلة:

$$T.C_n = Cost(Imperialist_n) + \xi mean\{Cost(Colonies\ of\ empire_n)\}\ (13)$$

تمثل $T.C_n$ الكلفة الكلية للإمبراطورية n ويمثل ξ مقداراً صغيراً موجباً أصغر من الواحد، يعبر عن كمية تأثير كلفة المستعمرات في الكلفة الكلية للإمبراطورية.

- **التنافس الإمبريالي (Imperialistic competition)**

يتم في هذه المرحلة التنافس بين الإمبراطوريات للحصول على مستعمرات، يتم اختيار الإمبراطورية الأضعف والاستيلاء على أضعف مستعمرة فيها من قِبل إمبراطوية قوية، حيث تتنافس الإمبراطوريات للحصول على المستعمرة الضعيفة.

إن تحديد الإمبراطوريات القوية يتم بحساب إمكانية استيلاء كل إمبراطورية، يتم أولاً حساب كلفة الإمبراطورية n

$$N.T.C_n = T.C_n - max_i\{T.C_i\}\ (14)$$

يمثل $N.T.C_n$ قيمة تابع الكلفة للإمبراطورية n بعد التقييس.

يمثل $T.C_n$ قيمة تابع الكلفة للإمبراطورية n

يتم بعدها حساب إمكانية استيلاء كل إمبراطورية كما في المعادلة:

$$p_{p_n} = \left| \frac{N.T.C_n}{\sum_{i=1}^{N_{imp}} N.T.C_i} \right|\ (15)$$

يتم بعدها تقسيم المستعمرات على الإمبراطوريات بحسب إمكانية استيلاء كل إمبراطورية لذلك يتم حساب الشعاع P

$$P = \left[p_{p_1},\ p_{p_2}, p_{p_3}, \dots, p_{p_{N_{imp}}}\right]\ (16)$$

ثم يتم إنشاء الشعاع R بحجم الشعاع P نفسه ويحتوي كل مسقط فيه قيمة عشوائية بين الصفر والواحد:

$$R = \left[r_1,\ r_2, r_3, \dots, r_{N_{imp}}\right]\ (17)$$

$$r_1,\ \ r_2, r_3, \dots, r_{N_{imp}} \sim U(0,1)$$

ثم يتم تشكيل شعاع الفرق بين الشعاعين السابقين:

$$D = P - R = \left[p_{p_1} - r_1,\quad p_{p_2} - r_2,\quad p_{p_3} - r_3,\ \ \dots,\quad p_{p_{N_{imp}}} - r_{N_{imp}}\right]$$

$$D = \left[D_1,\ \ D_2, D_3, \dots, D_{N_{imp}}\right] \quad (18)$$

القيمة الأكبر من D ذات الدليل n توافق الإمبراطورية ذات الدليل نفسه والتي تمتلك أعلى إحتمالية استيلاء على المستعمرة الأضعف في الإمبراطورية الأضعف[3].

يتم إعادة حساب الإمبراطورية الأضعف والإمبراطورية ذات احتمال الاستيلاء الأعلى وتستمر المنافسة.

- **إهمال الإمبراطورية الأضعف**

بعد عدة عمليات استيلاء قد تفقد إمبراطورية ما جميع مستعمراتها وعندها يتم استبعاد هذه الإمبراطورية من المنافسة ويتناقص عدد الإمبراطوريات، ثم يتم الرجوع إلى مرحلة تحرك المستعمرات باتجاه الإمبريالي في كل إمبراطورية وتستمر المنافسة، إلا إذا بقي إمبراطورية واحدة فقط عندها تتوقف الخوارزمية.

- **توقف الخوارزمية عند الوصول لإمبراطورية واحدة**

تتوقف خوارزمية التنافس الإمبريالي عند الوصول إلى إمبراطورية واحدة، ويكون الحل الأمثلي الذي وصلت إليه الخوارزمية هو الإمبريالي في هذه الإمبراطورية الوحيدة، والذي يمتلك أفضل قيمة لتابع الكلفة.

في حالة الجدولة يمثل الإمبريالي في الإمبراطورية الوحيدة بعد توقف الخوارزمية الحل الأمثلي الذي وصلت إليه الخوارزمية، وهو عبارة عن مجموعة الثنائيات التي تمثل إسناد كل مهمة إلى جهاز إفتراضي معين بحيث يكون زمن التنفيذ الكلي لمجموعة الإسنادات أصغر مايمكن.

2.4 استخدام خوارزمية ICA في جدولة المهام في الحوسبة السحابية

توصيف المسألة:

تنفيذ n مهمة (Cloudlet) على m جهاز افتراضي Vm بحيث يكون زمن التنفيذ الكلي Makespan أصغر مايمكن.

بفرض أن n=5 ، m=4 وأن الشكل العام للحل (الدولة country) هو

(C0,V2) (C1,V1) (C2,V3) (C3,V0) (C4,V3)

إن الحل الذي يتم البحث عنه هو تشكيلة من الإسنادات، يتم إسناد كل مهمة c_i إلى جهاز افتراضي ما v_j

بفرض أن الأرقام المعرفة للمهام هي 0,1,2,3,4 وأن الأرقام المعرفة للأجهزة الافتراضية هي 0,1,2,3

زمن التنفيذ الكلي Makespan هو مجموع أزمنة التنفيذ لكل ثنائية في الحل المقترح (الدولة country)، بفرض أن عدد الثنائيات k يكون زمن التنفيذ الكلي:

$$\text{Makespan} = \sum_{i=1}^{k} ETCi$$

يمثل الحد ETC الزمن المتوقع لتنفيذ مهمة (Cloudlet) على جهاز افتراضي (VM)

$$ETC(i,j) = \frac{Length\ of\ cloudlet(i)}{MIPS\ of\ VM(j)} \quad (\text{second} * 10^{-6})$$

إن الحل النهائي الذي تتوصل إليه الخوارزمية هو تشكيلة من الثنائيات (دولة country) تحقق أصغر زمن تنفيذ Makespan، وتكون الدولة في هذه الحالة هي الإمبريالي في الإمبراطورية، وذلك عند الوصول لإمبراطورية واحدة وعندها تتوقف الخوارزمية.

في كل عمليات التبديل والتعديل على الحل، مايتم تعديله هو الجهاز الافتراضي الذي سينفذ المهمة، أي المسقط الثاني من الثنائية فقط.

دخل الخوارزمية هو عدد الإمبراطوريات N_{imp} وعدد الدول N_{pop}

2.4.1 تهيئة الإمبراطوريات وحساب عدد المستعمرات في كل إمبراطورية في مسألة الجدولة

بعد أن تم إنشاء فضاء الحلول الأولي الذي يمثل كل سطر فيه دولة country (حل مرشح) يتم أولاً تحديد الإمبرياليين وهم الأسطر التي تحقق أفضل قيمة لزمن التنفيذ Makespan ثم يتم توزيع بقية الدول (أصبح اسمهم مستعمرات colonies) على الإمبرياليين.

مثال: جدولة عشر مهام (Cloudlets) على تسعة أجهزة إفتراضية (Virtual machins)، فضاء الحلول الأولي يكون كما هو موضح في الشكل 2-11 حيث تم تحديد دخل الخوارزمية كما يلي:

$N_{pop} = 10, N_{imp} = 3$

عدد الأسطر هو عدد الدول

عدد الأعمدة هو عدد المهام الواجب جدولتها (مرقمة من 0 إلى 9)

```
#1# Initial Population: size = 10 = number of countries
 Each country represents a candidate solution as pairs row of cloudlet and Vm (CloudletId, VmID):
(0,0)(1,1)(2,2)(3,3)(4,4)(5,5)(6,6)(7,7)(8,8)(9,0) >>> Country: [0] >> Makespan = 86.53185507303155
(0,1)(1,2)(2,3)(3,4)(4,5)(5,6)(6,7)(7,8)(8,0)(9,1) >>> Country: [1] >> Makespan = 75.03727383727384
(0,2)(1,3)(2,4)(3,5)(4,6)(5,7)(6,8)(7,0)(8,1)(9,2) >>> Country: [2] >> Makespan = 71.80935926818279
(0,3)(1,4)(2,5)(3,6)(4,7)(5,8)(6,0)(7,1)(8,2)(9,3) >>> Country: [3] >> Makespan = 69.07668279432986
(0,4)(1,5)(2,6)(3,7)(4,8)(5,0)(6,1)(7,2)(8,3)(9,4) >>> Country: [4] >> Makespan = 66.28051425698484
(0,5)(1,6)(2,7)(3,8)(4,0)(5,1)(6,2)(7,3)(8,4)(9,5) >>> Country: [5] >> Makespan = 63.3515895868837
(0,6)(1,7)(2,8)(3,0)(4,1)(5,2)(6,3)(7,4)(8,5)(9,6) >>> Country: [6] >> Makespan = 60.29523678935443
(0,7)(1,8)(2,0)(3,1)(4,2)(5,3)(6,4)(7,5)(8,6)(9,7) >>> Country: [7] >> Makespan = 57.12886068180186
(0,8)(1,0)(2,1)(3,2)(4,3)(5,4)(6,5)(7,6)(8,7)(9,8) >>> Country: [8] >> Makespan = 53.869573563691205
(0,0)(1,1)(2,2)(3,3)(4,4)(5,5)(6,6)(7,7)(8,8)(9,0) >>> Country: [9] >> Makespan = 86.53185507303155
```

الشكل 2-11 فضاء حلول أولي لخوارزمية ICA في حالة عشر دول وثلاث إمبراطوريات

يتم تحديد أفضل ثلاث دول (عدد الإمبراطوريات 3 بحسب دخل الخوارزمية)، وفقاً لقيمة زمن التنفيذ Makespan المحسوب والمسجل بجانب كل سطر من الفضاء الأولي السابق.

إن أفضل ثلاث قيم لزمن التنفيذ في حالتنا هي أصغر ثلاث قيم وهي القيم الموافقة للدول 6 – 7 – 8 كما هو موضح في الشكل 2-11 حيث يتم اعتبار هذه الدول إمبرياليين وسيتم توزيع بقية الدول عليهم بحسب طاقة كل منهم.

يتم بعدها حساب عدد المستعمرات التي سيتم تخصيصها لكل إمبريالي، عدد المستعمرات التي سيتم توزيعها هو:

$$N_{col} = N_{pop} – N_{imp} = 10 -3 = 7$$

يتم حساب طاقة الإمبريالي وعدد المستعمرات التابعة لكل إمبريالي بحسب المعادلات (11) و (12).

تم تقسيم المستعمرات كما يوضح الشكل 2-12 وهو صورة شاشة من تنفيذ خوارزمية ICA على بيئة التطوير.

```
Pn:  [0.3698933280093466, 0.3329826009154299, 0.29712407107522354]
sum colonies after calculating: 7.0 , Ncol =  7
number Of Colonies In each Empire: [3.0, 2.0, 2.0]
```

الشكل 2-12عدد المستعمرات التابعة لكل إمبريالي وطاقة كل إمبريالي.

بالتالي يكون الرماز المعبّر عن مراحل خوارزمية ICA في جدولة المهام كما يلي:

Empires: integer // empires number: input
Countries: integer // countries number: input
Empires_number ← Empires
Do
C_e ← colonies number in each Empire
For e =1 to Empires_number
For j =1 to C_e
Move the colonies toward their relevant imperialist (Assimilating).
if $Makespan(colony_j) < Makespan(imperialist_e)$ then
$imperialist_e \leftarrow colony_j$
End if
End for
End for
Compute the total cost of each Empire
$Empire_w$ ← the weakest Empire
$Empire_s$ ← the stronger Empire
WeakestC ← weakest colony in $Empire_w$
$Empire_w$_Colonies ← $Empire_w$_Colonies - WeakestC
$Empire_s$_Colonies ← $Empire_s$_Colonies + WeakestC
For k = 1 to Empires_number
if ($empire_k$ has no colonies) then
Empires_number ← Empires_number -1

```
  break
End if
  End for
While Empires_number > 1
```

2.5 الخوارزمية الهجينة ICA-SOS

بعد دراسة كل من الخوارزميتين كلٌّ على حدة، وجدنا أن خوارزمية SOS تتقارب من الحل الأمثلي بشكل أسرع حيث تصل إلى حل أمثلي بعد عدد محدد من التكرارات، ولاتصل لحل أفضل حتى لو ازداد عدد التكرارات عن هذا الحد، سنرى ذلك في فصل التطبيق العملي، وبالمقابل فإن خوارزمية ICA تأخذ وقتاً لإنهاء عمليات التنافس الإمبريالي ولاسيما عندما تكون الإمبراطوريات متقاربة في طاقتها.

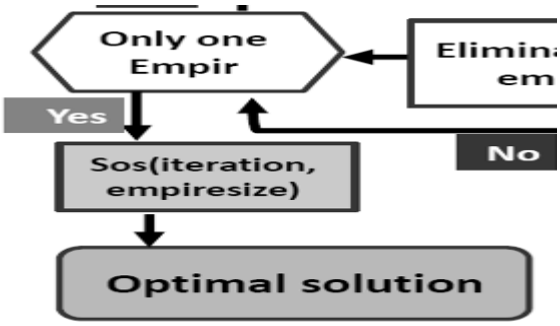
مما سبق يمكننا أن نتوقع بأن إدراج خوارزمية SOS في مرحلة تقارب المستعمرة من الإمبريالي ضمن خوارزمية ICA، على اعتبار أن الإمبراطورية هي فضاء حلول أولي لخوارزمية SOS فإن هذا الإدراج سيحسن من خوارزمية ICA.

إن إدراج خوارزمية SOS بعد مرحلة تهيئة الإمبراطوريات وقبل عملية تقارب المستعمرة من الإمبريالي يساعد على تجهيز امبراطورية محسّنة أساساً قبل أن يتم اختبار تقارب المستعمرات، بحيث يكون دخل Input الخوارزمية التكافلية SOS هو عدد التكرارات Iteration وهو مقدار يتم تحديده عند البداية كدخل للخوارزمية المقترحة، أما عدد الكائنات (مقدار الدخل الثاني للخوارزمية SOS) هو عدد الدول في الإمبراطورية (empireSize) التي سيتم تطبيق الخوارزمية التكافلية SOS عليها، الشكل 2-13.

الشكل 2-13 إدراج خوارزمية SOS قبل مرحلة تقارب المستعمرة من الامبريالي.

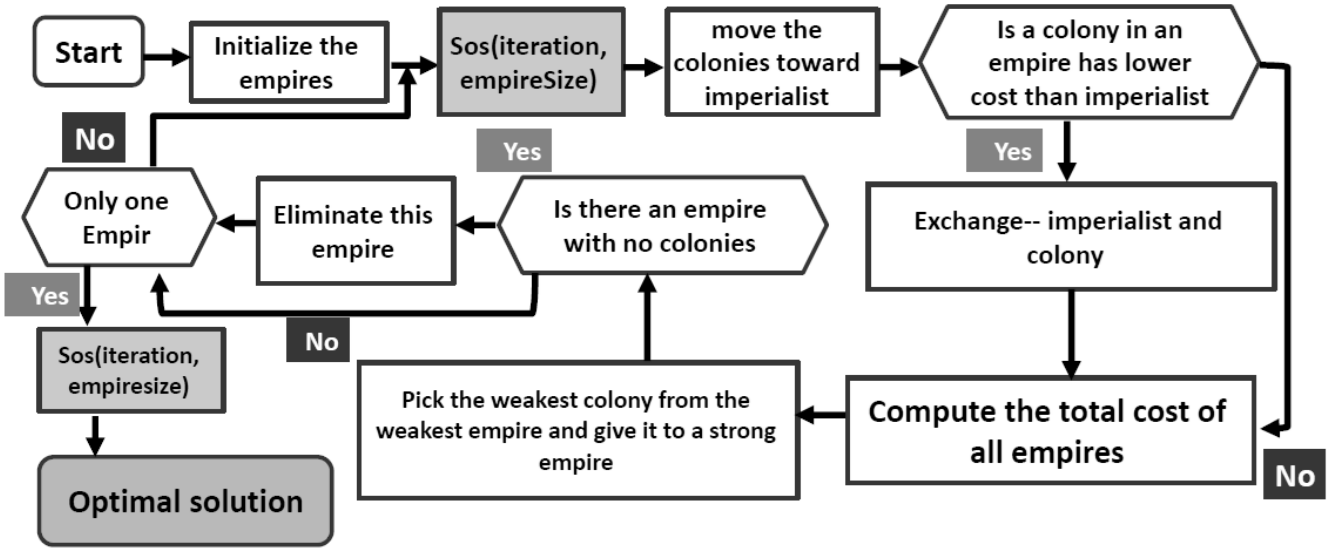
بالتالي قد لاتحتاج الإمبراطورية المحسّنة إلى عمليات تبديل بين المستعمرات والإمبريالي، وهذا يؤدي إلى توفير وقت في البحث بكل خوارزمية، يبقى هذا الوقت أكبر من الوقت الذي استهلكته خوارزمية SOS التي تم حقنها لأنها تتقارب بشكل سريع من الحل، وهنا في حالة الدمج لم نكرر SOS أكثر من ثلاث مرات –كما ستظهر النتائج– في كل عملية حقن.

كما أن الإمبراطورية الأخيرة الوحيدة التي تصل إليها خوارزمية ICA يمكن عدُّها فضاء حلول أولي لخوارزمية SOS، حيث تم حقن خوارزمية SOS بعد التوصل لإمبراطورية واحدة وقبل التوقف لاختيار الحل الأمثلي Optimal Solution بحيث يكون عدد التكرارات Iteration لخوارزمية SOS هو عدد التكرارات الذي تم تحديده عند البدء كدخل للخوارزمية المقترحة ICA-SOS، أما عدد الكائنات هو عدد الدول في الإمبراطورية الأخيرة التي بقيت بعد انتهاء عمليات المنافسة. إن الحل الأمثلي Optimal Solution هو الدولة (مجموعة الثنائيات) التي تمثل بالنسبة لخوارزمية SOS الكائن الحي الذي يمتلك أعلى درجة تكيف بالنسبة لزمن التنفيذ، بحيث يكون زمن تنفيذ هذه التشكيلة من الثنائيات أصغر ما يمكن، الشكل 2-14.



الشكل 2-14 حقن خوارزمية SOS بعد الامبراطورية الأخيرة وقبل التوقف.

إذاً يمكن صياغة خطوات الخوارزمية الجديدة ICA-SOS كما يوضح الشكل 2-15.



الشكل 2-15 مخطط الخوارزمية الهجينة ICA-SOS

دخل الخوارزمية ICA-SOS هو عدد التكرارات وعدد الإمبراطوريات وعدد الدول الكلي.

Inputs : I (Iterations) , E (Empires) , C (Countries)

يتم تلخيص الخوارزمية الهجينة ICA-SOS بالخطوات الرئيسية التالية:

1. تهيئة الإمبراطوريات بعدد E وتوزيع الدول عليها، حيث عدد الدول الكلي C.
2. تطبيق مراحل SOS على كل إمبراطورية بعدد تكرارات I أما عدد الكائنات بالنسبة لخوارزمية SOS فهو عدد الدول countries في الامبراطورية التي يتم تطبيق SOS عليها.
3. تحرّك كل مستعمرة Colony باتجاه الامبريالي Imperialist التابعة له في كل امبراطورية Empire.
4. في حالة أن الموقع الجديد للمستعمرة ذو كلفة أقل من موقع الامبريالي يتم تبديل مواقع المستعمرة والإمبريالي (Exchanging) وتصبح هذه المستعمرة امبريالي، ويصبح الامبريالي مستعمرة.
5. حساب الطاقة الكلية لكل امبراطورية.
6. اختيار أضعف مستعمرة في أضعف امبراطورية وإعطائها لامبراطورية ذات احتمالية استيلاء عالية وهنا يبدأ التنافس الإمبريالي (Imperialistic competition).
7. إهمال الإمبراطورية التي لاتمتلك مستعمرات واستبعادها من المنافسة.
8. إذا بقي امبراطورية واحدة يتم الانتقال للخطوة التالية وإلا يتم العودة للخطوة 2.

9. تطبيق مراحل SOS على الإمبراطورية الأخيرة بعدد تكرارات I أما عدد الكائنات بالنسبة لخوارزمية SOS فهو عدد الدول countries في الامبراطورية الأخيرة التي تم تطبيق SOS عليها.
10. تتوقف الخوارزمية ICA-SOS عند انتهاء عدد التكرارات I بعد تطبيق SOS على الامبراطورية الأخيرة، ويكون الحل الأمثلي هو الامبريالي في هذه الامبراطورية وهو نفسه الكائن ذو أعلى درجة تكيف بالنسبة لخوارزمية SOS أي مجموعة الثنائيات من الإسنادات التي يكون بها زمن التنفيذ الكلي Makespan أصغري.

الرماز المعبّر عن الخوارزمية ICA-SOS:

Empires: integer *// empires number: input*
Countries: integer *// countries number: input*
Iter: integer // iteration number in SOS: input

Empires_number ← Empires
Do
N_e ← countries number in an Empire
C_e ← colonies number in each Empire
For e =1 to Empires_number
For j =1 to C_e
// SOS in Empire e
//input: Size: N_e Iteration: Iter
iteration_number ← 0
xbest ←0
Do
iteration_number ←iteration_number +1
i←0
Do
i←i+1
For j =1 to N_e
if $Makespan(x_j) < Makespan(x_{best})$ *then*
$x_{best} \leftarrow x_j$
End if
End for
//mutualism phase
Randomly select x_j with $i \neq j$
F_1 ← 1 or 2 , F_2 ← 1or 2
r_1 ← rand(0,1) , r_2 ← rand(0,1)
$x_{inew} = x_i + r_1 * \left(x_{best} - \frac{x_i + x_j}{2} * F_1\right)$

$$x_{jnew} = x_j + r_2 * \left(x_{best} - \frac{X_i + X_j}{2} * F_2\right)$$

$if\ Makespan(x_{inew}) < Makespan(x_i)\ then$

$x_i \leftarrow x_{inew}$

End if

$if\ Makespan(x_{jnew}) < Makespan(x_j) then$

$x_j \leftarrow x_{jnew}$

End if

//commensalism phase

Randomly select x_j with $i \neq j$

$r_1 \leftarrow$ rand(-1,1)

$x_{inew} = x_i + r_1 * (x_{best} - x_j)$

$if\ Makespan(x_{inew}) < Makespan(x_i)\ then$

$x_i \leftarrow x_{inew}$

End if

//parasitism phase

Randomly select x_j with $i \neq j$

Create a parasite vector x_p from x_i using random number

$if\ Makespan(x_p) < Makespan(x_j)\ then$

$x_j \leftarrow x_p$

End if

While i <= N_e

//End Do While

While iteration_number <= iter

//End Do While

$\boldsymbol{imperialist_e \leftarrow x_{best}}$

Move the colonies toward their relevant imperialist (Assimilating).

$\boldsymbol{if\ Makespan(colony_j) < Makespan(\ imperialist_e)\ then}$

$\boldsymbol{imperialist_e \leftarrow colony_j}$

End if

End for // End For j =1 to C_e

End for // End For e =1 to Empires_number

Compute the total cost of each Empire

$Empire_w \leftarrow$ the weakest Empire

$Empire_s \leftarrow$ the stronger Empire

WeakestC $\leftarrow$ weakest colony in $Empire_w$

$Empire_w$_Colonies $\leftarrow$ $Empire_w$_Colonies - WeakestC

$Empire_s$_Colonies $\leftarrow$ $Empire_s$_Colonies + WeakestC

For k = 1 to Empires_number

$\boldsymbol{if\ (empire_k\ has\ no\ colonies)\ then}$

Empires_number $\leftarrow$ Empires_number -1 // delete Empire

break

End if

End for

If(Empires_number == 1) // Last Empire, Just one Empire

// SOS in Last Empire

//input: Size: Countries number in last empire : N_e , Iteration: Iter

iteration_number ← 0

xbest ←0

Do

iteration_number ←iteration_number +1

i←0

Do

i←i +1

For j =1 to N_e

$if\ Makespan(x_j) < Makespan(x_{best}) then$

$x_{best} \leftarrow x_j$

End if

End for

//mutualism phase

Randomly select x_j with $i \neq j$

F_1 ← 1 or 2 , F_2 ← 1or 2

r_1 ← rand(0,1) , r_2 ← rand(0,1)

$$x_{inew} = x_i + r_1 * \left(x_{best} - \frac{x_i + x_j}{2} * F_1\right)$$

$$x_{jnew} = x_j + r_2 * \left(x_{best} - \frac{X_i + X_j}{2} * F_2\right)$$

$$if\ Makespan(x_{inew}) < Makespan(x_i)\ then$$

$x_i \leftarrow x_{inew}$

End if

$$if\ Makespan(x_{jnew}) < Makespan(x_j) then$$

$x_j \leftarrow x_{jnew}$

End if

//commensalism phase

Randomly select x_j with $i \neq j$

r_1 ← rand(-1,1)

$$x_{inew} = x_i + r_1 * (x_{best} - x_j)$$

$$if\ Makespan(x_{inew}) < Makespan(x_i)\ then$$

$x_i \leftarrow x_{inew}$

End if

//parasitism phase

Randomly select x_j with $i \neq j$

Create a parasite vector x_p from x_i using random number

$$if\ Makespan(x_p) < Makespan(x_j)\ then$$

$x_j \leftarrow x_p$

End if

While i <= N_e

// End Do While

While iteration_number <= iter

// End Do While

End if

While Empires_number > 1 // if Empires_number ==1 => Stop

استعرضنا في هذا الفصل مراحل كل خوارزمية، وتم اقتراح طريقة دمج الخوارزميتين بخوارزمية واحدة تحافظ على تكرارات خوارزمية SOS كمحدد دخل، وهذا يعطي الخوارزمية الجديدة مرونة لمعايرة التقارب المطلوب في مراحل خوارزمية ICA.

وكذلك تم المحافظة على عملية المنافسة بين الإمبرياليين لأن هذه المنافسة هي التي تؤدي إلى زيادة طاقة الإمبراطورية عند كل عملية استيلاء، بالمحصلة تم الاستفادة من ميزة كل خوارزمية، وهذا ما ستوضحه نتائج تنفيذ الخوارزميات في فصل التطبيق العملي.

الفصل الثالث: التطبيق العملي

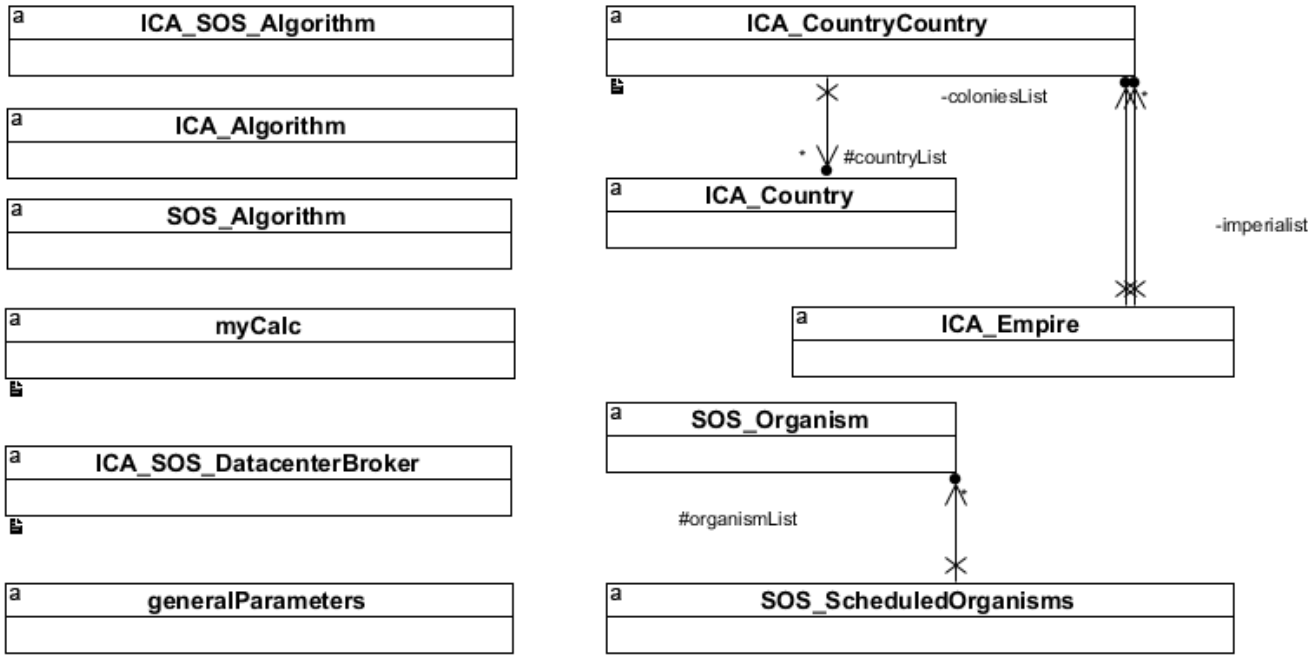
بعد استعراض الخوارزميتين ودراستهما، وشرح كيفية نمذجة كل خوارزمية، وإسقاط خطواتهما على مسألة جدولة المهام، نستعرض في هذا الفصل التطبيق العملي للشرح السابق والنتائج العملية التي توضح ميزات كل خوارزمية، وكذلك نتائج تطبيق الخوارزمية الجديدة ICA-SOS التي تم الوصول إليها بدمج الخوارزميتين.

3.1 بيئة التطوير وبناء المشروع.

تم استخدام بيئة المحاكاة cloudsim وتضمين المكتبات الخاصة بها على بيئة التطوير Netbeans IDE 8.2

تم كتابة الكود البرمجي بلغة Java version 1.8.0_251

تم تنظيم المشروع في بيئة التطوير ضمن عدة صفوف كما يوضح الشكل 3-1:

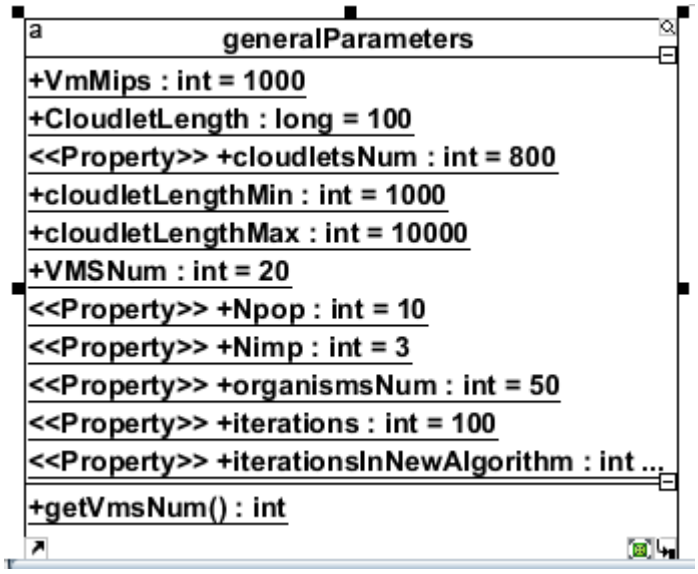


الشكل 3-1 الصفوف التي تم بناؤها في بيئة التطوير.

الصف generalParameters:

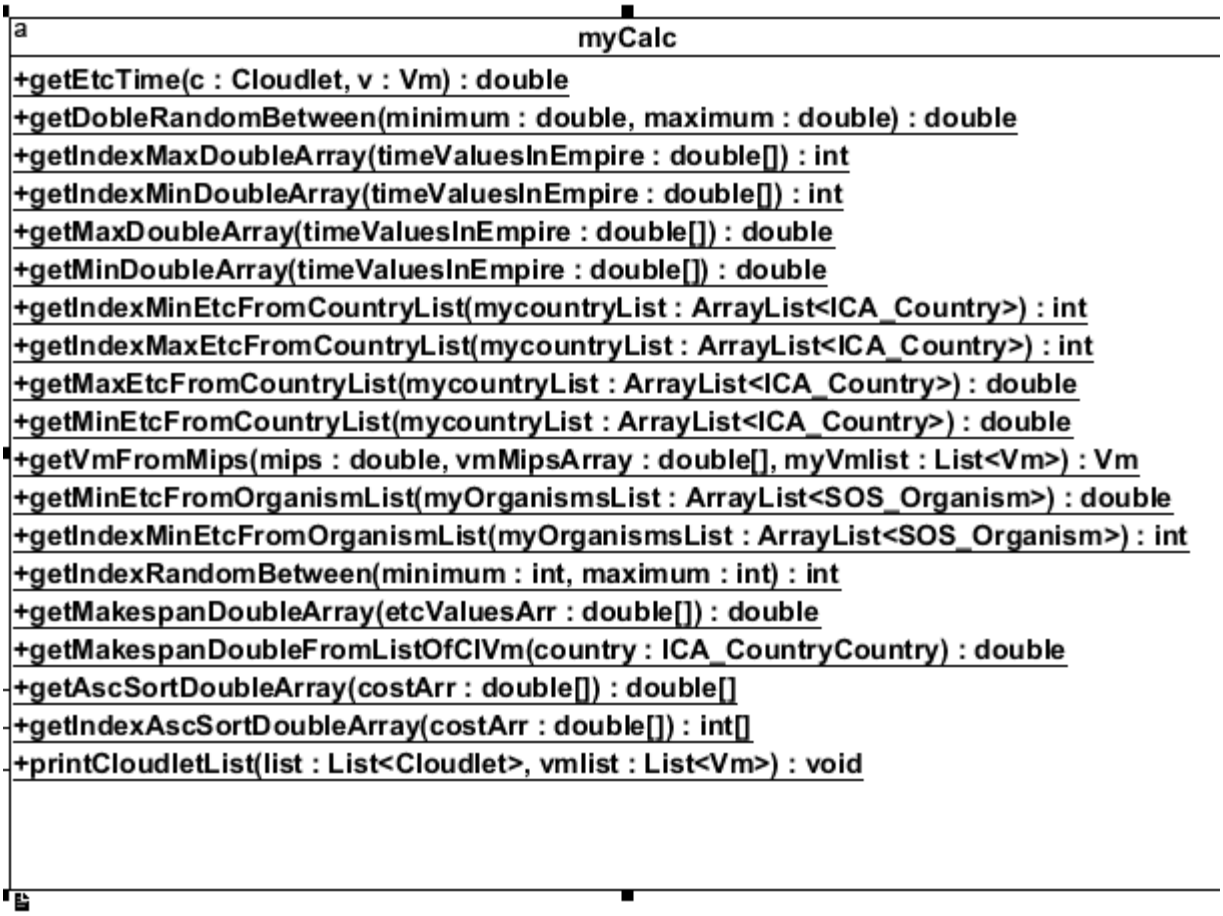
يضم هذا الصف متغيرات دخل كل خوارزمية، وتوابع لإرجاع قيمة كل متغير، الغرض من بناء هذا الصف هو سهولة اختبار الخوارزميات حيث يتم تغيير قيم المتحولات في هذا الصف فقط وتندرج على بقية الصفوف

من خلال استدعاء التوابع التي تعيد هذه المتحولات، وقد تم تعريف جميع المتحولات والتوابع في هذا الصف من النمط static من أجل الوصول إليها مباشرة في بقية الصفوف بالنقطة من دون تعريف غرض من الصف.



الصف myCalc

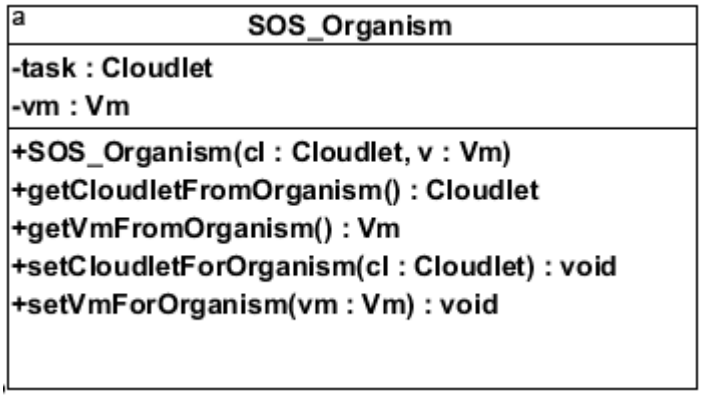
يضم هذا الصف عدة توابع يتم استخدامها في بقية الصفوف لتسهيل العمليات الحسابية ولتسهيل تنظيم كتابة الكود البرمجي ضمن كل خوارزمية، من هذه التوابع مثلاً حساب قيمة عشوائية بين قيمتين، حساب زمن التنفيذ لكائن حي أو لدولة، حساب زمن تنفيذ ضمن ثنائية، وغيرها من التوابع.



نمذجة خوارزمية SOS

تم شرح طريقة استخدام خوارزمية SOS لنمذجة مسألة الجدولة في الفصل الثاني، ولتحقيق ذلك عملياً تم بناء عدة صفوف نشرحها فيما يلي:

الصف SOS_Organism: يعبر هذا الصف عن ثنائية ضمن الكائن الحي، المسقط الأول من الثنائية هو المهمة المراد إسنادها، يتم تعريفها كمتحول عضو في الصف اسمه task من النمط Cloudlet أما المسقط الثاني من الثنائية هو الجهاز الإفتراضي الذي يتم إسناد المهمة إليه، يتم تعريفه كمتحول عضو في الصف اسمه vm من النمط Vm، تسمح بيئة المحاكاة cloudsim بتعريف كل من النمط Cloudlet و النمط Vm

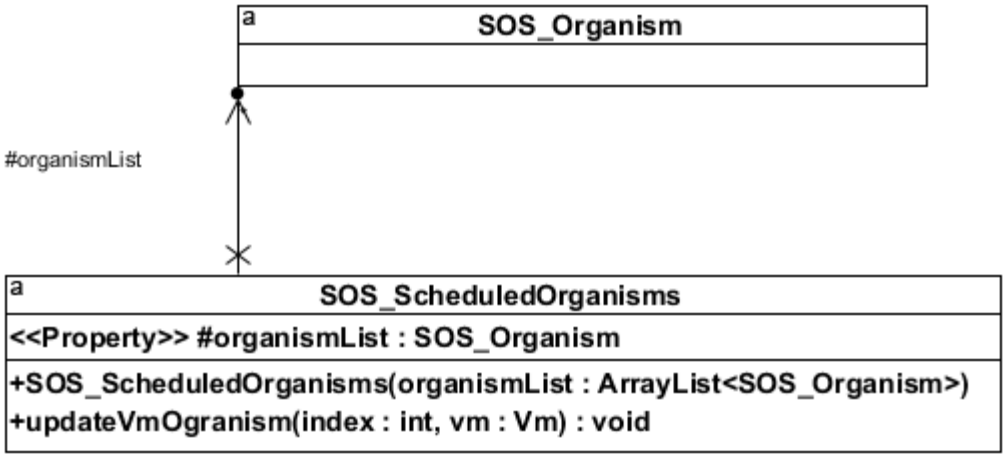


التوابع الأعضاء في الصف موضحة في الجدول 3-1:

الجدول 3-1 التوابع الأعضاء في الصف SOS_Organism

التابع	شرح
SOS_Organism(Cloudlet,Vm)	الباني، مهمته إنشاء غرض من الصف.
getCloudletFromOrganism()	إرجاع المهمة الحالية في الغرض، يرجع قيمة من النمط Cloudlet
Vm getVmFromOrganism()	إرجاع الجهاز الإفتراضي الحالي في الغرض، يرجع قيمة من النمط VM
setCloudletForOrganism(Cloudlet cl)	تعديل المهمة الحالية في الغرض إلى مهمة جديدة cl
setVmForOrganism(Vm vm)	تعديل الجهاز الإفتراضي الحالي في الغرض إلى جهاز إفتراضي جديد vm

الصف SOS_ScheduledOrganisms: إنَّ الكائن الحي Oganism الذي يتم التعبير عنه بمجموعة ثنائيات؛ أي مجموعة إسنادات كما تم توضيحه في الفصل الثاني، تم إنشاؤه في بيئة التطوير من خلال بناء الصف SOS_ScheduledOrganisms الذي يمثل كائناً حياً ضمن النظام البيئي.



يضم الصف المتحولات والتوابع الأعضاء الموضحة في الجدول 3-2:

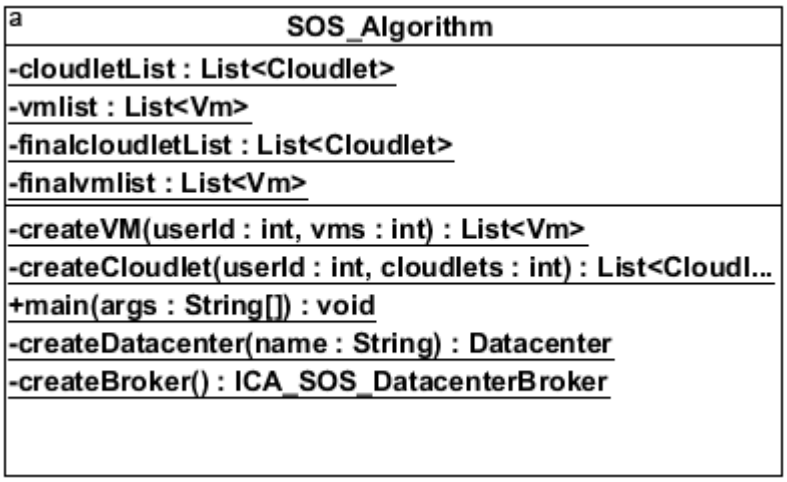
الجدول 3-2 مكونات الصف المعبر عن الكائن الحي.

العضو	شرح
ArrayList<SOS_Organism>	متحول عضو من النمط قائمة من الصف SOS_Organism يعبر عن مجموعة ثنائيات، سطر ضمن النظام البيئي، حل مرشح، كائن حي.
SOS_ScheduledOrganisms	باني لإنشاء غرض من الصف.
updateVmOgranism	تعديل جهاز إفتراضي ما في الكائن الحي، أي تعديل المسقط الثاني من إحدى الثنائيات في الكائن الحي، يتطلب إدخال دليل الثنائية المراد تعديل مسقطها الثاني، والجهاز الإفتراضي الجديد.

الصف ICA_SOS_DatacenterBroker: يضم هذا الصف عدة توابع تم تعريفها مسبقاً ضمن إطار عمل cloudsim، مهمة هذا الصف، هي استقبال المهام المراد جدولتها وإسنادها إلى الأجهزة الافتراضية المعنية، وكذلك استقبال المعلومات الخاصة بمراكز البيانات المتاحة، يقوم بعمل دور المجدول Scheduler، الحالة الإفتراضية للمجدول ضمن بيئة العمل هي جدولة المهام على مبدأ القادم أولاً يتم تخديمه أولاً FCFS (First Come First Served)، تم تعديل هذه الطريقة لتصبح الجدولة بحسب نتيجة جدولة الخوارزمية المعنية، لذلك

قمنا بإرسال المهام والأجهزة الإفتراضية التي سيتم إسناد كل مهمة إليها إلى هذا المجدول بعد توقف كل خوارزمية، ثم يقوم المجدول بإسناد كل مهمة إلى الجهاز الإفتراضي المحدد له، ثم حساب زمن الوصول وزمن الإنتهاء.

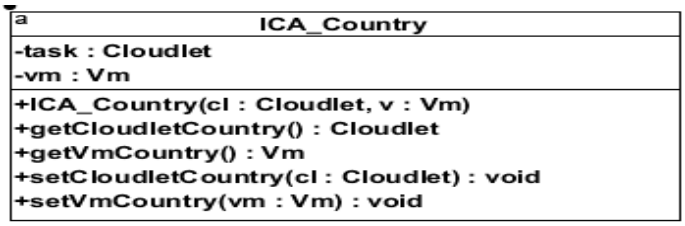
الصف SOS_Algorithn: يضم هذا الصف كتابة كامل خوارزمية SOS بالإعتماد على الصفوف السابقة، ويتم تنفيذ الخوارزمية بتنفيذ التابع main ضمن هذا الصف الذي يستدعي توابع من الصفوف السابقة، ويتم في هذا الصف تعريف مراكز البيانات والمضيف الفيزيائي وكذلك تعريف الأجهزة الافتراضبة والمهام المراد جدولتها.



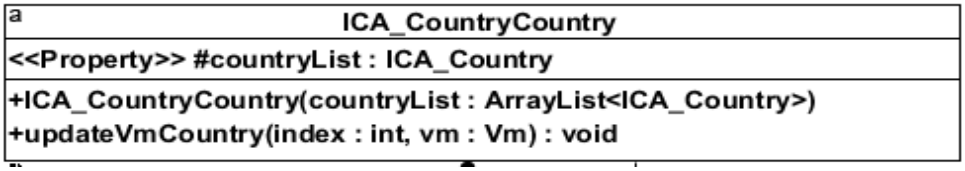
نمذجة خوارزمية ICA

تم شرح طريقة استخدام خوارزمية ICA لنمذجة مسألة الجدولة في الفصل الثاني، ولتحقيق ذلك عملياً تم بناء عدة صفوف نشرحها فيما يلي:

الصف ICA_Country: إن هذا الصف مشابه للصف sos_organism فهو يعبر عن ثنائية ضمن الدولة (التي يتم التعبير عنها بمجموعة من الثنائيات، وتمثل حلاً مرشحاً)، المسقط الأول من الثنائية هو المهمة المراد إسنادها، ويتم تعريفها بكونها متحولاً عضواً في الصف اسمه task من النمط Cloudlet أما المسقط الثاني من الثنائية هو الجهاز الإفتراضي الذي يتم إسناد المهمة إليه، يتم تعريفه بكونه متحولاً عضواً في الصف اسمه vm من النمط Vm، تسمح بيئة المحاكاة cloudsim بتعريف كل من النمط Cloudlet و النمط Vm



الصف ICA_CountryCountry: تم بناء هذا الصف ليعبر عن حل مرشح ضمن مجموعة حلول وهو الدولة country التي قد تكون إمبريالي imperialist أو مستعمرة colony كما تم شرحه في الفصل الثاني، وهي عبارة عن مجموعة من الثنائيات.

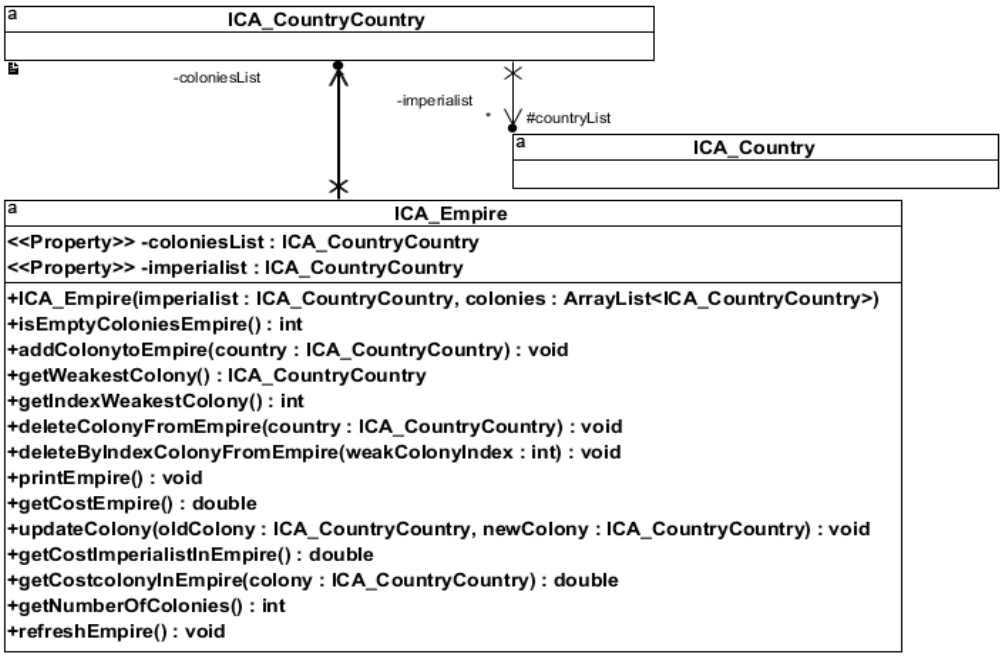


يتكون الصف ICA_CountryCountry من المتحولات والتوابع الأعضاء الموضحة في الجدول الجدول 3-3:

الجدول 3-3 مكونات الصف المعبّر عن الدولة.

العضو	شرح
ArrayList<ICA_Country>	متحول عضو من النمط قائمة من الصف ICA_Country يعبر عن مجموعة ثنائيات، سطر ضمن فضاء الحلول، حل مرشح، دولة.
ICA_CountryCountry	باني لإنشاء غرض من الصف.
updateVmCountry	تعديل جهاز إفتراضي ما في الدولة، أي تعديل المسقط الثاني من إحدى الثنائيات في الدولة(مجموعة ثنائيات)، يتطلب إدخال دليل الثنائية المراد تعديل مسقطها الثاني، والجهاز الإفتراضي الجديد.

الصف ICA_Empire: تم بناء هذا الصف ليمثل إمبراطورية في خوارزمية ICA.



يوضح الجدول 3- 4 شرح كل تابع عضو في هذا الصف.

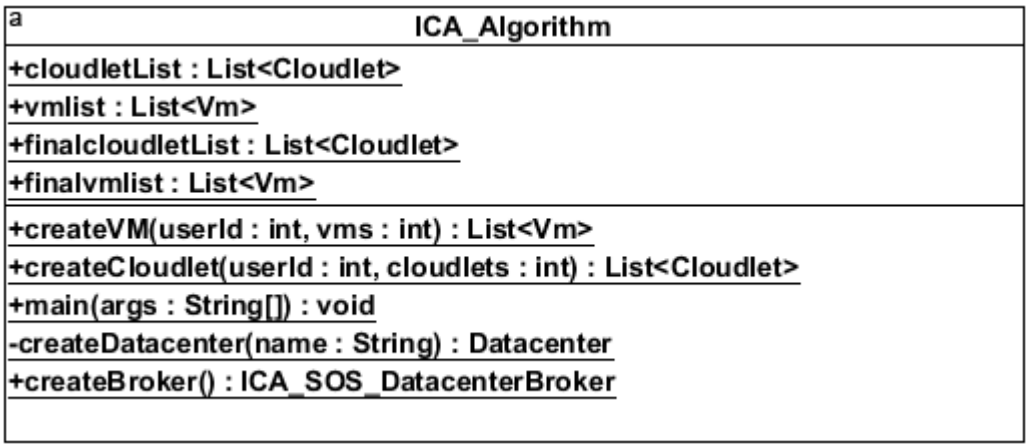
الجدول 3-4 المتحولات والتوابع الأعضاء في الصف ICA_Empire

العضو	شرح
ArrayList<ICA_CountryCountry> coloniesList	متحول عضو من النمط قائمة من الصف ICA_CountryCountry يعبر عن مجموعة دول تمثل المستعمرات التابعة للإمبراطورية، أي عدة أسطر ضمن فضاء الحلول.
ICA_CountryCountry imperialist	متحول عضو من النمط ICA_CountryCountry يعبر عن دولة تمثل الإمبريالي الوحيد ضمن الإمبراطورية.
ICA_Empire(ICA_CountryCountry imperialist, ArrayList<ICA_CountryCountry> colonies)	باني لتعريف غرض من الصف.
getColoniesList()	إرجاع قائمة بمستعمرات الإمبراطورية.

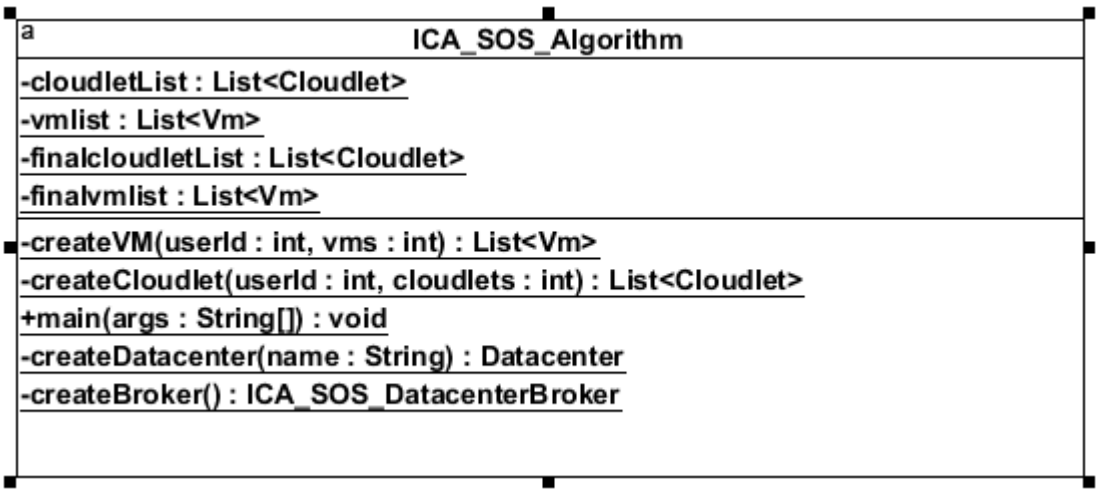
ICA_CountryCountry getImperialist()	إرجاع الإمبريالي في الإمبراطورية.
isEmptyColoniesEmpire()	يرجع القيمة 1 إذا كانت الإمبراطورية لاتحوي مستعمرات، وإلا يرجع القيمة 0.
addColonytoEmpire(ICA_CountryCountry)	يضيف مستعمرة إلى قائمة المستعمرات في الإمبراطورية.
getWeakestColony()	يحسب المستعمرة الأضعف ضمن الإمبراطورية أي التي تحقق أكبر قيمة لزمن التنفيذ Makespan ويرجع هذه المستعمرة.
getIndexWeakestColony()	يرجع دليل أضعف مستعمرة في الإمبراطورية ضمن قائمة المستعمرات.
deleteColonyFromEmpire(ICA_CountryCoutr)	يحذف مستعمرة ضمن الإمبراطورية(نحتاج هذا التابع في عملية التنافس الإمبريالي حيث يتم حذف مستعمرة من إمبراطورية ثم إضافة هذه المستعمرة لإمبراطورية أخرى).
deleteByIndexColonyFromEmpire(int)	يحذف مستعمرة بحسب دليل هذه المستعمرة ضمن قائمة مستعمرات الإمبراطورية.
printEmpire()	طباعة الإمبراطرية في الخرج (طباعة الإمبريالي والمستعمرات بشكل أسطر من الثنائيات) وطباعة كلفة الإمبراطورية وعدد مستعمراتها.
getCostEmpire()	إرجاع كلفة الإمبراطورية.
setImperialist(ICA_CountryCountry country)	تعديل الإمبريالي الموجود في الإمبراطورية بإمبريالي آخر.
updateColony(ICA_CountryCountry oldColony, ICA_CountryCountry newColony)	تعديل مستعمرة موجودة في الإمبراطورية بمستعمرة أخرى.
getCostImperialistInEmpire()	إرجاع كلفة الإمبريالي في الإمبراطورية.

getCostcolonyInEmpire(ICA_CountryCount)	إرجاع كلفة مستعمرة في الإمبراطورية.
getNumberOfColonies()	إرجاع عدد المستعمرات في الإمبراطورية.
refreshEmpire()	تحديث الإمبراطورية، حيث يتم تبديل بين مستعمرة وإمبريالي في حالة زمن التنفيذ للإمبريالي ليس الأفضل.

الصف ICA_Algorithm: يتم في هذا الصف كتابة خوارزمية ICA بشكل كامل بالإعتماد على الصفوف السابقة المعرفة للإمبراطورية والدولة والثنائية، يضم هذا الصف التابع الرئيسي main الذي يتم من خلاله تنفيذ جميع التوابع والإسنادات وطباعة النتائج.



نمذجة الخوارزمية الهجينة ICA-SOS: تم كتابة خطوات الخوارزمية الهجينة في الصف ICA_SOS_Algorithm بالإعتماد على الصفوف التي تم شرحها سابقاً، حيث تم حقن خوارزمية التكافل البيئي SOS ضمن مرحلة تقارب المستعمرة من الإمبريالي في كل إمبراطورية، وكذلك في الإمبراطورية الأخيرة.


a
ICA_SOS_Algorithm
-cloudletList : List<Cloudlet>
-vmlist : List<Vm>
-finalcloudletList : List<Cloudlet>
-finalvmlist : List<Vm>
-createVM(userId : int, vms : int) : List<Vm>
-createCloudlet(userId : int, cloudlets : int) : List<Cloudlet>
+main(args : String[]) : void
-createDatacenter(name : String) : Datacenter
-createBroker() : ICA_SOS_DatacenterBroker

3.2 النتائج العملية.

تم تهيئة بيئة المحاكاة بالقيم الموضحة في الجدول 3-5 والجدول 3-6 والجدول 3-7 قبل البدء بتجريب الخوارزميات وتسجيل النتائج:

مراكز البيانات Datacenters:

الجدول 3- 5 مواصفات مراكز البيانات في بيئة المحاكاة

عدد المضيفين في كل مركز Hosts	سرعة تنفيذ التعليمات Million Instructions Per Second MIPS
2	1000000

المضيفون الفيزيائيون Hosts:

الجدول 3-6 مواصفات كل مضيف فيزيائي في مركز البيانات

رقم تسلسلي للمضيف Host	الذاكرة RAM (GB)	التخزين Storage (TB)	التردد Bandwidth (GB/s)	طريقة جدولة الآلات الإفتراضية VM Scheduling	هرمية النظام system architecture	نظام التشغيل operating system	عدد النوى Cores
1	20	1	10	time-shared	x86	Linux	2
2	20	1	10	time-shared	x86	Linux	4

الآلات الافتراضية Virtual Machines:

الجدول 3-7 مواصفات كل آلة افتراضية في مركز البيانات

الذاكرة RAM (GB)	التردد Bandwidth (GB/s)	عدد المعالجات	سرعة تنفيذ التعليمات MIPS
0.5	1	1	من 1000 إلى 10000

تم تجريب الخوارزميات على ثلاث مسائل:

المسألة الأولى والثانية: تم تحديد أطوال المهام بشكل مسبق، وكذلك تحديد مجال عدد التعليمات المنفذة في كل آلة افتراضية MIPS، أما بقية المعايير فبقيت كما هو موضح في الجدول 3-5 والجدول 3-6 والجدول 3-7.

ثم تم تنفيذ كل خوارزمية من الخوارزميات الثلاث SOS, ICA, ICA-SOS وتسجيل النتائج ومقارنة نتائج الخوارزمية الجديدة ICA-SOS مع كل من خوارزمية SOS وخوارزمية ICA.

المسألة الثالثة: تم توليد أطوال المهام بشكل عشوائي بين القيمتين 1000 و 10000 وجدولتها على آلات افتراضية عددها 20 والالتزام بالمعايير المذكورة في الجدول 3-5 والجدول 3-6 والجدول 3-7 وهي المعايير نفسها التي تم اعتمادها في الورقة البحثية[2] لتنفيذ خوارزمية SOS لجدولة المهام، ثم مقارنة نتائج الخوارزمية ICA-SOS مع نتائج تنفيذ SOS في الورقة البحثية[2]

المسألة الأولى: جدولة 1000 مهمة على 125 جهاز افتراضي

المهام التي سيتم جدولتها ليست متساوية جميعها بالطول، كما يوضح الجدول 3-8:

الجدول 3-8 الأرقام المعرفة للمهام وأطوال المهام.

Cloudlet_ID	0	1	2	3	4	5	...	24	25	26	...	999
Length	1000	1100	1200	1300	1400	1500	...	3400	1000	1100	...	3400

وكذلك الأجهزة الإفتراضية VMs ليست متساوية جميعها بعدد التعليمات التي تنفذها بالثانية (مضروباً بمليون) كما يوضح الجدول 3-9:

الجدول 3-9 الأرقام المعرفة للأجهزة الإفتراضية وعدد التعليمات.

VM_ID	0	1	2	3	4	5	6	7	8	9	10	11	...	124
MIPS	50	150	250	350	450	550	650	750	850	950	50	150	...	450

الحل باستخدام خوارزمية SOS دخل الخوارزمية: عدد الكائنات Organisms وعدد التكرارات Iterations

يوضح الجدول 3-10 نتائج تنفيذ الخوارزمية.

الجدول 3- 10نتائج الخوارزمية SOS لحل المسألة الأولى

Organisms (input)	Iterations (input)	Makespan (ms)
10	1	5.393
10	3	3.873
10	5	3.556
10	7	3.063
10	10	3.034
10	12	3.205
10	15	2.589
10	17	2.530
10	20	2.669
10	22	2.692
10	25	2.914
10	40	2.800
15	3	3.616
15	5	3.152
15	7	3.159
15	10	2.919
15	12	2.962
15	15	2.520
15	17	2.632
15	40	2.823
50	10	2.671
500	16	2.523

بملاحظة النتائج السابقة نجد أن خوارزمية SOS استطاعت الوصول إلى حل أمثلي لجدولة 1000 مهمة على 125 جهازاً افتراضياً بزمن تنفيذ يقارب ال 2.5 ms بعدد تكرارات لم يتجاوز ال 15 تكراراً، ولم يتم الوصول لحل أفضل وإنْ زاد عدد التكرارات، وذلك بعدد كائنات مختلف في كل مرة، تم التجريب على 10 و 15 و 50 و500 كائن وتم الوصول للحل الأمثلي بالكفاءة نفسها عند عدد التكرارات نفسها.

الحل باستخدام خوارزمية ICA دخل الخوارزمية: عدد الدول Countries وعدد الإمبراطوريات Empires

يوضح الجدول 3-11 نتائج تنفيذ الخوارزمية.

الجدول 3-11 نتائج الخوارزمية ICA لحل المسألة الأولى

Countries (input)	Empires (input)	Competitive count (عدد مرات الاستيلاء على مستعمرة- حجم المنافسة)	Makespan (ms)
10	3	151	4.259
10	3	610	3.989
10	3	623	4.014
10	3	141	3.955
10	3	43	5.501
10	3	721	3.116
10	4	149	4.356
10	4	111	5.491
10	4	127	4.979
10	4	163	5.311
10	4	174	4.080
10	5	54	4.899
10	5	9	6.637
10	5	115	4.972
10	5	79	6.188
10	5	43	5.788

بملاحظة النتائج السابقة نجد أن خوارزمية ICA استطاعت الوصول إلى حل أمثلي لجدولة 1000 مهمة على 125 جهازاً افتراضياً بزمن تنفيذ يقارب ال 3.1 ms وهو أفضل حل تمكنت من الوصول إليه بتغيير عدد الإمبراطوريات ومهما زادت المنافسة وعمليات الإستيلاء.

نلاحظ أن الحل الذي توصلت إليه خوارزمية SOS أفضل من الحل الذي توصلت إليه خوارزمية ICA من أجل حل مسألة الجدولة نفسها.

الحل باستخدام الخوارزمية الهجينة ICA-SOS

دخل الخوارزمية: عدد الدول Countries وعدد الإمبراطوريات Empires وعدد التكرارات Iterations لخوارزمية SOS ضمن مرحلة تقارب المستعمرة وفي الإمبراطورية الأخيرة.

يوضح الجدول 3-12 نتائج تنفيذ الخوارزمية.

الجدول 3-12 نتائج الخوارزمية ICA-SOS لحل المسألة الأولى

Countries (input)	Empires (input)	Iterations (input)	Competitive count (عدد مرات الاستيلاء على مستعمرة- حجم المنافسة)	Makespan (ms)
10	3	1	11	3.752
10	3	3	329	2.316
10	3	5	545	2.316
10	4	1	81	2.583
10	4	3	13	3.884
10	4	3	22	2.981
10	4	3	8	2.524
10	4	5	19	2.559
15	4	2	4034	2.316

بملاحظة نتائج الخوارزمية الجديدة نجد أنها استطاعت الوصول لحل أمثلي يقارب ال 2.316 ms لجدولة 1000 مهمة على 125 جهازاً افتراضياً ولم تصل إلى حل أفضل منه مع تغيير عدد التكرارات وعدد الإمبراطوريات وعدد المدن، كما نلاحظ أن الخوارزمية وصلت لهذا الحل بعدد تكرارات لم يتجاوز ثلاثة تكرارات فقط في كل مرة تم حقن خوارزمية SOS ضمن مرحلة من مراحل خوارزمية ICA

الحل الذي وصلت إليه الخوارزمية الجديدة أفضل من الحل الذي وصلت إليه كل من الخوارزميتين على حدة.

يمكن تجميع النتائج السابقة لحساب نسب التحسين في كل خوارزمية كما يوضح الجدول 3-13:

الجدول 3-13 نسب التحسين في كل خوارزمية للمسألة الأولى

Sos_Makespane Organisms = 10 Iterations = 15 (ms)	ICA_Makespane Empires=3,Countries=10 (ms)	ICA_SOS_Makespane Empires=3,Countries=10, Iterations=3	SOS Optimization	ICA Optimization
2.589	3.116	2.316	10.54 %	25.67 %

الخوارزمية الهجينة ICA_SOS استطاعت التفوق على خوارزمية SOS بالوصول إلى حل أمثلي أفضل بنسبة 10.54 % وكذلك أفضل من خوارزمية ICA بنسبة 25.67 %

المسألة الثانية: جدولة 5000 مهمة على 200 جهاز افتراضي

المهام المراد جدولتها والأجهزة الافتراضية تمتلك المواصفات نفسها في المسألة الأولى، الجدول 3-8 و الجدول 3-9.

<u>الحل باستخدام خوارزمية SOS</u>

دخل الخوارزمية: عدد الكائنات Organisms وعدد التكرارات Iterations

يوضح الجدول 3-14 نتائج تنفيذ الخوارزمية.

الجدول 3-14 نتائج الخوارزمية SOS لحل المسألة الثانية

Organisms (input)	Iterations (input)	Makespan (ms)
10	3	20.579
10	5	17.056
10	10	14.703
10	13	14.542
10	15	14.122
10	20	14.542
20	3	20.214

<u>الحل باستخدام خوارزمية ICA</u>

دخل الخوارزمية: عدد الدول Countries وعدد الإمبراطوريات Empires

يوضح الجدول 3-15 نتائج تنفيذ الخوارزمية.

الجدول 3-15 نتائج الخوارزمية ICA لحل المسألة الثانية

Countries (input)	Empires (input)	Competitive count (عدد مرات الاستيلاء على مستعمرة- حجم المنافسة)	Makespan (ms)
10	3	223	21.577
10	3	82	20.469
15	4	363	16.814

الحل باستخدام الخوارزمية الهجينة ICA-SOS

دخل الخوارزمية: عدد الدول Countries وعدد الإمبراطوريات Empires وعدد التكرارات Iterations لخوارزمية SOSضمن مرحلة تقارب المستعمرة وفي الإمبراطورية الأخيرة.

يوضح الجدول 3-16 نتائج تنفيذ الخوارزمية.

الجدول 3-16 نتائج الخوارزمية ICA-SOS لحل المسألة الثانية

Countries (input)	Empires (input)	Iterations (input)	Competitive count (عدد مرات الاستيلاء على مستعمرة– حجم المنافسة)	Makespan (ms)
10	3	3	222	12.765
15	4	3	309	12.622

نسب التحسين:

الجدول 3- 17نسب التحسين في كل خوارزمية للمسألة الثانية.

Sos_Makespane Organisms = 10 Iterations = 15 (ms)	ICA_Makespane Empires=4,Countries=15 (ms)	ICA_SOS_Makespane Empires=4,Countries=15, Iterations=3	SOS Optimization	ICA Optimization
14.122	16.814	12.622	10.62 %	24.93 %

المسألة الثالثة: جدولة عدد من المهام (تبدأ ب 100 وتصل إلى 1000 مهمة) على 20 جهازاً افتراضياً

تم توليد أطوال المهام بشكل عشوائي بين القيمتين 1000 و 10000 ثم اعتماد هذه العينة من أطوال المهام واختبار الخوارزميات عليها.

الحل باستخدام خوارزمية SOS

دخل الخوارزمية: عدد الكائنات Organisms وعدد التكرارات Iterations

تم تثبيت عدد الكائنات 100 في الورقة البحثية [2] وتثبيت عدد التكرارات 1000 ثم تغيير عدد المهام في كل مرة، ولقد تم التوصل للنتائج المبينة في الشكل 3-2

Task size	SAPSO			DSOS	
	Best	Worst	Avg	Best	Worst
100	42.85	56.71	50.74	51.66	60.21
200	85.64	118.67	106.82	102.07	122.75
300	154.58	205.79	175.48	156.74	180.00
400	235.96	283.93	253.93	209.56	243.20
500	300.42	384.22	338.74	286.86	334.59
600	413.89	482.06	450.72	367.46	407.87
700	510.38	638.15	570.55	427.65	488.48
800	583.46	693.98	654.16	492.08	577.81
900	642.47	827.08	755.51	554.29	655.03
1000	806.24	940.52	883.69	646.60	733.27

الشكل 3-2 نتائج خوارزمية الكائنات المتكافلة مقارنة مع خوارزمية SAPSO

الحل باستخدام خوارزمية ICA

تم تطبيق المعايير المستخدمة في الورقة البحثية [2] وتنفيذ الخوارزمية ICA التي تم نمذجتها في هذا البحث، وتم التوصل للنتائج الموضحة في الجدول 3-18.

الجدول 3- 18نتائج خوارزمية ICA لحل المسألة الثالثة

Countries (input)	Empires (input)	Cloudlets	Makespan ($s * 10^{-6}$)	Competitive count (عدد مرات الاستيلاء على مستعمرة- حجم المنافسة)
10	3	100	67.22	969
10	3	200	138.60	854
10	3	300	215.89	429
10	3	400	254.64	781
10	3	500	377.58	463
10	3	600	515.92	455
10	3	700	626.14	170
10	3	700	488.50	467
10	3	700	445.07	587
10	3	700	583.18	316
10	3	800	559.76	197
10	3	800	594.90	133
10	3	800	682.43	117
10	3	900	570.01	321
10	3	900	552.41	400
10	3	1000	703.40	569
10	3	1000	768.03	172
10	3	1000	705.57	328
10	3	1000	641.07	1310

الحل باستخدام خوارزمية ICA-SOS

باستخدام المعايير نفسها توصلنا للنتائج التالية:

الجدول 3-19 نتائج خوارزمية ICA-SOS لحل المسألة الثالثة

Countries (input)	Empires (input)	Iterations (input)	Cloudlets	Makespan ($s * 10^{-6}$)	Competitive count (عدد مرات الاستيلاء على مستعمرة– حجم المنافسة)
10	3	3	100	48.73	114
10	3	3	200	99.96	47
10	3	3	300	153.70	32
10	3	3	400	209.94	18
10	3	3	500	268.67	60
10	3	3	600	329.91	78
10	3	3	700	370.82	86
10	3	3	800	413.62	41
10	3	3	900	458.28	130
10	3	3	1000	504.83	182

نسب التحسين في خوارزمية ICA.

الجدول 3- 20نسب التحسين في خوارزمية ICA للمسألة الثالثة

Cloudlets	ICA Makespan ($s * 10^{-6}$)	Competitive count	ICA-SOS Makespan ($s * 10^{-6}$)	ICA Optimization
100	67.22	969	48.73	27.51%
200	138.60	854	99.96	27.88 %
300	215.89	429	153.70	28.81 %
400	254.64	781	209.94	17.55 %
500	377.58	463	268.67	28.84 %
600	515.92	455	329.91	36.05 %
700	445.07	587	370.82	16.68 %
800	594.90	133	413.62	30.47 %
900	552.41	400	458.28	17.04 %
1000	641.07	1310	504.83	21.25 %
				Average = 25.21 %

نلاحظ من الجدول 3-20 أن خوارزمية ICA قد تم تحسينها بنسبة 25% حيث تم تسجيل أفضل نتيجة توصلت إليها خوارزمية ICA عند كل عدد معين من المهام كما هو موضح في الجدول 3-18 ثم مقارنة النتيجة مع نتيجة الخوارزمية الهجينة، وتسجيل النسب في الجدول 3-20.

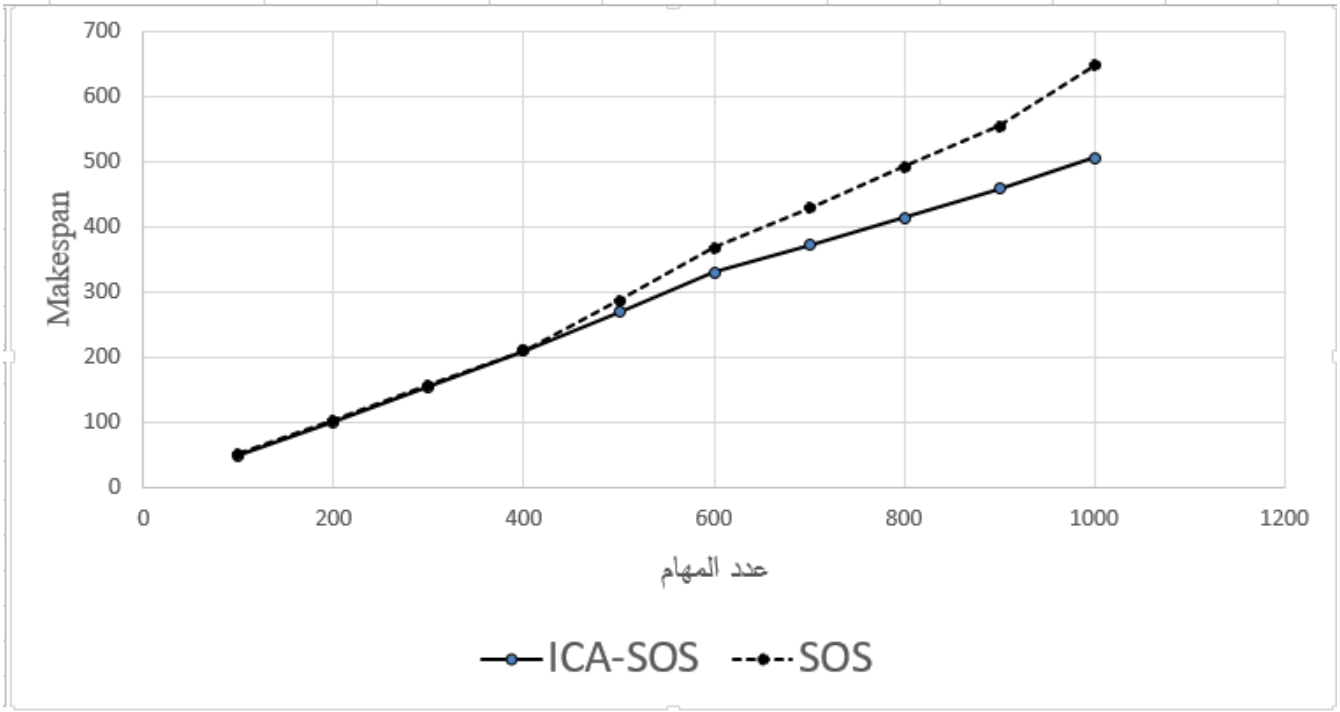
نسب التحسين في خوارزمية SOS مقارنة بالورقة البحثية[2]

يحتوي الجدول 3-21 النتائج الموضحة في الشكل 3-2 والنتائج التي توصلت إليها الخوارزمية ICA-SOS ونسب التحسين.

الجدول 3- 21مقارنة نتائج ICA-SOS مع نتائج SOS

Cloudlets	SOS Makespan ($s * 10^{-6}$)	ICA-SOS Makespan ($s * 10^{-6}$)	SOS Optimization
100	51.66	48.73	5.67 %
200	102.07	99.96	2.07 %
300	156.74	153.70	1.94 %
400	209.56	209.94	-0.18 %
500	286.86	268.67	6.34 %
600	367.46	329.91	10.22 %
700	427.65	370.82	13.29 %
800	492.08	413.62	15.94 %
900	554.29	458.28	17.32 %
1000	646.6	504.83	21.93 %
			Average = 9.45 %

يوضح الشكل 3-3 تحسن زمن تنفيذ المهام Makespan في الخوارزمية الهجينة ICA-SOS مقارنة مع خوارزمية SOS في فضاء متقطع في الورقة البحثية[2]



الشكل 3-3 تحسن زمن التنفيذ Makespan مع ازدياد عدد المهام في خوارزمية ICA-SOS

نلاحظ من الشكل 3-3 أن خوارزمية ICA-SOS تحسّن من أداء خوارزمية SOS بالنسبة لزمن تنفيذ المهام وذلك مع ازدياد عدد المهام.

النتائج والآفاق المستقبلية

تم في هذا البحث اقتراح خوارزمية جدولة جديدة ونمذجتها بالإعتماد على خوارزميتين من خوارزميات الأمثَلة العددية فائقة الاستدلال Metahuristic هما خوارزمية البحث في الكائنات المتكافلة SOS و خوارزمية التنافس بين المستعمرين ICA حيث تم تنجيز كل من الخوارزميتين في بيئة المحاكاة cloudsim وباستخدام لغة java ثم اختُبرَت الخوارزميتان على مسألة جدولة مجموعة من المهام في بيئة سحابية مكونة من مجموعة من الآلات الافتراضية بحيث يكون زمن التنفيذ أمثلياً.

أعطت كل خوارزمية من الخوارزميتين (ICA,SOS) على حدة حلاً أمثلياً، إلا أن الخوارزمية المقترحة أعطت حلاً أمثلياً أفضل من كلتا الخوارزميتين، حيث قامت بتحسين الحل بنسبة 10% مقارنة بخوارزمية SOS، وبنسبة 25% مقارنة بخوارزمية ICA.

نقترح كأعمال مستقبلية استخدام هذه الخوارزمية في البيئة السحابية لتحسين موازنة الحمل أو تحسين إحدى الخوارزميتين ودمجها مع الأخرى، ثم اختبار الخوارزمية الناتجة في أمثَلة زمن التنفيذ أو موازنة الحمل.

المراجع

[1] Cheng, M. Y., & Prayogo, D. (2014). Symbiotic organisms search: a new metaheuristic optimization algorithm. Computers & Structures, 139, 98-112.

[2] Abdullahi, M., & Ngadi, M. A. (2016). Symbiotic organism search optimization based task scheduling in cloud computing environment. Future Generation Computer Systems, 56, 640-650.

[3] Arshad, R., & Rafeh, R. (2015, November). Deadline-constrained workflow scheduling using imperialist competitive algorithm on infrastructure as a service clouds. In 2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI) (pp. 835-842). IEEE.

[4] Atashpaz-Gargari, E., & Lucas, C. (2007, September). Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In 2007 IEEE congress on evolutionary computation (pp. 4661-4667). Ieee.

[5] Habibi, M., & Navimipour, N. J. (2016). Multi-objective task scheduling in cloud computing using an imperialist competitive algorithm. IJACSA) International Journal of Advanced Computer Science and Applications, 7(5), 289-293.

[6] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience, 41(1), 23-50.

[7] Li, Y. (2019). ACO-SOS-based task scheduling in cloud computing. International Journal of Performability Engineering, 15(9), 2534.

[8] Li, Y., & Yao, Y. (2019). Scheduling Algorithm for a Task under Cloud Computing. International Journal of Performability Engineering, 15(8), 2081.

[9] "Main Categories of Optimization Algorithms." 1/5/2022. https://www.al-roomi.org/.

[10] "Optimization."1/5/2022. https://www.al-roomi.org/.

[11] Patidar, S., Rane, D., & Jain, P. (2012, January). A survey paper on cloud computing. In 2012 second international conference on advanced computing & communication technologies (pp. 394-398). IEEE.

[12] Sharma, M., & Verma, A. (2017, February). Energy-aware discrete symbiotic organism search optimization algorithm for task scheduling in a cloud environment. In 2017 4th International Conference on Signal Processing and Integrated Networks (SPIN) (pp. 513-518). IEEE.

[13] Miao, F., Zhou, Y., & Luo, Q. (2019). Complex-valued encoding symbiotic organisms search algorithm for global optimization. Knowledge and Information Systems, 58(1), 209-248.
[14] Arshad, R., & Rafeh, R. (2015, November). Deadline-constrained workflow scheduling using imperialist competitive algorithm on infrastructure as a service clouds. In 2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI) (pp. 835-842). IEEE.
[15] Kashikolaei, S. M. G., Hosseinabadi, A. A. R., Saemi, B., Shareh, M. B., Sangaiah, A. K., & Bian, G. B. (2020). An enhancement of task scheduling in cloud computing based on imperialist competitive algorithm and firefly algorithm. The Journal of Supercomputing, 76(8), 6302-6329.
[16] Jennrich, R. I., & Robinson, S. M. (1969). A Newton-Raphson algorithm for maximum likelihood factor analysis. Psychometrika, 34(1), 111-123.
[17] Fliege, J., & Svaiter, B. F. (2000). Steepest descent methods for multicriteria optimization. Mathematical methods of operations research, 51(3), 479-494.
[18] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.