Literature Review for IFT-7020 Research Project

Amirhossein Esmaeilpour
Ph.D. Student in Administration Sciences - Operation and Decision Systems
Supervisor: Dr. Michael Morin
Submit to Prof. Claude-Guy Quimper

1 Abstract

The efficacy of search and rescue missions is crucial for saving lives and minimizing losses during sea-related emergencies. In this research, we seek to improve the current limitation of the "Optimizer" module of Search Planner- the Canadian Coast Guard's decision support system that assists them in maritime search and rescue. In this research, we design a model to prevent the overlap of search and rescue units by integrating constraint programming with black-box optimization. Our model ensures that search areas for different units are non-overlapping, thereby improving operational safety and efficiency. The final results show the effectiveness of our model in generating feasible and optimized search rectangles within tight time constraints.

2 Introduction

The efficacy of search and rescue (SAR) missions is crucial for saving lives and minimizing losses in maritime emergencies. In the past, initial advancements in SAR operations depended significantly on manual techniques. In Canada, the Canadian Search Area Definition ($CSAD$) method, developed based on crash site location data from 1981-1986, was the primary approach used for planning search operations. However, research has shown that manual methods for SAR have significant limitations. They often fail to account for search variables like sweep width, sensor capabilities, and environmental conditions that affect search effectiveness.

In 2005, Prof. Abi-Zeid and Prof. Frost [1] presented a product called "SARPla", a geographic decision-support system designed to assist the Canadian Coast Guard to help them in their search mission to find the missing aircraft. The system was developed to address limitations in existing manual methods, particularly the $CSAD$ approach. SARPlan optimally plans the search mission to maximize the mission's probability of success. They used Search Theory, Gradient Search methods, and Constraint Satisfaction programming in SARPlan. In their research, the search object was the missing crashed aircraft. Therefore, they were working on a stationary search object. To explain the contribution of this paper, we need to briefly explain a few concepts in Search Theory.

In search theory, the ultimate goal is to devise a search plan that maximizes the chances of finding the target object within the minimum amount of time. The Probability of Success (POS) serves as an important indicator. For a static search object, POS is the result of combining the Probability of Containment and the Probability of Detection, which is depicted in $Equation1$. The likelihood that the search item is enclosed within a specific area is called Probability of Containment (POC). *Effort* has units of length or time on the scene, meaning a Search and Rescue Unit can be on the scene for how long or it can traverse how far. The probability of Detection (POD) is the probability of detecting the search object as a function of effort expended in the given area. The sweep width indicates the level of visibility of the search object by the sensors in the current environmental conditions during the search.

$$POS = POD(Effort) \times POC \tag{1}$$

Designing search plans that incrementally optimize POS ensures efficient resource allocation, significantly enhancing the likelihood of a successful search.

SARPlan's key innovation is its optimal effort allocation capability, which helps maximize the POS by optimally allocating search resources based on multiple variables including POC, POD, sweep width considerations, and environmental factors. The system integrates Geographic

Information System (GIS)capabilities allowing users to define search areas using digital maps, create theme grids for various factors, and automatically populate grid cells with relevant data.

SARPlan enables users to outline the possibility area. The primary data structure in SARPlan for data representation is the fundamental grid, which consists of equally sized square cells created over the possibility area by SARPlan. To optimally allocate available effort f, they formulate the below optimization problem:

$$\text{Maximize} \quad \text{POS} = \sum_{j \in J} \text{POC}(j)\text{POD}(j, z(j)) \tag{2}$$

$$\text{subject to} \quad f = \sum_{j} z(j) \tag{3}$$

$POC(j)$ represents the likelihood of the search object being in cell j of the area J divided into cells, $z(j)$ denotes the search effort in cell j, and $POD(j, z(j))$ stands for the detection function. If the search mission at time t-1 was unsuccessful, the input POC grid will be updated with the below penalty function:

$$POC_t(j) = POC_{t-1}(j)[1 - POD_t(j)] \tag{4}$$

Performance testing showed that SARPlan consistently outperformed the traditional Canadian Search Area Definition ($CSAD$) method, demonstrating a 30-50% improvement in POS. The system showed particularly strong performance during crucial initial search hours and superior results across various conditions including poor search conditions and varying sweep width scenarios. This provides different benefits compared to manual techniques.

SARPlan runs on a Windows platform using client-server architecture, implemented in C++ with an ORACLE 8i database. It uses MAPOBJECTS 2.0 for geographic capabilities and incorporates various numerical optimization algorithms.

We can conclude that SARPlan provides significant improvements in search efficiency, leading to more lives saved, reduced SAR costs, and decreased risk to searchers through earlier detection and more efficient resource utilization.

This paper was the start of the idea of the SAR Optimizer and includes different concepts that led to developing the SAR Optimizer for the Canadian Coast Guard which we will cover in the next paper.

In 2019, Prof. Abi-Zeid, Dr. Morin, and their colleagues [2] introduced Search Planner- An artifact intended to suggest the optimal search plans of a search operation, specifically to determine the most effective combination of search rectangles. Search Planner was developed as a decision support system for missing persons in the maritime for the Canadian Coast Guard and they have been using it until now. Their search objects are life rafts or missing persons in the maritime therefore the search objects are moving. In each SAR mission, there are different steps to take to prepare the information we need to find the best Search Area, Start Location, and Movement pattern to search. Here are the steps that will be done when a search and rescue case opens before Search Planner:

i SMC (SAR Mission Coordinators) Planning a Search Mission by creating a SAR Case including all the information that they have, like the available search units, last known location of Search Objects, possible sightings, etc.

ii In the first phase, it produces a Probability Distribution for the Search Objects by randomly seeding the space and also taking into account the particles' last known location. Most programs use a Mont-Carlo-based stochastic drift Simulation in this regard.

iii Based on their drift model, which contains information on surface currents and winds, the particles will move in time and space by the simulation.

iv The temporal arrangement of particle positions within each set serves as a probable trajectory for a sought-after object [2, 4]. The Simulation output contains the particle's position in each time step and will be the input to the Search Planner.

v At this step, the SMC has the option to either **generate a search operation manually and submit it to the Search Planner for assessing its Probability of Success**, or

it can ask the Search Planner to propose a search operation and, consequently, activate the Optimizer module in the process [2].

2.1 Search Planner

Search Planner consists of three modules: "Simulator", "Evaluator", and "Optimizer". Here are the steps in the Search Planner:

1. Input parameters: Number of search units (U), Drift File
2. For each search unit u in $1 \dots U$:
   - **A** Construct a Convex Hull based on its on-scene time.
   - **B** Create a Minimum Spanning Rectangle around the Convex Hull (Region of Interest or ROI).
   - **C** Adjust the ROI by shrinking, enlarging, or shifting its center, depending on constraints, to generate a set of feasible Search Rectangles ($SR(u)$).
3. Treat $SR(u)$ as a Candidate Search operation.
4. Simulate the search units into those rectangles and evaluate to find the POS for the search operation.
5. Compare the result with the highest POS so far and update the best POS if necessary.

The Search Planner strives to maximize POS by positioning the search units within rectangles while adhering to specified constraints. These rectangles define the paths, which are then utilized to compute POS. The optimization problem is described by the variables that define the rectangles, including their center (two variables), length, width, and orientation (θ). However, these five variables are insufficient to fully specify the search path. The induced path comprises parallel segments known as search legs, linked by perpendicular segments referred to as cross legs, all of which are of equal length. In Figure 1 you can see an example of a Search Operation designed by the Search Planner for 2 search units in the maritime. The dotted rectangle is the ROI and the brown vector shows the mass center of the drift instance "D1". The red and Green rectangles are the search rectangles generated by SAR Optimizer that overlap each other.

The evolution of SAR decision support systems is demonstrated through these two papers, showing a progression from stationary to dynamic search objects. SARPlan represented a pioneering effort focused on stationary targets (crashed aircraft), introducing the concepts of optimal effort allocation and GIS integration to improve upon the manual $CSAD$ method. The 2019 SAR Optimizer builds upon these foundational concepts but makes a significant leap forward by addressing moving objects (life rafts, persons) in maritime environments, and by its modular architecture (Simulator, Evaluator, and Optimizer) introducing innovations such as simulation-based evaluation and dynamic search rectangle optimization.

In our optimization project, we plan to manually generate an optimal search rectangle for a search unit and submit it to the Search Planner to assess its Probability of Success. Therefore, we only use SAR Optimizer to evaluate our plan.

Black-box optimization (BBO) contains optimization problems where the objective function and constraints can only be evaluated through external processes - whether through computer simulations, existing software systems, or physical experiments. In these cases, we don't have direct access to the mathematical form of these functions; instead, we can only observe their outputs for given inputs [3]. The BBO problems can be structured as follows:

$$\min_{x} f(x, y) \tag{5}$$

subject to:

$$g_i(x, y) \leq 0 \quad \forall i \in P := 1, \dots, p \tag{6}$$

where $x \in R^n$ are the independent variables, $y \in R^q$ are the dependent variables, f(x,y) is the objective function, and g(x,y) denotes the constraints [3].

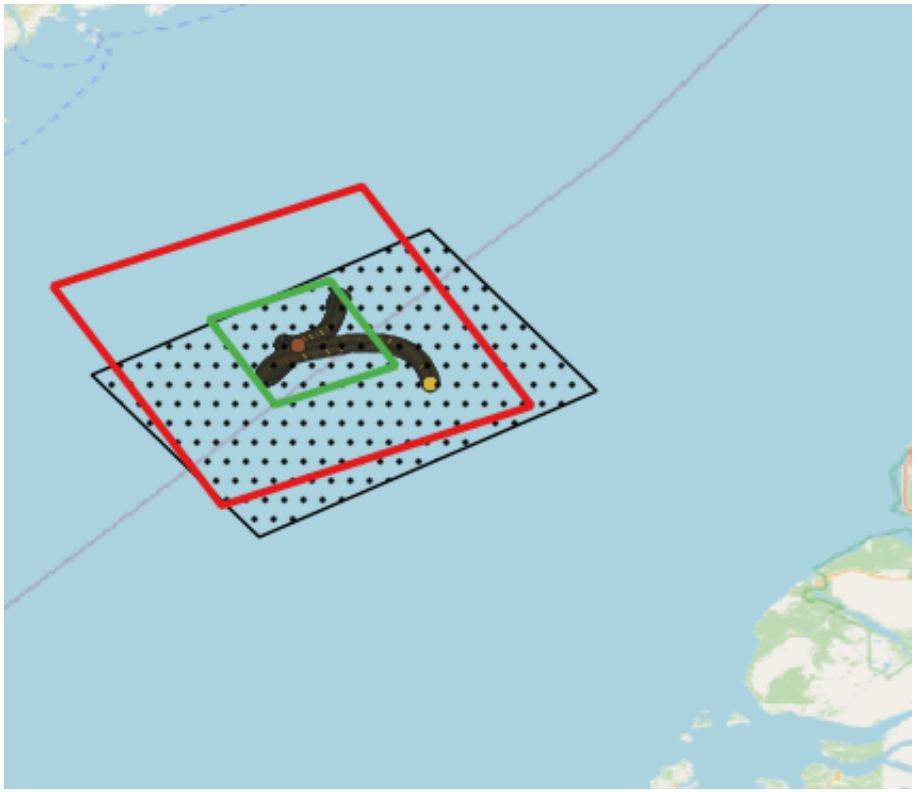

Figure 1: An example Search Operation for 2 search units on Instance D1 before using our proposed method

"Hexaly" Solver, previously called "LocalSolver" [5], is a global optimization solver. It combines the heuristic and exact operational research methods and allows to modelling of different optimization problems like combinatorial, mixed, and continuous and solves them on large instances. This solver also supports the Black-Box optimization. The black-box optimization method used in the hexaly solver is based on Gutmann's radial basis function substitution method. It uses an RBF (Radial Basis Function) type surrogate to handle constraints in the objective function. Each constraint is assigned a margin criterion to ensure feasible solutions. This criterion is updated iteratively based on how feasible the previous evaluation point was. This approach improves the likelihood of finding the optimal solution, even when it lies at the boundaries of the solution space [6]. The hexaly solver proposed different implementation options in different programming languages.

The concepts and definitions introduced in this section form the foundation for our next section. In the following sections, we will formulate our problem and detail our proposed approach.

3 Problem Description

In a search and rescue operation, sometimes the available search and rescue unit can have the same type. For example, in a maritime incident, SMC plans the search mission by checking the available search and rescue units. Consider the only available search unit at that time was Helicopter and Fixed-wing. The use of the same types of search and rescue units which in this case is aircraft, can cause collision and safety problems when their search area is the same location. Currently, in the search planner, there is no limitation for different search and rescue units' on-scene positions and in the "Optimizer" module, any possible ROI can be a feasible search rectangle for a search and rescue unit. Therefore, it is possible to have two search rectangles partially or fully overlap each other. This study addresses this problem by proposing a combination of constraint programming and black-box optimization.

The Maritime Search and Rescue (MSAR) rectangle placement problem takes as input:

- A maximum search rectangle (Minimum Spanning Rectangle around the Convex Hull of

search objects) defined by four corners in Mercator projection coordinates: $(max_{ul_x}, max_{ul_y}), (max_{ur_x}, max_{ur_y}), (max_{lr_x}, max_{lr_y}), (max_{ll_x}, max_{ll_y})$

- Based on the SMC case and our resources, for each Search and Rescue Unit $i \in \{1, 2\}$:
  - Minimum required search area A_i^{min} in square meters
  - Maximum allowed search area A_i^{max} in square meters

A valid solution consists of two non-overlapping rectangles R_1 and R_2, where each rectangle:

- Must lie within the maximum search rectangle's edges
- Must maintain rectangular shape (parallel to coordinate axes)
- Must have an area within its required bounds
- Must not overlap with the other rectangle

4 Proposed Method

To address the mentioned problem, we combine constraint programming with black-box optimization. We define a model that can generate two search rectangles which are limited to each search and rescue unit's resources and we used the Evaluator module of the Search Planner for the objective function. Therefore, we have an external black-box objective function.

In the first part, we read a SAR case file and got the important information that we needed like "Total search effort", "Min and Max Track Spacing", "search unit On-scene time", and "Search Speed" related to the case. Then we read the input drift file that contains the position of each search unit in different time steps. The idea is to extract the data for the on-scene time of that search unit, then do the same thing that the Search Planner do for each search unit (explained in 2.1) like create a context hull around the search objects and create a Maximum rectangle (ROI) around that. Then the output of this section will be the input of our constraint programming.

Here is the mathematical model of our proposed constraint programming method:

Bounds derived from maximum rectangle:

$$
\begin{aligned}
max_x &= \max(max_{ul_x}, max_{ur_x}, max_{lr_x}, max_{ll_x}) \\
max_y &= \max(max_{ul_y}, max_{ur_y}, max_{lr_y}, max_{ll_y}) \\
min_x &= \min(max_{ul_x}, max_{ur_x}, max_{lr_x}, max_{ll_x}) \\
min_y &= \min(max_{ul_y}, max_{ur_y}, max_{lr_y}, max_{ll_y})
\end{aligned}
$$

Variables:

- For each rectangle $i \in \{1, 2\}$:
  - Upper left corner: $(ul_{ix}, ul_{iy}) \in [min_x - b, max_x + b]$
  - Upper right corner: $(ur_{ix}, ur_{iy}) \in [min_x - b, max_x + b]$
  - Lower right corner: $(lr_{ix}, lr_{iy}) \in [min_x - b, max_x + b]$
  - Lower left corner: $(ll_{ix}, ll_{iy}) \in [min_x - b, max_x + b]$
    where b is a buffer parameter which we set 0 for that, and min_x, max_x are derived from the maximum rectangle corners.
  - Variable indicating the position of each search unit concerning another search unit (for example z_1 can be 1 only if the position of the second search unit be at the left side of the first search unit) $z_1, z_2, z_3, z_4 \in \{0, 1\}$

Constraints:

1. Rectangle shape (for each $i \in \{1, 2\}$):

$$\begin{aligned} ul_{ix} &= ll_{ix} \\ ur_{ix} &= lr_{ix} \\ ul_{iy} &= ur_{iy} \\ ll_{iy} &= lr_{iy} \\ ul_{iy} &> ll_{iy} \\ ul_{iy} &> lr_{iy} \\ ul_{ix} &< ur_{ix} \\ ul_{ix} &< lr_{ix} \\ ur_{iy} - ll_{iy} &= ul_{iy} - lr_{iy} \\ ur_{ix} - ll_{ix} &= lr_{ix} - ul_{ix} \end{aligned}$$

   These constraints ensure that each rectangle maintains a valid rectangular shape with aligned edges. They enforce that the upper-left corner is positioned above and to the left of the other corners.

2. Area requirements (for each $i \in \{1, 2\}$):

$$\begin{aligned} width_i &= |lr_{ix} - ll_{ix}| \\ height_i &= |ul_{iy} - ll_{iy}| \\ A_i^{min} - t \leq width_i &\times height_i \leq A_i^{max} + t \end{aligned}$$

   where t is a tolerance parameter which we set to 1. This constraint ensures that our generated rectangles follow the area requirement of the input SAR case file.

3. Non-overlapping (exactly one must be true):

$$z_1 = 1 \implies ul_{1x} + ob > ur_{2x} \tag{7}$$
$$z_2 = 1 \implies ul_{2x} + ob > ur_{1x} \tag{8}$$
$$z_3 = 1 \implies ll_{1y} + ob > ul_{2y} \tag{9}$$
$$z_4 = 1 \implies ul_{1y} + ob < ll_{2y} \tag{10}$$

   where ob is an overlap buffer parameter that makes a little separation between the rectangle and we set it to 10.

4. Enforce that exactly one condition is true:

$$z_1 + z_2 + z_3 + z_4 = 1$$

   These two conditions ensure the second search rectangle is either on the right, left, up, or down of the first search rectangle and is not over the first one.

5. Symmetry breaking (for each $i \in \{1, 2\}$):

$$\begin{aligned} &\text{If } max\_y\_height \geq max\_x\_width: \\ &\qquad height_i \geq width_i \\ &\text{Otherwise:} \\ &\qquad width_i \geq height_i \end{aligned}$$

   where $max\_y\_height = \max(|max_{ul_y} - max_{ll_y}|, |max_{ur_y} - max_{lr_y}|)$
   and $max\_x\_width = \max(|max_{ur_x} - max_{ul_x}|, |max_{lr_x} - max_{ll_x}|)$

   This prevents symmetric solutions where rectangles can be rotated 90 degrees.

Objective Function:

$$\begin{aligned}\text{Maximize } POS = \text{external\_function}(&ul_{1x}, ul_{1y}, ur_{1x}, ur_{1y}, lr_{1x}, lr_{1y}, ll_{1x}, ll_{1y},\\ &ul_{2x}, ul_{2y}, ur_{2x}, ur_{2y}, lr_{2x}, lr_{2y}, ll_{2x}, ll_{2y})\end{aligned} \tag{11}$$

The evaluator module of the Search Planner needs a valid SAR case file as input, and in that case file, we have to insert the search rectangle's coordinates and the search pattern for that search rectangle. Therefore the $external\_function$ will generate the related search pattern for the generated rectangles, create the modified SAR case, run the evaluator module of Search Planner and return the value of POS to the objective function.

5 Experimental protocol

In this study, we used 4 instances of drift files. Each instance is a CSV file containing information on the position of 5000 search objects simulated in different time steps of 5 minutes. In addition, each instance includes a case file containing all the information about the search units, such as the on-scene time, track spacing, speed, etc. We use this information in our model to generate a suitable search rectangle for each search unit type. In our study, we used Hexaly solver because it supports both Python and external functions. Therefore, we could read the input files in Python and calculate the required information for our model. Also, we could call the "SAR Optimizer" of the "Search Planner" from a Python method, and then that method would be a part of our external function. Our experiment includes two phases. We ran our model with and without the Surrogate function of hexaly. This was because we wanted to know if this function contributed to our results or not. Since the result of our model without the surrogate was deterministic, we ran it once. In contrast, we ran the model with the surrogate function 30 times. Because time is an important resource in any search and rescue operation, we set the time limit to 45 seconds.

Here is the summary of our experimental configurations:

1. Deterministic Phase:
    - Model execution without Surrogate function
    - Single run per instance due to deterministic nature
    - Time limit: 45 seconds per run
2. Stochastic Phase:
    - Model execution with Surrogate function enabled
    - 30 independent runs per instance
    - Time limit: 45 seconds per run

We performed a statistical analysis with Mean Performance, Standard Deviation, and Best solution found. Also, The 95% confidence intervals are calculated using Student's t-distribution. For each instance, a box plot is generated to show the results.

6 Results

The measurements collected from our experimental protocol are related to solution quality measurement.

Table 1 presents the comparative results of both optimization approaches across all test instances. While the deterministic approach consistently resulted in infeasible solutions, the stochastic approach with surrogate modelling achieved feasible solutions in all instances, with success rates of 100% (30/30 runs). Instance D1 showed the highest variability (std = 0.021) while B1 demonstrated the most consistent results (std = 0.001). As you can see, without the surrogate function, all instances couldn't find a feasible and logical solution, since the domain is continuous and there are unlimited possibilities for generating rectangles and this is because the time limitation is tight (45 seconds). However, within that tight time limit, the surrogate function helped the model not only effectively find the feasible rectangles, but also got a very high probability of success.

Table 1: Comparison of Deterministic and Stochastic Optimization Results

Instance	Deterministic	Stochastic (30 runs)				
	Status	Status	Best POS	Mean POS	Std	CI (95%)
D1	INFEASIBLE	FEASIBLE	0.999	0.974	0.021	[0.966, 0.982]
B1	INFEASIBLE	FEASIBLE	0.996	0.995	0.001	[0.994, 0.995]
C1	INFEASIBLE	FEASIBLE	0.567	0.534	0.037	[0.521, 0.548]
A1	INFEASIBLE	FEASIBLE	0.933	0.921	0.009	[0.918, 0.925]

Note: CI = Confidence Interval

Figure 2 shows the search rectangles for both deterministic and stochastic approaches across all test instances (A1, B1, C1, D1). For each instance, the left panel represents the deterministic search pattern, while the right panel shows the stochastic surrogate-assisted search pattern.

The stochastic approach shows better coverage of the search space, which likely contributed to its success in finding feasible solutions in all instances. In addition, for C1, the notably lower performance metrics (mean = 0.534, std = 0.037) can be attributed to its unique Z-shaped on-scene data distribution. This zigzag pattern presents a particularly challenging scenario for rectangle fitting because the sharp turns in the drift data make it difficult to achieve high coverage with rectangular shapes. These visual patterns align with our numerical results, where the stochastic approach consistently outperformed the deterministic method by finding feasible solutions across all test instances.

We performed statistical analysis with Mean Performance, Standard Deviation, and Best solution found. Also, The 95% confidence intervals are calculated using Student's t-distribution. For each instance, a box plot is generated to show the results.

Figure 3 presents the box plot analysis of the stochastic surrogate-assisted optimization results across four test instances (A1, B1, C1, and D1). The deterministic approach results are not included as they yielded infeasible solutions with objective values of 0 across all instances.

Comparing the four instances reveals distinct performance patterns in the stochastic optimization approach. Instance B1 demonstrated the most robust performance with consistently high objective values (median = 0.995) and minimal variation, indicating reliable solution quality across multiple runs. Instance D1 also achieved strong results with objective values between 0.96-1.00, though with notably more variation than B1. A1 showed good performance with values centered around 0.92, displaying moderate variability. In contrast, C1 exhibited significantly lower performance (median = 0.54) and the highest variability among all instances, with outliers dropping as low as 0.45. This performance hierarchy (B1 > D1 > A1 > C1) shows the varying levels of complexity in the underlying geometric distributions of each instance, with C1's Z-shaped distribution presenting the most challenging case for rectangular approximation. The outliers in C1 and B1 represent runs where the algorithm found less optimal solutions, highlighting that while the method is generally consistent, it can occasionally converge to inferior results.

In table 2 you can see the result of SAR Optimizer or it is better to say SAR Optimizer without the overlap constraints (like using the input case and input data without our interference). The algorithm demonstrated high POS values for most instances. Instance D1 achieved the best performance with a POS of 0.999 and the fastest execution time of 31.8 seconds. B1 and A1 also showed strong results with POS values of 0.991 and 0.978 respectively, though requiring nearly double the execution time (approximately 60 seconds each). C1, with its challenging Z-shaped distribution, achieved a POS of 0.780, significantly higher than constrained versions but still the lowest among all instances. Notably, while these results show higher POS values compared to the constrained version, they don't represent practically viable solutions due to the potential rectangle overlaps. Also, directly comparing our POS values with SAR Optimizer is not feasible since the search pattern generation methodologies may differ between implementations.

Table 2: SAR Optimizer Performance Without Overlap Constraint

Instance	POS	Execution Time (s)
D1	0.999	31.8
B1	0.991	61.9
C1	0.780	58.8
A1	0.978	59.8

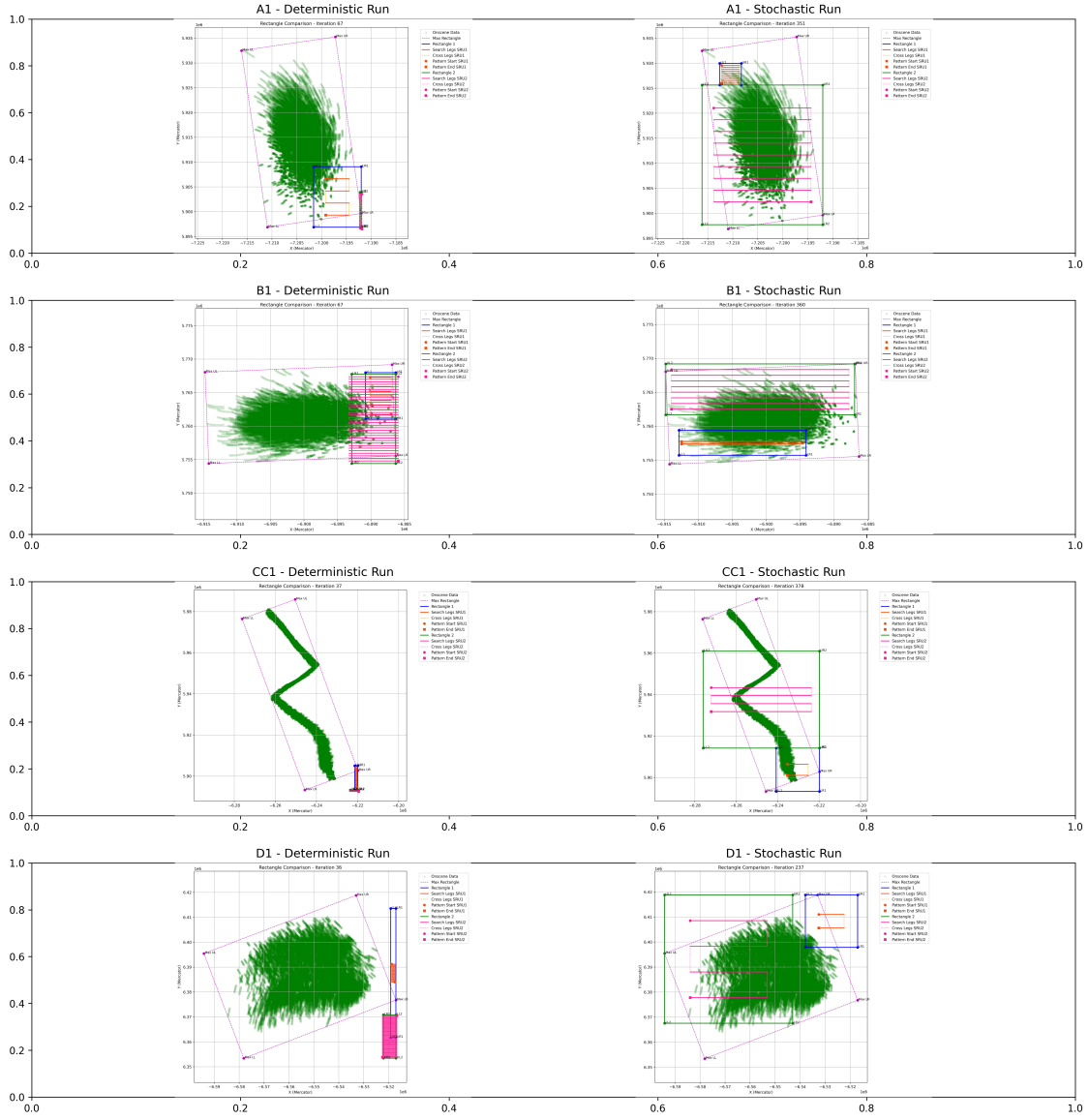

Figure 2: Generated search rectangles for both deterministic and stochastic approaches across all instances

7 Discussion

Our experimental results showed that our model without the surrogate function couldn't find a feasible solution within this time constraint. This is because the domain is continuous and there are unlimited possibilities for generating rectangles without considering previously gathered data. However, with the surrogate function enabled, we not only effectively found feasible rectangles but also achieved a very high probability of success values.

We were working on longitude and latitude projected using Mercator, and since our problem was continuous, even with a high value of POS there could be unlimited rectangles to get the best results. This is because the difference of each geographical percent degree can change the rectangle but not necessarily contribute to or detract from the solution quality.

When we ran the MSAR without using our model (without overlap constraints), it achieved higher POS values (0.999 for D1, 0.991 for B1), but the rectangles generated didn't have the overlap constraint. While these unconstrained results showed higher POS values compared to our constrained version, they don't represent practically viable solutions due to the potential rectangle overlaps. Our constrained model, though achieving slightly lower POS values, ensures non-overlapping search patterns that are implementable in real SAR operations.

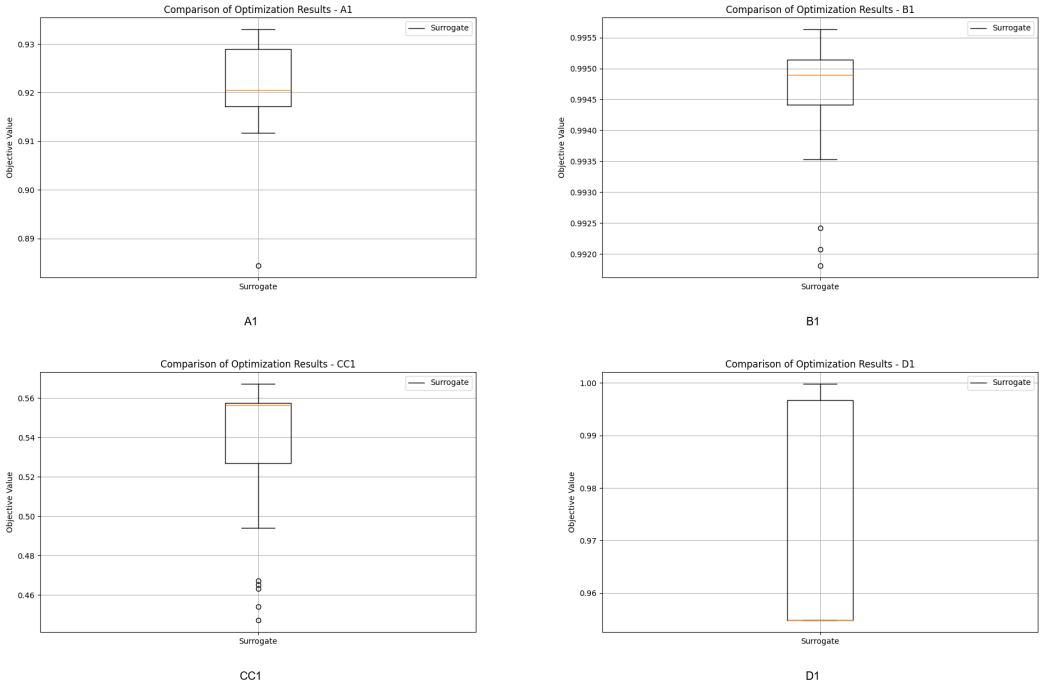

Figure 3: The box plot of the stochastic surrogate-assisted optimization results across four test instances

8 Conclusion

This study addresses the overlapping problem of search areas in Maritime Search and Rescue operations for moving search objects by integrating constraint programming with black-box optimization. The proposed method ensures that search rectangles generated for different search units are non-overlapping, thereby enhancing operational safety and efficiency.

Our experimental results demonstrate that the deterministic approach fails to find feasible solutions within the imposed time constraints due to the continuous nature of the search domain. In contrast, the stochastic approach utilizing surrogate modelling consistently identifies feasible and effective search rectangles across all test instances, achieving a high value of POS within the 45-second time limit. This highlights the significant advantage of using surrogate models in optimizing complex and real-time decision-making processes.

Furthermore, while the unconstrained SAR Optimizer achieved higher POS values, these solutions could be impractical for real-world applications due to potential overlaps. Our constrained model, though slightly lower in POS, provides practically viable and safe search patterns essential for successful SAR missions.

Future work could explore the scalability of this approach to accommodate more search units and larger search areas, as well as targeting to improve the POS of the non-spherical drift shapes.

9 Appendix

You can find the programming code in the attachment.

References

[1] Irene Abi-Zeid and John R Frost. Sarplan: A decision support system for canadian search and rescue operations. *European Journal of Operational Research*, 162(3):630–653, 2005.

[2] Irène Abi-Zeid, Michael Morin, and Oscar Nilo. Decision support for planning maritime search and rescue operations in canada. *SciTePress*, 2019.

[3] Ishan Bajaj, Akhil Arora, and MM Faruque Hasan. Black-box optimization: Methods and applications. In *Black box optimization, machine learning, and no-free lunch theorems*, pages 35–65. Springer, 2021.

[4] Oyvind Breivik and Arthur A Allen. An operational search and rescue model for the Norwegian Sea and the North Sea. *Journal of Marine Systems*, 69(1-2):99–113, 2008. Publisher: Elsevier.

[5] Hexaly. Constrained black-box optimization.

[6] Emeline Tenaud. Optimisation de fonctions boîtes noires avec et sans contraintes. In *23ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, 2022.