2024 International Conference on Intelligent and Innovative Computing Applications (ICONIC)
7th-9th November 2024, Pearle Beach Resort & Spa, Flic en Flac, Mauritius
https://doi.org/10.59200/ICONIC.2024.023

A Hybrid Agent-Oriented Stochastic Diffusion Search and Beam Search Architecture

Sithembiso Dyubele and Duncan Anthony Coulter
University of Johannesburg
Johannesburg, South Africa
ctheradyubele@gmail.com, dcoulter@uj.ac.za

Abstract
Various swarm intelligence-based algorithms have been developed and explored over the years. These algorithms include particle swarm optimisation, spider monkey optimisation, artificial bee colony algorithm, ant colony optimisation, and bacterial foraging optimisation, among many others. However, according to the reviewed literature, classical or traditional optimisation methods are confronted with difficulties when scaling up to real-world optimisation problems; therefore, there is a need to develop efficient and robust computational algorithms that can solve problems numerically, irrespective of their sizes. Inspired by nature-inspired swarm intelligence algorithms, this study has created a hybrid-based algorithm utilising Stochastic Diffusion Search (SDS) and Beam Search algorithms. In this model, SDS is incorporated because of its ability to operate as a multi-agent population-based global search and optimisation algorithm. The Beam Agent (BA) is utilised to initialise, update, and maintain a list of candidate regions in the search space. In addition, it is responsible for recruiting agents for those regions in the search space. A variation of the knapsack problem was employed to test the created hybrid model. In this problem, constraints were established, as discussed later in the paper (in section 3.5). The results discussed in section 4 indicated that the algorithm found a better solution in the search space. The results also showed a strong and consistent beam after a series of iterations during the simulation. The specific improvements observed with the hybrid algorithm are that, because it is implemented as an actor-oriented system, it is completely parallelised, every actor is independent of every other actor and can be run automatically, on its own individual green thread but can, in fact, be run on another computer. This parallelisability, composability, and the resulting distributable nature of the new algorithm are the main advantages over the standard implementation of either stochastic diffusion search or beam search, neither of which are parallelised by default. Its implication is based on the fact that the pace of improvement in available computing power has levelled off following several decades of sustained growth characterised by Moore's law. The hybrid algorithm is highly parallelisable, making it easy to take advantage of multiple cores on a single computer or multiple machine instances in a cloud computing scenario. Therefore, this is a modern version of both component algorithms within the proposed hybrid approach as it translates better into environments where it is easier to scale outwards rather than upwards.

Keywords: Swarm Intelligence, Stochastic Diffusion Search, and Beam Agent

1. Introduction
Osaba, Del Ser, Jubeto, Iglesias, Fister, Gálvez, & Fister (2020) revealed that Stochastic Diffusion Search (SDS) was established in 1989 as a population-based pattern-matching method. Similarly, Martin, Bishop, Robinson, & Myatt (2019) claim that SDS is a novel probabilistic technique utilised to recognise and match the best fit. Kamaraj, Lanitha, Karthic, Prakash, and Mahaveerakannan (2023) indicated that SDS with the mathematical foundation is used to describe the algorithm behaviour by linear time complexity, convergence criteria to their minimum, stoutness, convergence at global optimum level, and resource assignment. Martin et al. (2019) alluded that agents collectively construct the solution by performing independent searches followed by the diffusion of information through the population. According to Maroufpoor, Azadnia, & Bozorg-Haddad (2020) and Osaba et al. (2020), SDS can be utilised to address optimisation-related problems even where

the objective can be divided into different components that can be examined separately. Al-Rifaie & Bishop (2020) indicated that various sectors have been leveraging the swarm intelligence-based algorithm known as Stochastic Diffusion Search (SDS) to solve many issues, including optimisation-related problems. Tair, Bacanin, Zivkovic, & Venkatachalam (2022) described optimisation as a process which aims to discover a solution that is either optimum or near-to-optimal based on the stated goals and given constraints. Researchers have shown interest in developing a robust and simpler architecture of nature-inspired swarm intelligence algorithms, especially in complex search spaces, to solve optimisation-related problems (Rifaie & Bishop, 2020). This study created a hybrid actor-oriented algorithm utilising both the Stochastic Diffusion Search (SDS) and Beam Search algorithms. The model accelerates the exploration property of the search space by the agents. All agents report their data to the controller agent. The controller agent evaluates all the results (statistics received from the agents while exploring the search environment), analyses them, and decides which cluster has the best quality.

1.1 The motivation behind the use of Stochastic Diffusion Search (SDS) and Beam Search algorithms

Stochastic Diffusion Search (SDS) is inherently agent oriented. While it has been previously implemented as an actor system, it is certainly not well explored. Its combination with Beam Search is accomplished through a local search using an ordered list of candidate regions to explore. Similarly, SDS has a pool of inactive agents that need to be recruited to potentially profitable regions of the search space. In the standard algorithm, they are recruited randomly with no knowledge of the topology of the search space. The proposed hybrid algorithm brings in knowledge of potentially good regions of the search space and stores it in the Beam data structure. The goal is to make search space exploration more effective through a more robust recruitment mechanism and to make the architecture of such systems more modular, composable, and distributable.

1.2 Statement of the problem

This study looks at the problem statement from a modular distributed AI point of view. It is based on the fact that modern AI systems have to be highly distributed, which taps into the advantages of cloud computing. However, many algorithms have been developed over the years without an eye towards being composable, modular, distributable systems; therefore, this study is motivated to try and find ways of creating such distributable and composable AI algorithms.

1.3 The research gap

The gap this current study seeks to address is the non-distributable nature of the normally used algorithm. No previous work has been identified combining Beam Search and Stochastic Diffusion Search as an actor system from the existing literature. The novelty of this study is not only based on the parallelism mentioned previously but also on intrinsic similarities between SDS and Beam Search. Both algorithms manage the exploration recruitment within the search space, with Beam Search providing a more nuanced mechanism for doing so for SDS.

2. Literature Review

Noisy environments and incomplete data are often at the heart of complex, real-world search and optimisation-related problems, generating input that established search heuristics sometimes have difficulty dealing with (Rifaie & Bishop, 2020). Similarly, Tair et al. (2022) revealed that since the turn of the century, there have been optimisation issues in both combinatorial and global optimisation. Over the years, many models have been developed to address these challenges with algorithms such as population-based stochastic metaheuristics. Tair et al. (2022) further indicated optimisation issues have been observed in many machine learning models. Goel, Neog, Aman, & Kaur (2020) revealed that many algorithms have been developed to address these problems; however, there is a need to develop more and better models to improve the results. Some meta-heuristic global optimisation algorithms include the harmony search algorithm, Stochastic Diffusion Search (SDS), Particle Swarm Optimisation (PSO), etc.

2.1 Stochastic Diffusion Search

Suganya and Rajan (2021) stated that agents in the population contest for direct communication patterns like cooperative transport, which is present in social insects when performing evaluations of search space. However, according to Maroufpoor et al. (2020) and Martin (2021), for SDS, the hypothesis regarding potential solutions concerning the agent population is partially to offer feedback that ensures the convergence of agents on a promising solution. Similarly, Osaba et al. (2020) conducted a study that gravitates on a heterogeneous Swarm Robotics system that utilise SDS as the coordination heuristics for the exploration, location and delimitation of areas scattered over the area where robots are deployed. Likewise, the reviewed literature revealed that a gradient boost-modified classifier with Particle Swarm Optimisation (PSO) and Stochastic Diffusion Search method for data optimisation in wireless sensor networks could be developed, with the results demonstrating the model's efficiency for data optimisation and improved network performance. However, despite the upsurge of literature on using Stochastic Diffusion Search for various purposes, to the best of the authors' knowledge, a hybrid-based model utilising Stochastic Diffusion Search (SDS) and Beam Search algorithms has not been developed.

2.2 Beam Search Algorithm

Lemons, López, Holte, & Ruml (2022) described Beam search as a well-known satisficing method for heuristic search problems that enables an individual to trade increased computation time for lower solution cost by increasing the beam width parameter. Similarly, Meister, Vieira, & Cottrell (2021) revealed that beam search is a common heuristic algorithm for decoding structured predictors. Despite providing no formal guarantee of finding the highest-scoring hypothesis under the model, beam search yields impressive performance on various tasks (Stahlberg and Byrne, 2019). Beam search generates the sequence tokens individually while keeping a fixed number of active candidates (beam size) at each step (Cohen & Beck, 2019). However, Meister et al. (2020) indicated that little work has been done to speed up beam search operation. Equally, Stahlberg & Byrne (2019) claim that in most cases, beam search fails to find the optimal output sequence. Beam search also chooses items based purely on individual scores, with no means for encoding candidate interaction (Choo, Kwon, Kim, Jae, Hottung, Tierney, & Gwon, 2022).

2.3 Detailed comparisons between SDS and other swarm intelligence algorithms explain why SDS was chosen as a base for the hybrid model.

Stochastic Diffusion Search differs from other Swarm algorithms because of its one-to-one direct recruitment mechanism, unlike Particle Swarm Optimisation, where a neighborhood function defines a set of neighboring search agents. Each neighbor contributes incrementally to the resultant vector that moves each particle. Its application is similar to Beam Search in that it keeps track of what is happening in other agents and the regions that they are exploring. However, one of the advantages is that Beam Search is able to see more of the search context, whereas in Particle Swarm Optimisation only sees what each local social group sees. The beam acts in a similar role to the neighborhoods function. However, the difference is that neighborhoods are stable in Particle Swarm Optimisation over the lifetime of the algorithm. In contrast, the information exchange moderated by the Beam Search varies as the search progresses. The beam is not tied to a particular set of actors as it distributes the information as needed.

3. Methodology

3.1 Resources

This hybrid model was developed using Python. Johansson, Johansson, & John (2019) described Python as one of the programming languages of many. Johansson et al. (2019) further indicated that it is a powerful, elegant programming language that is easy to understand and read. Lee (2019) claims that Python makes machine learning easy for beginners and experienced developers and has powerful computing capabilities that are useful in modern-day projects like computer and data science. The above-mentioned Python capabilities have convinced the authors to adopt Python as a programming tool for this hybrid classifier. All the libraries imported to perform this operation are indicated in Table (3).

3.1.1 Pseudocode for the algorithm and in-depth analysis

❖ **Importing Libraries**

Table 1: Libraries

```
[ ] import NumPy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import
Axes3D
```

Table 1 illustrates the necessary Python libraries for numerical operations, plotting, and 3D visualisations are imported. This includes NumPy for mathematical operations, matplotlib, and mpl_toolkits.mplot3d for plotting and visualising data in 3D space.

❖ **Item Class Definition**

This defines a class Item to represent items with attributes like budget, staff, hardware, and value. This could be used in simulations that involve resource allocation or optimisation tasks.

❖ **Calculation Functions**

Table 2: Cheap Calculation Function

```
[ ] def cheap_calculation(items):
        if items and isinstance(items[0], Item):
            return sum(item.value for item in items)
    else:
        return sum(items)
```

Table 2 shows a Cheap Calculation Function: A function to calculate a simple sum of values for a given list of items or numerical values.

The cheap calculation function is defined as:

$$CheapCalculation(items) = \sum_{item \in items} item.value$$

This function computes a simple sum of the value attributes of a list of Item objects, providing a straightforward metric for evaluation.

Table 3: Expensive Calculation Function

```
[ ] def expensive_calculation(items):
        total_value = 0
        for item in items:
            if is instance(item, Item):
                total_value += item.value * np.log(item.budget + 1) * item.staff / (item. hardware + 1)
            else:
                total_value += item
        return total_value
```

Table 3 illustrates Expensive Calculation Function: A more complex function calculates a sum based on item attributes, adjusted by logarithmic and division operations, to represent a more complex valuation of items.

The expensive calculation involves a more nuanced approach, incorporating multiple attributes of each item:

$$ExpensiveCalculation(items) = \sum_{items \in items} (item.value \times log(item.budget + 1) \times \frac{item.staff}{item.hardware + 1})$$

This formula calculates a composite score by factoring in the logarithm of the budget and the ratio of staff to hardware, thereby providing a more comprehensive evaluation of each value of the items.

❖ **Knapsack Variant Function**

Table 4: Knapsack Variant Function

```
[ ] def select_items(items, max_budget, max_staff, max_hardware):
        return [item for item in items if (
            item.budget <= max_budget and
            item.staff <= max_staff and
            item.hardware <= max_hardware
        )]
```

Table 4 shows the Knapsack Variant Function. This function selects items based on budget, staff, and hardware constraints. It mimics the classic knapsack problem by choosing items within specified limits.

The selection function emulates a variant of the knapsack problem, selecting items based on constraints:

$$SelectItems\begin{pmatrix} items, \max\_budget, \max\_staff, \\ max\_hardware \end{pmatrix} = \{\ item | item.budget \le max\_budget \wedge\ item.staff \le max\_staff \wedge item.hardware \le max\_hardware\}$$

❖ **Agent and Beam Agent Class Definitions**

Agents navigate the optimisation landscape while Beam Agents track and analyse their progress.

Agent Movement and Satisfaction: Agents assess their position based on a threshold and move to optimise their score:

$$IsUnhappy(cheap\_estimate, threshold) = \begin{cases} True & if\ cheap\_estimate < threshold \\ False & otherwise \end{cases}$$

Agent Position Update:

$$NewPosition = CurrentPosition + (Gradient \times LearningRate) + Uniform\ (-1,1)$$

Beam Agent Update: Track agent positions and their fitness estimates, maintaining an average over time.

Table 5: Agent Class Function

```
[ ] Class Agent:
        def __init__(self, position):
            self.position = position

        def is_unhappy(self, cheap_estimate, threshold):
            return cheap_estimate < threshold

        def move_to(self, new_position):
            self.position = new_position
```

Table 5 shows the Agent Class Function: It represents an agent with a position in space, capable of evaluating its satisfaction (is unhappy) and moving to a new position with added jitter.

Table 6: Controller Agent Class Function

```
[ ] Class BeamAgent:
        def __init__(self, max_beam_size):
            self.beam = [# List of tuples (agent_position, cheap_estimate)
            self.max_beam_size = max_beam_size

        def update_beam(self, current_position, cheap_estimate):
            self.beam.append((current_position, cheap_estimate))
            self.beam.sort(key=lambda x: x[1], reverse=True)
            if len(self.beam) > self.max_beam_size:
                self. beam.pop()

        def recruit_agent(self, current_position):
            candidate_positions = [pos for pos, _ in self.beam if not np.array_equal(pos, current_position)]
            if candidate_positions:
                return candidate_positions[0]
            else:
                return current_position
```

Table 6 shows a Beam Agent Class: It manages a collection of agents' positions and their evaluations (cheap estimates), tracking the average fitness over time without a limit on the number of tracked positions.

❖ **Explore Vector Space Function**

Table 7: Vector Space Exploration Function

```
[ ] def explore_vector_space(agents, beam_agent, items, threshold, random_explore_prob=0.1):
        for agent in agents:
            if np.random.rand() < random_explore_prob:
                new_position = np.random.randint(1, 5, size=3)
            else:
                selected_items = select_items(items, *agent.position)
                cheap_score = cheap_calculation(selected_items)
                beam_agent.update_beam(agent.position, cheap_score)

                if agent.is_unhappy(cheap_score, threshold):
                    # if the agent is unhappy, it gets assigned to a new  a new position
                    recruited_position = beam_agent.recruit_agent(agent.position)
                    new_position = recruited_position
                    agent.move_to(new_position)
```

Table 7 illustrates a function that orchestrates agents' exploration of vector space. Agents can move randomly or towards better positions based on item selections and their evaluations. It includes mechanisms for moving agents based on a gradient calculated from the fitness landscape to find optimal positions.

The exploration function guides agents through the vector space:

ExploreVectorSpace(agents,beam_agent, items,threshold,X,Y,Z)

Agents move randomly or towards more optimal positions based on the calculated gradients and item evaluations.

❖ **Compute Fitness Landscape and Gradient Functions**

This function computes the fitness landscape (a hypothetical space representing the success of different strategies or configurations) and the gradient of this landscape at any given position. These are essential for guiding agent movements in a simulated optimisation task.

The fitness landscape is defined by a function $Z = f(X, Y)$, and the gradient at any point is given by:

$$\nabla Z = (\frac{\partial Z}{\partial x}, \frac{\partial Z}{\partial y})$$

∇Z: This is the gradient of Z, denoted by ∇

∂Z: Partial derivatives of Z with respect to x and y. The gradient guides the agents' movement towards more optimal positions.

3.2 Overview

This section details the Stochastic Diffusion Search (SDS) and Beam Agent (BA). Fig. 1 illustrates the proposed hybrid model methodology of the current. Readers are to be reminded that the focus in this current study is not particularly on the dataset; therefore, no specific dataset is required because the focus is on the agent interactions rather than on any particular insights that might be drawn from the dataset.

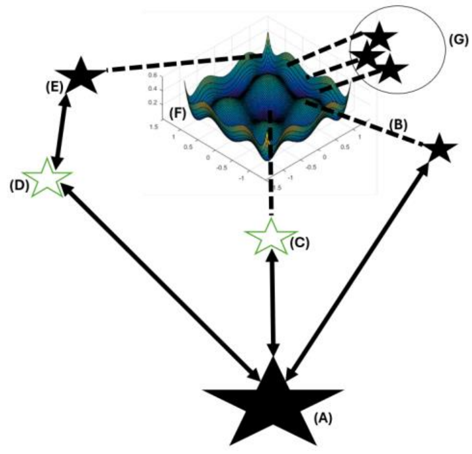

Figure 1: Proposed hybrid model SDS-BA.

The symbols demonstrated in the diagram in Figure 1 are explained below.

(A) Beam or Controller Agent, which is responsible for maintaining a list of potentially high-quality areas in the vector space for recruitment purposes.

(B) An active agent exploration the search space

(C) Inactive agent in the search space communicating with the Beam Agent

(D) In active agent seeking the solution (possible high-quality areas in the vector space) from the Beam Agent

(E) Recruited agent (received areas of exploration from the Beam Agent)

(F) Search space

(G) Converged agents in the search space (agents who found reach areas in the search space).

3.3 The Architecture or Optimisation of Stochastic Diffusion Search (SDS)

In this algorithm, the population is initialised to begin the search in the exploration space. Each agent has their own hypothesis (h) to examine a potential solution in the search space. All the associated phases are explained below:

- **Initialisation Phase**
    - Each agent is randomly allocated to the search space to find a region with documents.

- **Test Phase**

In this stage, SDS checks whether the agent hypothesis is successful by performing a partial hypothesis evaluation and returning a domain-independent Boolean value. Each agent also performs *partial function evaluation*, which is some function of the agent's hypothesis. In this study, the partial function evaluation entails a region or search space which is randomly selected and defined by the agent, and it performs the following hypothesis.

- Each agent conducts exploration in the search space
- Agents are classified into Active (happy) and Inactive (unhappy) groups.

- **Diffusion Phase**

At this level, each agent recruits another agent for interaction and potential communication of the hypothesis. In this study, diffusion is performed by communicating a region with possible solutions.

- Inactive agents have to consider a new region by communicating with another agent.
- If the selected agent is also inactive, there will be no information flow among the agents; instead, the selected agent must consider another region randomly.

3.4 Beam Agent Architecture

As indicated in the above sections, the purpose of the beam agent is to maintain a list of potentially high-quality areas in the vector space that, when stochastic diffusion agents become inactive and need to be recruited somewhere, has a list of possible places for them to go. All the associated stages are explained below:

- Initialises the Beam
- Update the Beam
  - ➢ Update the beam with the current agent's position and cheap estimate.
  - ➢ Parameters;
    - ▪ current_position(tuple): current position of the agent
    - ▪ cheap_estimate (float): cheap estimate value.
- Recruit the Agent
  - ➢ Recruit an agent from the beam based on the current agent's position.
  - ➢ Parameters;
    - ▪ current_position (tuple): current position of the agent.
    - ▪ returns:
      - o tuple: position of the recruited agent.

3.5 The Knapsack Problem

Sapra, Sharma, & Agarwal (2017) define the Knapsack Problem as a combinatorial optimisation maximisation problem that requires finding the number of each weighted item to be included in a hypothetical knapsack so that the total weight is less than or equal to the required weight. Similarly, Nomer, Alnowibet, Elsayed, & Mohamed (2020) indicated that imagine a group of items of known weights and values and a pack or bag with a constrained limit for filling the knapsack. To fill the indicated pack with the items in such a way that their entire sum is probably the highest without exceeding the pack's ability, a problem known as the knapsack problem was formulated. Nomer et al. (2020) revealed that given a set of *N* items numbered from 1 to *n,* each has a value (v_i) and weight (w_i). The maximum weight capacity for the sack is denoted *C*. The knapsack problem is defined as:

$$maximise \sum_{i=1}^{N} v_i \, x_i$$

$$subject\ to \sum_{i=1}^{N} w_i \, x_i \leq C$$

$$x_i \epsilon \, \{0,1\}$$

where v_i w_i C are all positive integers. Nomer et al. (2020) further indicated that informally, the problem is stated as follows: Given a set of items, each with a weight and a value, determine which items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

In the context of this study, a knapsack variant in the project management domain was utilised to test the hybrid model. This knapsack problem had three units. These units include *Budget, Staff and Hardware*. The main goal is to fill in objects inside these three quantities or units while maintaining the optimal balance or solution. In other words, these items are added to the knapsack to maximise the weight without exceeding the capacity or threshold. The results demonstrated a high level of values, indicating a better solution found by the agents in the search space. The results also showed a strong and consistent beam after a series of iterations during the simulation.

3.5.1 Knapsack problem application

Stochastic Diffusion Search requires problems that can be broken down into independently evaluatable subproblems. So, not all problems types are suitable; however, the knapsack problem allows for such a decomposition and for this reason, it has been used in this study. It is worth noting that the Knapsack problem was picked as a testbench for the architecture rather than the problem motivating the algorithm.

4. Experiments, Results and Discussion

This section explains the results, starting with the resources used for the experiment and concluding with the final beam agent Visualisation and Average Fitness over time.

4.1 Beam Agent (Figure 2)

Figure 2 (3D graph) represents the "Final Beam Agent Visualisation." It shows the positions of agents within a three-dimensional space, with the axes representing different factors or dimensions of the optimisation problem that the agents are navigating.

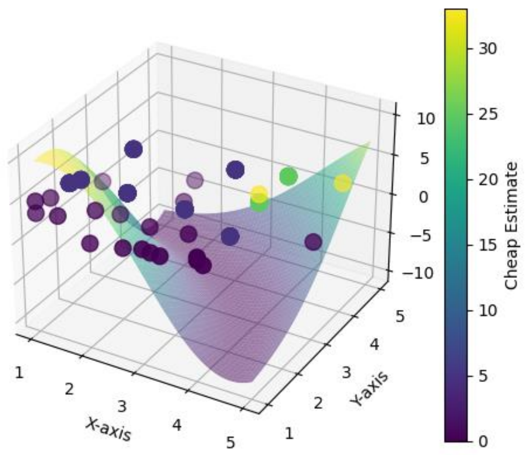

Figure 2: Beam Agent

The interpretation this graph (Figure 2) can be interpreted as below:

- **Axes:** The x, y, and z-axes correspond to the three different dimensions the agents are considering. These could represent different factors, such as budget, staff, and hardware. Each agent position in space represents a potential solution with those specific values for each factor.
- **Surface:** The Coloured surface represents the fitness landscape, with different colours indicating different fitness function values. This fitness function measures how "good" a particular solution is, with higher values being better. The surface is like a plot of the function Z = f (X, Y).
- **Agents (Points):** Each point represents an agent. The position of a point indicates the solution that the agent is currently considering based on the factors represented by the x, y, and z-axes.
- **Colour of Agents:** The colour of the agents corresponds to the value of their "cheap estimate," as indicated by the colour bar on the right side of the graph. A gradient from purple to yellow indicates increasing estimate values, with yellow representing higher values and purple representing lower values.

From this graph, the following summary is observed:

- **Distribution of Solutions:** The spread of the agents across space shows which areas have been explored more densely. Clusters of agents indicate regions where many potential solutions can be found, suggesting that these areas are of particular interest or have higher fitness values.
- **Fitness Landscape:** The shape of the fitness landscape (the Coloured surface) suggests how the fitness value changes with different combinations of the x and y factors. Peaks represent high fitness values, while valleys represent low fitness values. Agents ideally should move towards the peaks to maximise their fitness.
- **Quality of Estimates:** The colour intensity of the agents indicates the quality of their estimates. Agents with a brighter colour (closer to yellow) are considered better in terms of the cheap estimate calculation. An accumulation of bright-Coloured agents in one area could suggest a region of higher fitness.
- **Optimisation Potential:** If agents have high-value estimates on the edges or outside the explored regions of the surface, it could indicate that further exploration might yield even better solutions.

In the context of this study, the visualisation (in Figure 2) helps understand the results of the agent-based optimisation process, showing which solutions were found to be promising according to the cheap calculation function and how these solutions are distributed across the possible space of budget, staff, and hardware configurations.

4.2 Average Fitness Over Time Graph (Figure 3)

The graph (Figure 3) plots Average Fitness Over Time. The below indicates how this graph can be interpreted.

- **X-axis (Iteration):** This represents the iteration number in the optimisation process. Each point corresponds to a single iteration where the fitness is evaluated.
- **Y-axis (Average Fitness):** This axis shows the average fitness value of the agents at each iteration.
- **Data Points:** Each data point represents the average fitness of all agents at a given iteration. This is the result of calculations like the cheap or expensive calculation functions.

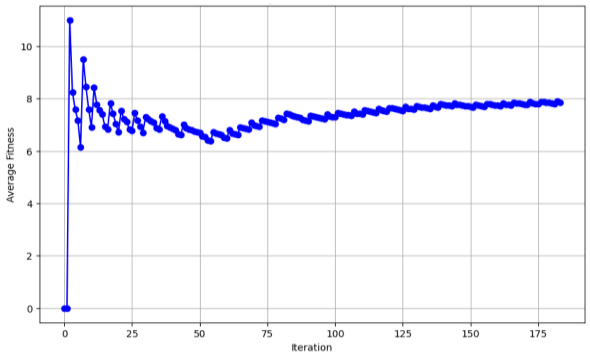

Figure 3: Line Graph for Average Fitness Over Time

Summary of the Graph (Figure 3):

- **Initial Fluctuations:** The early iterations show significant fluctuations in fitness. This indicates a period of exploration where agents are testing a wide variety of solutions.
- **Convergence Over Time:** As iterations progress, the average fitness seems to converge towards a more stable value. This suggests that the agents are refining their search and converging on solutions with similar fitness values.
- **Optimisation Progress:** The general trend appears to be a sharp increase in average fitness initially, followed by a plateau. This pattern is typical in optimisation problems where easy gains are made early, with subsequent improvements being more challenging to find as the solutions approach an optimal state.
- **Stability of Solutions:** The plateau at the later stages indicates that the agents are clustered around the most promising solutions, and additional iterations do not result in significant changes to the average fitness. This could suggest that the optimisation process has found a stable solution or is trapped in a local optimum.
- **Determining Termination:** In this practice, one might use such a graph to determine when to terminate the optimisation process. Since the graph levels are off and remain relatively constant, one could argue that further iterations might not yield substantial improvements.

The behaviour observed in Figure 3 is typical for many optimisation algorithms where initial improvements are rapid, followed by slower convergence as the algorithm fine-tunes the solutions. It can also indicate the balance between exploration (searching new areas of the solution space) and exploitation (refining known good areas).

4.3 Advantages/Benefits of the developed hybrid model (Stochastic Diffusion Search and Beam Agent combination)

This work represents the first time an actor-oriented Stochastic Diffusion Search (SDS) classifier is implemented with a Beam Agent (BA), where the hybrid SDS-BA optimises the recruitment and distribution of inactive agents to a new search space or region. The actors communicate with the controller agent, which maintains a list of data structures called a beam agent (candidate positions in the search space). Each agent reports their data to the beam agent. The controller agent evaluates all the results in order to determine the ranking of clusters in terms of their quality. The search space represents all possible environments where agents can conduct their own exploration. The advantages of this model, compared to a standard (non-actor-oriented) implementation, come from the parallelism and distributed nature of the actor model. For distribution, the model can use a larger number of cheaper and lower-resourced computers to run the system rather than a single, more expensive machine instance for processing. The actor model is inherently fault-tolerant; however, that aspect is not explored in this paper. Future work will consider these potential advantages. The beam can be viewed as a list of potentially high-quality regions within the search space that can be distributed to the agents when they are not satisfied with their performance in whatever area they are located in the search space.

4.4 Summary of the Results

This paper presented a robust and efficient solution that can be used to address optimisation problems in machine learning models. It is a hybrid model that combines Stochastic Diffusion Search and Beam Agent algorithms. This algorithm initialises the population to begin exploration in the search space. Each agent has their own hypothesis (h) to examine a potential solution in the search space. The developed hybrid model was tested using a knapsack variant. This knapsack problem had three units: Budget, Staff and Hardware. The main goal was to fill in objects inside these three quantities or units, ensuring the optimal balance or solution was maintained without exceeding the threshold. As indicated in the results in Figure 2, the classifier showed higher values, which means better solutions found by agents in the search space. The study also developed a line graph (Figure 2) to show how the average fitness score of agents changes over a series of iterations (or time steps) during the simulation. The graph demonstrates a number of changes in the beam, especially when the number of iterations increases.

4.5 Potential trade-offs between computational efficiency and solution quality

Trade-offs exist for all population-based optimisation algorithms. The one presented in this study is no different; at its core, it's a population-based search. The higher the dimensionality of the problem, the longer it takes each actor to evaluate the subproblem that it is working on, which would slow the convergence of the overall algorithm. However, because actors are running in parallel, the whole algorithm should not stagnate. Not all regions of the problem space are equally challenging to evaluate, as some problems in the problem space are more complex. Therefore, in this algorithm, if some of the actors wind up getting into regions where it takes a long time to evaluate the subproblems, the rest of the algorithm can still explore regions where it is quicker to evaluate. Consequently, the whole algorithm does not get stuck if one of the actors wanders into a problematic region, which would happen in a non-actor oriented or non-parallelised version of the algorithm.

5. Conclusion

In the case of machine learning algorithms, the performance of the model is a very critical issue. This developed a hybrid actor based autoencoder model to address the non-distributable nature of the normally used algorithms. The newly developed hybrid algorithm is completely parallelised. Every actor is independent of every other actor and can be run automatically, on its own individual green thread but can, in fact, be run on another computer. The results also showed a strong and consistent beam agent after a series of iterations performed during the simulation. Future studies will compare the model with other local searches beyond beam search for future research. It will also be beneficial to look at applications in component-oriented software engineering, such as automated software factories assembling component-oriented software. Such an approach would search the space of programs by building them up out of already existing subprograms within resource, computational, and size limits.

6. References

Choo, J., Kwon, Y. D., Kim, J., Jae, J., Hottung, A., Tierney, K., & Gwon, Y. (2022). Simulation-guided beam search for neural combinatorial optimization. Advances in Neural Information Processing Systems, 35, 8760-8772.

Cohen, E., & Beck, C. (2019, May). Empirical analysis of beam search performance degradation in neural sequence models. In International Conference on Machine Learning (pp. 1290-1299). PMLR.

Goel, L., Neog, A., Aman, A., & Kaur, A. (2020). Hybrid Nature-Inspired Optimization Techniques in Face Recognition. Transactions on Computational Science XXXVI: Special Issue on Cyberworlds and Cybersecurity, 99-126.

Johansson, R., Johansson, R., & John, S. (2019). Numerical python (Vol. 1). New York, NY: Apress.

Kamaraj, K., Lanitha, B., Karthic, S., Prakash, P. N., & Mahaveerakannan, R. (2023). A Hybridized Artificial Neural Network for Automated Software Test Oracle. Computer Systems Science & Engineering, 45(2).

Lee, W.M. (2019). Python machine learning. John Wiley & Sons.

Lemons, S., López, C. L., Holte, R. C., & Ruml, W. (2022, June). Beam search: faster and monotonic. In Proceedings of the International Conference on Automated Planning and Scheduling (Vol. 32, pp. 222-230).

Majid-al-Rifaie, M., & Bishop, J. M. (2020). Stochastic Diffusion Search: A Tutorial. Swarm Intelligence Algorithms, 307-321.

Maroufpoor, S., Azadnia, R., & Bozorg-Haddad, O. (2020). Stochastic optimization: stochastic diffusion search algorithm. In Handbook of probabilistic models (pp. 437-448). Butterworth-Heinemann.

Martin, A. O. (2021). Stochastic Diffusion Search. In Handbook of AI-based Metaheuristics (pp. 151-198). CRC Press.

Martin, A. O., Bishop, J. M., Robinson, E. J., & Myatt, D. R. (2019). Local termination criteria for Swarm Intelligence: a comparison between local Stochastic Diffusion Search and ant nest-site selection. Transactions on Computational Collective Intelligence XXXII, 140-166.

Meister, C., Forster, M., & Cotterell, R. (2021). Determinantal beam search. arXiv preprint arXiv:2106.07400.

Nomer, H. A., Alnowibet, K. A., Elsayed, A., & Mohamed, A. W. (2020). Neural knapsack: a neural network based solver for the knapsack problem. IEEE access, 8, 224200-224210.

Osaba, E., Del Ser, J., Jubeto, X., Iglesias, A., Fister Jr, I., Gálvez, A., & Fister, I. (2020). Distributed Coordination of Heterogeneous Robotic Swarms Using Stochastic Diffusion Search. In Intelligent Data Engineering and Automated Learning–IDEAL 2020: 21st International Conference, Guimaraes, Portugal, November 4–6, 2020, Proceedings, Part II 21 (pp. 79-91). Springer International Publishing.

Sapra, D., Sharma, R., & Agarwal, A. P. (2017, January). Comparative study of metaheuristic algorithms using Knapsack Problem. In 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence (pp. 134-137). IEEE.

Shanthi, S., & Rajkumar, N. (2021). Lung cancer prediction using stochastic diffusion search (SDS) based feature selection and machine learning methods. Neural Processing Letters, 53(4), 2617-2630.

Stahlberg, F., & Byrne, B. (2019). On NMT search errors and model errors: Cat got your tongue? arXiv preprint arXiv:1908.10090.

Suganya, E., & Rajan, C. J. W. N. (2021). An adaboost-modified classifier using particle swarm optimization and stochastic diffusion search in wireless IoT networks. Wireless Networks, 27(4), 2287-2299.

Tair, M., Bacanin, N., Zivkovic, M., & Venkatachalam, K. (2022). A Chaotic Oppositional Whale Optimisation Algorithm with Firefly Search for Medical Diagnostics. Computers, Materials & Continua, 72(1).