

Table des matières

Introduction	1
1 Chapitre 1 : État de l'art	3
1.1 Introduction	3
1.2 Problématique	3
1.3 Paramètres du problème	4
1.3.1 Les systèmes multi-robots (SMRs)	5
Définition	5
Caractéristiques des SMRs [1]	5
Système de communication [2]	5
Comportement des robots [2]	6
1.3.2 Les capteurs	6
1.3.3 Les cibles	6
1.3.4 L'environnement [2]	6
1.3.5 Les obstacles	6
1.4 Travaux connexes	7
1.4.1 Les algorithmes anti-regroupements [3]	7
1.4.2 Approche des champs de potentiel [2]	7
1.4.3 Approches basées intelligence en essaim	7
BSO (Bee Swarm Optimization) (2013)	7
A-RPSO	8
MFSO (Multi-swarm hybrid FOA-PSO)	8
1.5 Conclusion	9
2 Chapitre 2 : Intelligence en essaim : BSO, EHO, EWSWA et GA	10
2.1 Introduction	10
2.2 Les algorithmes basées intelligence en essaim	10
2.3 Bee Swarm Optimization [4]	11
2.3.1 Inspiration des abeilles	11
2.3.2 Analogie entre le comportement des abeilles et BSO	12
2.3.3 Paramètres et fonctions	13
2.3.4 Pseudo-code BSO	14
2.4 EHO & EWSWA : Deux algorithmes basés essaim d'éléphants	15
2.4.1 Inspiration des éléphants	15
2.4.2 EHO : Elephant Herding Optimization	16
Analogie entre les éléphants et EHO [5]	16
Paramètres et fonctions [5]	16
Pseudo code de EHO [5]	18
2.4.3 EWSWA : Elephant Swarm Water Search Algorithm	19
Analogie entre les éléphants et EWSWA [6]	19
Paramètres et fonctions [6]	20
Pseudo code de EWSWA [6]	22
2.4.4 Remarque	23
2.5 Algorithme Génétique [7]	23

2.5.1	Inspiration de la génétique	23
2.5.2	Métaphore entre l'ADN et un algorithme génétique	23
2.5.3	Paramètres et fonctions	24
	Opération de croisement	24
	Opération de mutation	25
2.5.4	Pseudo code (GA incrémental)	26
2.6	Conclusion	26
3	Chapitre 3 : Modélisation du problème de recherche de cibles	28
3.1	Introduction	28
3.2	Modélisation de l'environnement de recherche	28
	3.2.1 Représentation de la solution	29
	3.2.2 Fonction objectif	29
	3.2.3 Les cibles	30
	3.2.4 Les obstacles	30
3.3	Stratégies d'évitement d'obstacles	31
	3.3.1 Description des approches existantes	31
	Les algorithmes Bug [8, 9]	31
	La méthode basée sur les champs de potentiel PF (Potential Field) [8]	32
	La méthode basée sur la fenêtre dynamique DW (Dynamic Window) [8]	32
	La logique floue	32
	L'échantillonnage de l'espace d'entrée ISS (Input Space Sample) .	33
3.3.2	Discussion et choix	33
3.3.3	Paramètres de la méthode d'échantillonnage	34
	Champ de vision	34
	Portée locale d'un robot	34
	Trajectoire	35
	Critères de choix de la trajectoire	36
3.4	Auto-Paramétrage avec GA	37
	3.4.1 Modélisation	37
	Solution	37
	Espace des solutions	38
	Fonction objectif	38
	Population	38
	Croisement	39
	Mutation	39
	3.4.2 Fonctionnement	39
3.5	Conclusion	41
4	Chapitre 4 : BSO pour le problème de recherche de cibles	42
4.1	Introduction	42
4.2	Mono-BSO vs Multi-BSO	42
4.3	Adaptation de BSO pour le problème de la recherche de cibles	42
	4.3.1 Solution	42
	4.3.2 Fonction objectif	43
	4.3.3 Flip	43
	4.3.4 MaxChance	43
	4.3.5 Table Dance	43
	Critère de qualité (Best In Quality)	43
	Critère de Diversité (Best In Diversity)	44

4.3.6	Fonctionnement du Mono-BSO	44
	Initialisation de la solution de référence (Sref)	45
	Insertion de Sref dans la liste <i>Tabou</i>	45
	Génération des zones de recherche	45
	Affectation des zones aux abeilles	45
	Recherche locale de chaque abeille	45
	Déplacement des abeilles	45
	Insertion des meilleures solutions dans la table <i>Dance</i>	45
	Choix de la nouvelle Sref	46
	Critère d'arrêt de la recherche	46
4.4	Multi-BSO : fonctionnement	48
4.4.1	Initialisation des solutions de référence (Srefs)	48
4.4.2	Insertion de Sref dans la Liste Tabou	48
4.4.3	Génération des zones de recherche	48
4.4.4	Affectation de chaque zone à une abeille	48
4.4.5	Recherche locale pour chaque abeille	48
4.4.6	Insertion des solutions dans les tables <i>Dance</i>	48
4.4.7	Test de la 1 ^{ère} condition d'arrêt	48
4.4.8	Choix de la nouvelle solution de référence	49
4.4.9	Déplacement des robots	49
4.4.10	Test de la 2 ^{ème} condition d'arrêt	49
4.5	Conclusion	49
5	Chapitre 5 : Algorithmes des éléphants pour le problème de recherche de cibles	50
5.1	Introduction	50
5.2	Adaptation de EHO pour le problème de la recherche de cibles	50
5.2.1	Solution	50
5.2.2	Fonction objectif	50
5.2.3	Éléphant	50
5.2.4	Clan	51
5.2.5	Chef de clan (Matriarche)	51
5.2.6	Éléphant mâle	52
5.3	EHO : Fonctionnement	52
5.3.1	Initialisation des positions des clans	53
5.3.2	Initialisation des positions des éléphants	53
5.3.3	Évaluation des solutions	53
5.3.4	Tri des clans	53
5.3.5	Mise à jour des positions des éléphants	53
5.3.6	Déplacement des robots	53
5.3.7	Test de la condition d'arrêt	53
5.4	Adaptation de EWSWA pour le problème de la recherche de cibles	55
5.4.1	Solution	55
5.4.2	Fonction objectif	55
5.4.3	Éléphant	55
5.4.4	Outils de mémoire	55
5.4.5	Vélocité	56
5.4.6	Mécanisme de déplacement	56
	Mise à jour de la vélocité de l'éléphant	56
	Mise à jour de la position de l'éléphant	57
	Mise à jour de w^t (poids d'inertie)	57
5.5	EWSWA : Fonctionnement	58

5.5.1	Initialisation	58
	Paramètres empiriques	58
	Positions initiales des éléphants	58
	Pbest pour chaque éléphant	58
	Gbest	58
5.5.2	Mise à jour de la vitesse pour chaque éléphant	58
5.5.3	Calcul de la prochaine position de chaque éléphant	58
5.5.4	Déplacement des robots	59
5.5.5	Évaluation des positions de chaque éléphant	59
5.5.6	Mise à jour de Pbest de chaque éléphant	59
5.5.7	Évaluation des positions des éléphants	59
5.5.8	Mise à jour du Pbest de chaque éléphant	59
5.5.9	Mise à jour de Gbest	59
5.5.10	Mise à jour du W^t (poids d'inertie)	59
5.5.11	Critère d'arrêt de la recherche	59
5.6	Conclusion	60
6	Chapitre 6 : Validation Expérimentale	61
6.1	Introduction	61
6.2	Environnement de développement	61
6.2.1	Matériel	61
6.2.2	Logiciels	61
6.3	Expérimentations relatives aux approches	62
6.3.1	Paramétrage du mini-GA incrémental	62
6.3.2	Paramétrage des approches de recherche de cibles	62
	Réglage de Paramètres de BSO	63
	Réglage de Paramètres de Multi-BSO	63
	Réglage de Paramètres de EHO	63
	Réglage de Paramètres de EWSWA	64
6.4	Expérimentations relatives à l'environnement	65
6.4.1	Organisation des types d'expérimentations	65
	Organisation des expérimentations par rapport à la portée des cibles	65
	Organisation des expérimentations par rapport à la taille de l'environnement	65
	Organisation des expérimentations par rapport au nombre de cibles	66
6.4.2	Expérimentations par rapport à la portée des cibles	67
	Taux de réussite	67
	Variation du nombre d'itérations et temps d'exécution	67
	- Analyse relative à la portée des cibles	72
6.4.3	Expérimentations par rapport à la taille de l'environnement	72
	Taux de réussite	72
	Variation du nombre d'itérations et temps d'exécution	74
	- Analyse relative à la taille des environnements	79
6.4.4	Expérimentations par rapport au nombre de cibles	80
	Taux de réussite	80
	Variation du nombre d'itérations et temps d'exécution	80
	Analyse relative au nombre de cibles	82
6.4.5	Comparaison des types d'environnement	83
6.4.6	Comparaison de nos quatre approches	83
6.5	Simulateur	83
6.5.1	Fonctionnement	83
6.5.2	Interface	84

Environnement	85
Métaheuristiques	86
6.6 Conclusion	89
Conclusion générale et perspectives	90

Listes des figures

1.1	Schéma résumant les classes de problèmes de détection de cibles [2].	4
1.2	Exemple de capture [2]	4
1.3	Exemple de patrouille [2].	4
2.1	Classification des méta-heuristiques	11
2.2	Croisement en un point.	24
2.3	Croisement en deux points.	24
2.4	Mutation avec inversement.	25
2.5	Mutation avec permutation.	25
3.1	Représentation de l'environnement.	29
3.2	Représentation d'une solution dans notre modélisation.	29
3.3	Représentation de la portée d'une cible dans notre modélisation.	30
3.4	Calcul du chemin du robot par les algorithmes Bug : (a) Bug1, (b) Bug2 [9].	31
3.5	Champs de potentiel pour un environnement contenant deux obstacles et une position but.[10]	32
3.6	L'approche de la fenêtre dynamique montrant les régions admissibles et interdites par rapport à la fenêtre dynamique des vitesses atteignables par le robot [8].	32
3.7	Ensemble de trajectoires générées par échantillonnage de l'espace d'entrées [11].	33
3.8	Détermination de l'angle de vue d'un robot dans notre modélisation.	34
3.9	Représentation de la portée locale d'un robot dans notre modélisation.	35
3.10	Sélection des points(cases) d'une droite avec l'algorithme de BRESENHAM.	36
3.11	Choix de la trajectoire à suivre par le robot dans un environnement à obstacle.	36
3.12	Choix de la trajectoire à suivre par le robot dans des situations complexes.	37
3.13	Croisement de deux solutions.	39
3.14	Mutation d'une solution.	39
3.15	Organigramme du mode de fonctionnement de la méthode GA.	40
4.1	Représentation de la méthode de génération de solutions avec le flip.	43
4.2	Méthode de sélection de la meilleure solution en termes de Qualité.	44
4.3	Méthode de sélection de la meilleure solution en termes de Diversité.	44
4.4	Organigramme du mode de fonctionnement de l'approche mono-BSO.	47
4.5	Organigramme du mode de fonctionnement de l'approche multi-BSO.	49
5.1	Représentation de la méthode de mise à jour des positions des éléphant.	51
5.2	Représentation de technique de génération des positions des éléphants d'un même clan.	51
5.3	Représentation de la méthode de mise à jour de la position du meilleur éléphant/robot.	52
5.4	Représentation de la méthode de mise à jour de la position du pire éléphant/robot.	52
5.5	Organigramme du mode de fonctionnement de l'approche EHO.	54
5.6	Représentation de la méthode de mise à jours du Pbest d'un éléphant.	55

5.7	Représentation de la mise à jour de la vitesse d'un éléphant dans l'approche EWSA.	56
5.8	Organigramme du mode de fonctionnement de l'approche EWSA.	60
6.1	EWSA à l'itération 4	64
6.2	EWSA à l'itération 8	64
6.3	EWSA à l'itération 16	64
6.4	Représentation de l'organisation des expérimentations sur la portée des cibles.	65
6.5	Représentation de l'organisation des expérimentations sur la taille des environnements.	66
6.6	Représentation de l'organisation des expérimentations sur le nombre de cible.	66
6.7	Comparaison de la variation du taux de réussite des algorithmes selon la portée des cibles.	67
6.8	Comparaison de la variation du nombre d'itérations des algorithmes selon la portée de la cible (sans obstacles).	68
6.9	Comparaison de la variation du temps d'exécution des algorithmes selon la portée de la cible (sans obstacles).	68
6.10	Comparaison de la variation du nombre d'itérations des algorithmes selon la portée des cibles (sans obstacles).	69
6.11	Comparaison de la variation du temps d'exécution des algorithmes selon la portée des cibles (sans obstacles).	69
6.12	Comparaison de la variation du nombre d'itérations des algorithmes selon la portée de la cible (avec obstacles).	69
6.13	Comparaison de la variation du temps d'exécution des algorithmes selon la portée de la cible (avec obstacles).	69
6.14	Comparaison de la variation du nombre d'itérations des algorithmes selon la portée des cibles (avec obstacles).	70
6.15	Comparaison de la variation du temps d'exécution des algorithmes selon la portée des cibles. (avec obstacles)	70
6.16	Comparaison de la variation du nombre d'itérations des algorithmes selon la portée de la cible (complexe).	71
6.17	Comparaison de la variation du temps d'exécution des algorithmes selon la portée de la cible (complexe).	71
6.18	Comparaison de la variation du nombre d'itérations des algorithmes selon la portée des cibles (complexe).	71
6.19	Comparaison de la variation du temps d'exécution des algorithmes selon la portée des cibles (complexe).	71
6.20	Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (sans obstacles).	72
6.21	Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (sans obstacles).	73
6.22	Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (avec obstacles).	73
6.23	Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (avec obstacles).	73
6.24	Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (complexe).	74
6.25	Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (complexe).	74

6.26 Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (sans obstacles).	75
6.27 Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (sans obstacles).	75
6.28 Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (sans obstacles).	76
6.29 Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (sans obstacles).	76
6.30 Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (avec obstacles).	77
6.31 Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (avec obstacles).	77
6.32 Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (avec obstacles).	77
6.33 Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (avec obstacles).	77
6.34 Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (complexe).	78
6.35 Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (complexe).	78
6.36 Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (complexe).	79
6.37 Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (complexe).	79
6.38 Comparaison de la variation du taux de réussite des algorithmes selon le nombre de cibles.	80
6.39 Comparaison de la variation du nombre d'itérations des algorithmes selon le nombre de cible (sans obstacles).	80
6.40 Comparaison de la variation du temps d'exécution des algorithmes selon le nombre de cible (sans obstacles).	80
6.41 Comparaison de la variation du nombre d'itérations des algorithmes selon le nombre de cible (avec obstacles).	81
6.42 Comparaison de la variation du temps d'exécution des algorithmes selon le nombre de cible (avec obstacles).	81
6.43 Comparaison de la variation du nombre d'itérations des algorithmes selon le nombre de cible (complexe).	82
6.44 Comparaison de la variation du temps d'exécution des algorithmes selon le nombre de cible (complexe).	82
6.45 Schéma de communication du système multi-threads.	84
6.46 Présentation de l'interface	84
6.47 Présentation de la section Métaheuristiques	85
6.48 Les types d'environnements possibles	86
6.49 Affichage de l'environnement selon les préférences insérées	86
6.50 Auto-paramétrage avec l'algorithme génétique	87
6.51 Menu de paramétrage	87
6.52 Menu de paramétrage manuel pour BSO	87
6.53 Informations durant l'exécution de BSO	87
6.54 Liste des paramètres pour BSO	87
6.55 Le slider	88
6.56 Exemple d'exécution de BSO.	88

Listes des tableaux

2.1	Analogie entre les caractéristiques des essaims d'abeilles et BSO.	12
2.2	Analogie entre les caractéristiques des éléphants et EHO.	16
2.3	Analogie entre les caractéristiques des éléphants et EWSA.	20
2.4	Analogie entre les caractéristiques de l'ADN et GA.	24
3.1	Contraintes sur les paramètres des méta-heuristiques.	38

Liste des abréviations

PSO	Particle Swarm Optimization
GA	Genetic Algorithm
ACO	Ant Colony Optimisation
BA	Bees Algorithm
ABC	Artificial Bee Colony Optimisation
BFO	Bacterial Foraging Optimisation
GSO	Glowworm Swarm Optimisation
FA	Firefly Algorithm
DW	Dinamic Window
SAT	Satisfaction
FKD	Filtre de Kalman Distribué
FOA	Fruit Fly Optimization Algorithm
MSCM	Multi-Scale Cooperative Mutation
BSO	Bee Swarm Optimization
EHO	Elephant Herding Optimization
ESWSA	Elephant Swarm Water Search Algorithm
GA	Génétique Algorithme
Méta	Métaheuristique
Algo	Algorithme

Introduction générale

Une vague d'avancées technologiques a submergé le monde durant cette dernière décennie, l'intelligence artificielle est au centre de toutes les innovations étant par conséquent le sujet le plus actuel et un critère d'évaluation du développement des pays du monde.

Cet intérêt croissant porté à des villes intelligentes de voir le jour ainsi que des voitures autonomes ou encore des robots sociaux comme Sophia [12]. Cette dernière est un robot humanoïde créé à Hongkong par la société "Hanson Robotics" qui a comme objectif d'aider les personnes âgées.

Le développement de l'intelligence artificielle résulte des besoins obsessionnelles et nécessaires en gain de temps, d'efforts et de mains d'œuvres humaines dans l'accomplissement de tâches quotidiennes tout en gardant une haute performance et efficacité.

C'est pourquoi les chercheurs, encouragés par les investisseurs, se sont intéressés de près à proposer des solutions automatisant certaines tâches quotidiennes coûteuses en temps ou jugées dangereuses pour l'homme.

Les systèmes multi-robots font partie des innovations émergentes, ces derniers ont la main mise sur des domaines d'application des plus intéressants, allant de l'usage domestique au domaine militaire, passant par l'industrie ou encore le domaine de la santé. Ces multiples applications mettent l'accent sur l'adaptabilité des robots qui sont capables de résoudre divers problèmes [13].

Notons que la mobilité des robots est l'une de leurs capacités la plus convoitée, que ce soit dans l'assemblage et l'entreposage, le transport industriel, les équipements de nettoyage du sol [14], l'exploration de planète, la recherche et le secourisme des victimes dans une zone sinistrée ou encore la localisation de mines ou de bombes [15].

Le succès flagrant qu'ont connu les systèmes multi-robots, a fait d'eux un pôle de propositions d'amélioration et de développements particulièrement au niveau de la coopération entre robots pour une meilleure planification, cherchant à les rendre aussi intelligent que l'homme en simulant les interactions et processus de raisonnement humains. D'ailleurs une équipe de robots, bien qu'ayant un comportement simple ou n'étant pas puissant individuellement peut compenser et surpasser ses limites grâce à la coopération [16].

Parmi la panoplie de problèmes dans lesquels les systèmes multi-robots interviennent, on trouve les problèmes de recherche, détection et suivi. Nous nous intéressons particulièrement à la recherche ou détection de cibles dans des environnements inconnus et complexes, qui justement est au centre d'attention ces dernières années. À première vue, cela s'explique par la généralisation qu'offre le mot cible qui peut avoir multiples facettes, par exemple des personnes perdues ou des mines, de l'or, des puits de pétrole et bien plus encore. Mais il s'agit surtout du facteur "*complexité*" du problème, dont il découle un certain nombre de problématiques. Parmi lesquelles, on trouve le choix de la structure de coopération des robots afin de réduire les efforts et temps de recherche tout en augmentant les chances de succès, mais aussi en évitant la recherche répétitive de mêmes zones.

Puis vient la nécessité de concevoir un système d'évitement d'obstacles et de navigation fiable, pour lesquels des déplacements sans collisions au sein d'environnements

complexes est garantie. Enfin, les robots manquent dans la plupart des cas d'informations sur l'environnement de recherche, ils doivent alors adapter des stratégies intelligentes défiant ainsi ces contraintes [15].

À travers ce travail, nous chercherons à proposer de nouvelles méthodes de résolution pour la détection de cibles en tentant d'apporter des améliorations au niveau des performances. Notre contribution comportera une adaptation d'un système multi-robots aux quatre approches inspirées de l'intelligence en essaim, à savoir *BSO* et *Multi-BSO* qui sont basées sur le comportement des abeilles lors de la recherche du pollen, *EHO* et *ESWSA* qui sont inspirées de l'intelligence des éléphants en termes de communication et planification lors de la recherche d'eau et de nourriture. Un robot se comportera comme une abeille ou un éléphant selon l'approche choisie.

Ces quatre approches devront être adaptées à une même modélisation du problème afin de permettre une comparaison effective des résultats obtenus. Pour cette même raison, la stratégie d'évitement d'obstacles sera exploitée et intégrée pour toutes nos mét-heuristiques.

Ce mémoire compte quatre chapitres organisés comme suit :

D'abord, une synthèse sur les maintes variantes du problème et tous les paramètres influant la direction qu'il peut prendre est présentée dans le chapitre 1. Les algorithmes d'intelligence en essaim choisis pour l'élaboration de nos solutions sont décrits dans le chapitre 2. Le chapitre 3 fera l'objet de la modélisation de notre problème et du mécanisme de nos approches pour la recherche de cibles tout en contournant les obstacles. L'efficacité des approches proposées sera validée à travers plusieurs expérimentations dont les résultats seront exhibés au chapitre 4. Enfin nous terminerons par une conclusion sur l'ensemble de ce travail ainsi que des perspectives.

Chapter 1

Chapitre 1 : État de l'art

1.1 Introduction

La détection et le suivi de cibles englobent une large variété de problèmes décisionnels, tels que la surveillance, la recherche, les patrouilles, l'observation et la poursuite-évasion. Compte tenu du besoin croissant en effectif, en temps et en minimisation des coûts, de nombreuses applications ont été développées notamment dans des domaines comme le commerce et la défense.

Dans ce chapitre, nous nous attelons à fournir une synthèse sur les différents travaux inhérents à la recherche de multiples cibles dans un environnement inconnu et complexe. Au préalable, nous devons introduire certaines définitions et techniques propres à la nature des environnements auxquels nous aurons à faire face, d'autres concernant les méthodes de résolution. On finira par une conclusion dans laquelle nous situerons clairement la problématique que nous traitons.

1.2 Problématique

Pourquoi s'intéresser à la détection de cibles ?

La détection de cibles s'étale sur divers domaines, cela s'explique par l'amélioration et les bénéfices apportés par la résolution de ce genre de problème dans le quotidien.

Nous nous intéressons particulièrement à la recherche de mines anti-personnelles et bombes ainsi que la recherche de sources radioactives, car rien n'est plus important que la sécurité et la mise hors de danger de vies humaines.

Pour cela, nous aurons recours à des méthodes de résolution basées sur l'intelligence en essaim, car ce sont les méthodes qui s'y prêtent le mieux et c'est justement ces méthodes qui ont dernièrement permis maintes avancées.

De ce fait, nous aurons besoin de développer un système multi-robots où chaque robot se comportera comme une particule de l'essaim.

L'environnement inconnu est considéré en 2D, chaque robot n'a connaissance que du nombre de cibles recherchées, les bordures ou frontières de l'environnement de recherche ainsi que sa position initiale (position avant le début de la recherche).

Aussi l'environnement est complexe et englobe plusieurs obstacles, entravant son exploration.

Les positions des obstacles sont inconnues, c'est pourquoi les robots doivent être munis de capteurs adaptés à leur stratégie d'évitement d'obstacles. Les cibles (mines / sources de radiation) dont les positions sont inconnues, émettent un signal que les robots sont capables de réceptionner via leurs capteurs.

La validation du système sera effectuée par des expérimentations sur des environnements dont les positions des obstacles, cibles et des robots sont générées aléatoirement. Cela conformément à la procédure présentée dans des articles récents.

1.3 Paramètres du problème

Plusieurs paramètres entrent en jeu concernant le problème de recherche de cibles, donnant lieu à une arborescence de sous-classes de problèmes (voir figure 1.1), qui se diffèrent par les techniques de résolutions adaptées.

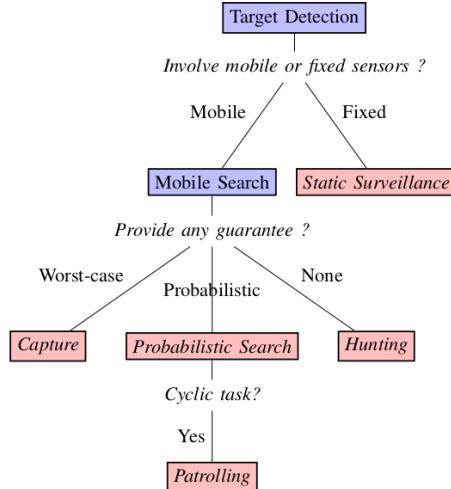


Figure 1.1: Schéma résumant les classes de problèmes de détection de cibles [2].

Nous distinguons quatre types de problèmes :

- **La surveillance statique** implique l'utilisation de robots fixes dans un environnement connu afin d'entièrement le couvrir pour y détecter les cibles [2].
- **La capture** a pour but de capturer toutes les cibles présentes dans un environnement connu. Seules les positions initiales des cibles ne sont pas connues par le(s) poursuiveur(s) [2]. La figure 1.2 illustre un exemple de capture.

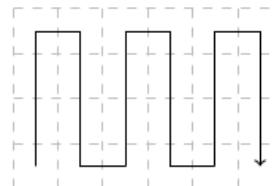


Figure 1.2: Exemple de capture [2].

- **La patrouille** a une formulation proche de celle de capture, mais avec un aspect cyclique. Autrement dit la zone n'est pas couverte qu'une seule fois, mais plusieurs fois [2], comme le montre la figure 1.3.

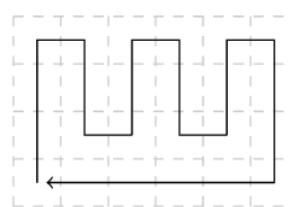


Figure 1.3: Exemple de patrouille [2].

- **La chasse** s'attaque à la détection de cibles sans aucune garantie. C'est-à-dire qu'on suppose qu'on ne connaît ni les positions initiales des cibles ni l'environnement de recherche [2].

À partir de la formulation de la problématique et des classes de problèmes décrits ci-dessus, on peut clairement situer notre contribution dans le cadre des problèmes de **Chasse**.

Dans ce qui suit nous décrivons les paramètres jugés les plus pertinents :

1.3.1 Les systèmes multi-robots (SMRs)

Définition

Un système multi-robots est un groupe de robots mobiles autonomes ayant un objectif ou un ensemble de tâches communes, ces robots ont la capacité de communiquer et se coordonner afin d'atteindre leur objectif [17].

Les robots mobiles se déplacent durant la recherche selon une stratégie visant à aboutir à des solutions optimales.

Il est important de souligner que le mode de mobilité des robots influence fortement la résolution du problème (vision, vitesse, agilité du mouvement) [1].

Caractéristiques des SMRs [1]

Les systèmes multi-robots se caractérisent par multiples aspects qui font leur force, à commencer par l'exécution des tâches en parallèle, ainsi qu'une couverture d'une plage assez importante de l'espace de recherche. D'autres particularités des SMRs spécifiques aux essaims de robots existent, telles que:

- *La robustesse*, qui offre une tolérance aux pannes.
- *la flexibilité* permet une rapide adaptation aux nouvelles exigences et différents environnements (favorisée par la redondance et la simplicité des comportements des robots).
- *l'évolutivité* qui est la capacité à fonctionner avec un nombre plus grand ou plus petit de robots sans impact considérable sur la performance.

Système de communication [2]

Deux types de systèmes sont les plus répandus: centralisés et décentralisés, ils sont choisis selon le besoin et les critères recherchés.

- **Les systèmes centralisés**, sont plus aptes à fournir une optimalité globale. Cependant, ils sont souvent confrontés à de fortes contraintes du monde réel, d'où la nécessité d'une connectivité totale dans de nombreuses situations.
- **Les systèmes décentralisés**, quant à eux fournissent des solutions jugées sous-optimales. Par contre ils sont plus robustes et plus flexibles de par leur adaptation aux environnements dynamiques ce qui leur permet d'aboutir à des performances intéressantes.

Comportement des robots [2]

Les robots peuvent s'approprier deux types de comportements, soient:

- **Indépendant**, chaque robot est responsable d'un certain nombre de tâches à exécuter.
- **Coopératif**, chaque décision et action sont diffusées à toute l'équipe. Ce mode coopératif peut être explicite, consistant à influencer un robot par une communication directe, ou bien implicite, qui est une prise de décision indépendante selon les informations individuellement recueillies.

1.3.2 Les capteurs

Selon l'environnement à explorer, le besoin en types de capteurs diffère, que ce soit pour reconnaître les cibles, détecter les obstacles afin de les éviter ou reconnaître les autres robots.

Parmi la panoplie des capteurs les plus utilisés : les caméras, les infrarouges, les capteurs lasers (distance), les capteurs à ultrason, la kinect, les capteurs de pression, les capteurs de gaz, GPS, ...etc [18]. Le choix des capteurs consiste à trouver le meilleur compromis entre coût et efficacité selon les contraintes imposées.

1.3.3 Les cibles

Le nombre de cibles est un paramètre fixé pour un environnement donné. Les problèmes **mono-cible** consistent à trouver la position d'une cible particulière dans l'environnement. En revanche les problèmes **multi-cibles** visent à maximiser le nombre de cibles trouvées.

Pour le problème de recherche ou détection, les cibles possèdent des positions inchangées dans l'environnement durant toute la recherche [1].

Aussi il est toujours supposé que les cibles émettent des radiations, sons, ondes, lumières, odeurs ou autres, qui sont maximales aux alentours de la cible s'estompant graduellement en s'éloignant de celle-ci. L'étendue du champ d'émission de ces cibles est aussi un paramètre qui influe directement l'efficacité des algorithmes de résolution.

1.3.4 L'environnement [2]

L'environnement est l'espace de recherche où évoluent les cibles et interviennent les robots comme fouilleurs. Il est décrit comme **connu** lorsqu'une carte est mise à disposition, et **inconnu** lorsqu'on manque d'informations globales n'ayant connaissance que de ses bordures. Ces environnements peuvent être des:

- **Environnements simples**, c'est-à-dire sans obstacles ne contenant aucune contrainte susceptible de bloquer la vue ou le mobilité des robots.
- **Environnements avec obstacles ou complexes**, Ceux-ci contenant des obstacles, contraignants ainsi les robots dans leurs déplacements et leurs perceptions, les poussant parfois à changer de trajectoire.

1.3.5 Les obstacles

Les obstacles sont des contraintes supplémentaires dans le processus de recherche de cibles. Divers types d'obstacles existent, citons les obstacles fixes et mobiles, les obstacles

franchissables mais entravant la vue ou encore les obstacles empêchant uniquement le passage sans obstruer la visibilité des robots, ... etc [2].

La présence d'obstacles dans les environnements de recherche impose aux méthodes de résolution une prise en considération d'une stratégie d'évitement d'obstacles, pour cela nombreuses sont les stratégies existantes.

Elles s'appuient sur les informations fournies par les capteurs embarqués afin d'orienter les robots à chaque laps de temps pour une réaction en temps réel [8].

1.4 Travaux connexes

Plusieurs approches ont été exploitées pour la résolution du problème de détection de cibles dans des environnements inconnus, nous citons à titre d'exemple:

1.4.1 Les algorithmes anti-regroupements [3]

Souvent appliqués aux problèmes de type "chasse", ces algorithmes imitent le comportement social des animaux solitaires (tigre, araignée,...). Ils se basent sur trois principales règles, soient:

- La prévention des collisions : en s'éloignant des obstacles les plus proches et en respectant une distance de sécurité.
- La décentralisation : en tentant de se séparer de ses voisins.
- L'égoïsme : si aucune des deux situations précédentes ne se produit, chaque robot choisit la direction permettant de maximiser ses propres gains.

1.4.2 Approche des champs de potentiel [2]

Dans cette approche on considère deux vecteurs de forces locales qui influencent les mouvements des robots telles que, les cibles proches émettent des forces attractives vers elles tandis que les robots proches émettent des forces répulsives éloignant ainsi les autres robots. L'algorithme de Parker pour le contrôle distribué A-CMOMMT est inspiré des champs de potentiels [19].

1.4.3 Approches basées intelligence en essaim

L'intelligence en essaim et les méta-heuristiques ont été très exploitées pour ce type de problème grâce à la facilité des opérations stochastiques qui les distingue des autres techniques. Leur force majeure est l'inspiration des comportements d'espèces ou phénomènes naturels [20].

PSO, GA, ACO, BA, ABC, BFO, GSO et FA sont les techniques de méta-heuristiques les plus utilisées revenants souvent dans la littérature avec des améliorations et contributions apportées par les auteurs [1].

Quant aux travaux récents on trouve:

BSO (Bee Swarm Optimization) (2013)

Une variante de l'algorithme Bee Swarm Optimization inspirée du comportement des abeilles dont les auteurs sont Hannaneh Najd Ataei, Koorush Ziarati et Mohammad Eghesad [21] comporte trois types d'abeilles (abeilles butineuses, abeilles éclaireuses, abeilles

spectatrices) chacune possède sa propre fonction de mise à jour de ses positions.

Cette variante a montré de meilleures performances que la version classique de PSO. Suite à des expérimentations sur un ensemble d'environnements aléatoires il a été prouvé que cette approche de BSO peut être 50,6% plus efficace que PSO.

Mais elle n'a été testée que sur des environnements mono-cibles. De plus la fonction objectif utilisée s'appuie sur la distance euclidienne entre les robots et la cible, or que les positions des cibles sont supposées inconnues.

A-RPSO

A-RPSO (Adaptative Robotic Particule Swarm Optimization) est une technique basée essaim proposée par les auteurs M. Dadgar, et al., où chaque particule représente un robot. La contribution de l'auteur par rapport à un PSO classique est l'ajout d'une stratégie d'évitement d'obstacles $c_3r_3(p_i^t - x_i^t)$ incluse dans l'équation de déplacement 1.1, ainsi qu'une mise à jour des paramètres classiquement constants comme inertia weight de façon dynamique dite adaptative.[15].

$$v_i^{t+1} = w_i^t v_i^t + c_1 r_1 (pbest - x_i^t) + c_2 r_2 (gbest - x_i^t) + c_3 r_3 (p_i^t - x_i^t) \quad (1.1)$$

Ces modifications ont eu comme objectif d'éviter les optimaux locaux et mieux contrôler et adapter l'allure des robots lors du rapprochement des cibles et des obstacles. La technique a été testée sur un simulateur 2D créé par les auteurs qui reste proche d'une modélisation d'une grille virtuelle 2D.

MFSO (Multi-swarm hybrid FOA-PSO)

Un nouvel algorithme multi-essaim hybride (MFPSO) basé sur FOA (Fruit Fly Optimization Algorithm) et PSO (Particule Swarm Optimization) pour la recherche d'une cible dans des environnements inconnus a été présenté par les auteurs, Hongwei Tang, Wei Sun, Hongshan Yu, Anping Lin, Min Xue et Yuxue Song [14]. Cette nouvelle mét-heuristique se singularise par les avantages suivants:

- Parallélisme grâce à un coefficient adaptatif.
- Présence de stratégies contournant la convergence prématuée grâce à l'indépendance des essaims de robots.
- Mécanisme d'évacuation (MSCM) comme stratégie d'évitement d'obstacles.

Les nombreuses expériences effectuées pour le mode mono-cible sur MFPSO prouvent qu'il surpassé de manière significative les autres approches (dont A-RPSO et RPSO), en termes de nombre d'itérations et taux de réussite dans la plupart des cas.

Par contre, le champ d'émission de la cible est inconnu, or comme mentionné plus haut, ce paramètre influence la difficulté du problème.

Les tests effectués comportent le cas du mono-cible seulement, ce qui implique que le comportement et l'efficacité de cet algorithme dans des environnements multi-cibles sont inconnus.

Aussi, le champ de vue des robots a été fixé par les auteurs à une valeur de dix pixels durant toutes les expérimentations.

Enfin, les auteurs utilisent une fonction objectif qui manipule la distance $Dist(x_i^t)$ entre les robots et la cible. Cependant la position de la cible est supposée inconnue. C'est pourquoi cette distance au lieu d'être directement utilisée comme fonction objectif elle a servi à calculer une estimation des signaux émis par la cible.

1.5 Conclusion

Ce chapitre nous a permis de décortiquer les maintes variantes des problèmes de détection de cibles et de clairement situer dans quelle branche notre travail interviendra. Tous les paramètres nécessaires pour clairement définir notre problème et aboutir à une bonne modélisation, ont été explicitement exposés. Notre méthode de résolution est basée en intelligence en essaim, d'où la nécessité d'éclaircir certains aspects incontournables relatifs aux approches choisies dans le chapitre suivant.

Chapter 2

Chapitre 2 : Intelligence en essaim : BSO, EHO, EWSA et GA

2.1 Introduction

Dans ce chapitre nous allons présenter les outils intelligents de résolution pour notre problématique. Ces derniers consistent en des méta-heuristiques inspirées de l'intelligence collective des animaux et insectes, autrement dit l'intelligence en essaim.

Nous commencerons par introduire les algorithmes basés intelligence en essaim et leurs principaux paramètres, suivis d'un décryptage des trois approches choisies pour notre étude, à savoir :

- Bee Swarm Optimization (BSO) qui est un algorithme de résolution s'inspirant de l'intelligence collective du comportement des abeilles lors de la recherche de nourriture. BSO a fait ses preuves face à de nombreux problèmes d'optimisation combinatoire (le problème de satisfiabilité SAT, le problème du voyageur de commerce, ...etc.).
- Elephant Herding Optimisation (EHO) et Elephant Swarm Water Search Algorithm (EWSA), tous deux des algorithmes inspirés du comportement des éléphants en groupe.
- On verra aussi l'algorithme génétique (GA) dans sa version GA incrémental.

On clôturera ce chapitre par les motivations qui nous ont poussées à choisir ces algorithmes intelligents comme méthodes de résolution pour la détection de cibles.

2.2 Les algorithmes basées intelligence en essaim

Swarm Intelligence ou l'intelligence en essaim étudie le comportement coopératif entre les espèces naturelles. C'est un domaine prometteur de l'intelligence artificielle dont le principe de base repose sur l'observation par la nature des phénomènes intelligents et surtout des comportements de groupe chez les animaux.

Une approche ou méthode basée intelligence en essaim est également considérée comme une méta-heuristique. Elle suit un schéma d'algorithme évolutif, dicté par la simulation de l'évolution des espèces naturelles [22].

Les méta-heuristiques sont des approches génériques qui visent à proposer des solutions à des problèmes complexes, en employant une certaine stratégie d'exploration de l'espace de recherche appelée **espace des solutions**. Ce dernier comporte toutes les solutions possibles au problème. Ces **solutions** sont ensuite filtrées grâce à une **fonction objectif** permettant de les ordonner selon leur optimalité [22].

La figure 2.1 montre qu'il existe deux catégories principales de méta-heuristiques. Les méta-heuristiques basées sur une solution unique, telles que, la recherche tabou TS (Tabu Search) [23], recuit simulé, la recherche locale guidée GLS (Guided Local Search) [24], ...etc, et les méta-heuristiques basées population comme l'algorithme génétique GA [25, 26, 27] ou une de ses extensions MA (Memetic Algorithm) [28, 26], la recherche par dispersion SS (Scatter Search) [27], l'algorithme Bat [29, 30, 31], les colonies de fourmis ACS (Ant Colony Optimization) [32, 29], PSO (Particle Swarm Optimization) [33], l'algorithme des lucioles FFA (Firefly algorithm) [34], l'essaim d'abeilles à savoir BSO (Bee Swarm Optimization) [35, 36, 37, 38], ABSO (Advanced Bee Swarm Optimization) [39], ou encore ABC (Artificial Bee Colony) [40] etc.

Les algorithmes basés intelligence en essaim peuvent être vus comme des méta-heuristiques basées population [22].

Ces nombreux travaux ont été développés au sein du laboratoire de recherche en l'intelligence artificielle LRIA¹.

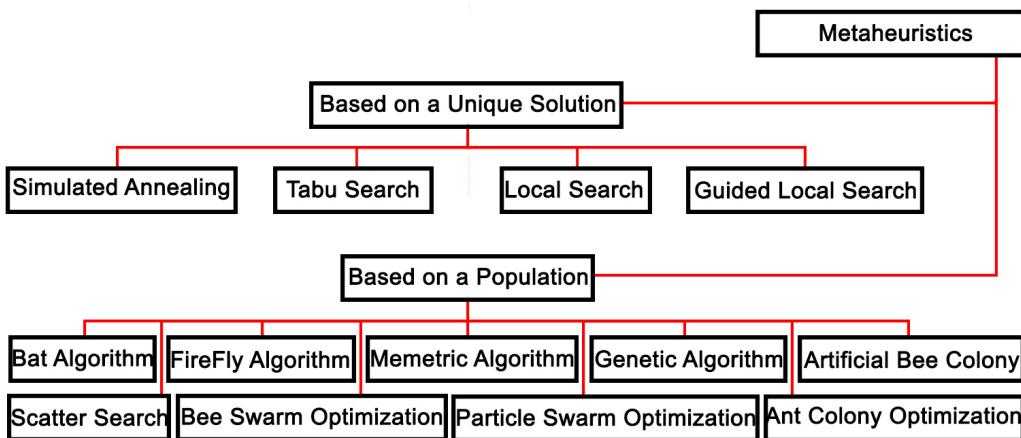


Figure 2.1: Classification des méta-heuristiques

Étant donné l'efficacité et la simplicité des méta-heuristiques, on va s'intéresser de plus près à BSO, EHO, EWSA et GA qui sont des méta-heuristiques basées population ayant connu un large intérêt et succès.

2.3 Bee Swarm Optimization [4]

2.3.1 Inspiration des abeilles

De nombreuses études biologiques se sont intéressées aux comportements et à la psychologie des abeilles afin de décortiquer leurs méthodologies et stratégies de recherche du nectar ainsi que leur façon de communiquer les informations concernant leurs trouvailles.

L'une des expériences les plus connues sur les abeilles, est celle de Seeley, Camazine et Sneyd en 1991 [41], elle a abouti sur les quelques caractéristiques suivantes :

- la catégorie d'abeilles "travailleuses" est celle qui se charge de trouver la nourriture ("Scout bee"). Les abeilles reviennent à la ruche en ramenant un peu de leurs récoltes.

¹<https://www.lria.usthb.dz/>

- Les abeilles ont tendance à privilégier les régions qui regorgent de nourriture aussi lointaines soient elles aux zones moins riches en pollen, mais à proximité de la ruche.
- L'abeille ayant trouvé la meilleure région en termes de nourriture effectuera la danse la plus rigoureuse, ce qui guidera l'essaim vers la meilleure région.
- La danse (mouvement circulaire) est le moyen de communication entre les abeilles, elle englobe les informations concernant la quantité, la direction et la distance entre la ruche et la région où se trouve la nourriture.

2.3.2 Analogie entre le comportement des abeilles et BSO

BSO est entièrement inspirée du comportement des abeilles pour la recherche du pollen, ci-dessous le tableau 2.1 fait l'analogie entre le comportement des abeilles travailleuses et leurs représentations simplifiées dans BSO :

Essaim d'abeilles	BSO
Une abeille éclaireuse est envoyée dans un champ de fleurs, et se positionne sur une fleur aléatoirement.	Une solution est générée aléatoirement à partir de l'espace des solutions, elle est ensuite affectée à l'abeille de référence.
Les abeilles s'orientent vers les champs de fleurs à la recherche de quantité de nourriture (pollen).	Les abeilles artificielles se déplacent dans l'espace des solutions à la recherche de bonnes solutions.
Les abeilles se dispersent autour de la zone indiquée par l'abeille éclaireuse pour couvrir au mieux le champ de fleurs.	La diversification est basée sur le Flip. Ce dernier permet la génération d'un certain nombre de solutions à partir de la solution de référence.
Chaque abeille procède à une recherche locale dans sa zone d'atterrissement.	Chaque abeille porteuse d'une solution effectue une recherche locale, ce processus est appelé " <i>intensification</i> ".
Chaque zone de fleurs est évaluée selon la quantité de pollen qui s'y trouve.	Chaque solution est évaluée selon la fonction objectif.
Chaque abeille se rappelle de la fleur source ayant la plus grande quantité de pollen ainsi que l'endroit où elle se trouve. Elle se met à danser juste au-dessus pour en informer l'essaim.	Chaque abeille sauvegarde sa meilleure solution locale dans une table appelée <i>Table Dance</i> .
La vigueur de la danse de l'abeille est proportionnelle à la richesse et quantité de nourriture trouvée.	La sélection d'une solution de la table <i>Dance</i> se base sur un critère de qualité. Toutes les abeilles sont mises au courant de la meilleure solution actuelle.
En cas de recherche non-fructueuse, les abeilles explorent d'autres zones distantes	en cas de stagnation de la recherche, la solution la plus diverse de la table <i>Dance</i> est sélectionnée comme solution de référence.

Table 2.1: Analogie entre les caractéristiques des essaims d'abeilles et BSO.

2.3.3 Paramètres et fonctions

BSO est caractérisée par plusieurs paramètres empiriques, qui sont :

- **nbrBee** : le nombre d'abeilles de l'essaim.
- **Flip** : la fréquence d'inversement.
- **MaxChance** : le nombre de chances maximal.
- **MaxItération** : le nombre d'itérations maximal.

Sref: La solution de référence est initialisée aléatoirement au début de la recherche, puis mise à jour selon les étapes de BSO, par la meilleure solution en termes de qualité ou de diversité.

Flip: C'est un paramètre qui détermine le nombre de variables à inverser au niveau de la solution de référence (*Sref*) afin d'obtenir de nouvelles solutions. $\frac{n}{flip}$ représente ainsi la distance qui sépare les nouvelles solutions de *Sref*, *n* étant la taille de la solution.

Détermination des zones de recherche: l'application du *Flip* sur une solution de référence génère un ensemble de solutions équidistantes de *Sref*, formant un cercle autour d'elle.

Recherche locale: Chaque abeille effectue une recherche locale à partir de la solution qui lui est affectée afin d'évaluer les solutions voisines en ne gardant que la meilleure.

MaxChance: Chaque solution possède un nombre de chances maximal pour être choisie comme solution de référence à chaque itération.

Table dance: La table "Dance" est le répertoire commun à toutes les abeilles de l'essaim pour y inscrire leurs meilleures solutions locales.

2.3.4 Pseudo-code BSO

```

1 Algorithme : BSO
2
3   Input : MaxIteration, Flip, Maxchances, Optimale, nombred'abeilles
4   Output : meilleureSolution : la meilleure solution trouvée
5   /* génération aléatoire de la solution de référence. */ 
6   Sref ← solutionAléatoire()
7   meilleureSolution ← Sref
8   /* Tant que le nombre d'itérations maximal n'est pas atteint et la
9    solution optimale n'est pas trouvée réitérer */ 
10  while ¬(MaxIteration or Optimale) do
11    /* Insertion de la solution de référence dans la liste Tabou */
12    Insertion(listeTabou,Sref)
13    /* Affectation des zones de recherche aux abeilles */
14    abeilles ← DeterminationDesZonesDeRecherche (Sref)
15    foreach abeille ∈ abeilles do
16      /* Effectuation d'une recherche locale */
17      solutionLocale ← rechercheLocale (abeille)
18      /* Evaluation et insertion de la meilleure solution locale dans
19       la table Dance. */
20      Insertion(Dance,MeilleuresolutionLocale)
21    end
22    /* Sélection de la meilleur solution de la table Dance. */
23    Sref ← MeilleureDeDance (Dance)
24    if sRef > meilleure then
25      /* Sauvegarde de la nouvelle meilleure solution. */
26      meilleureSolution ← sRef
27    end
28  end
29  return meilleureSolution

```

Algorithme 1 : BSO Générique

```

1 Fonction : DéterminationDesZonesDeRecherche
2
3   Input : Flip : fréquence d'inversement
4   Output : ensemble_K_solutions : ensemble de K solutions
5   h ← 0
6   while taille(espace_recherche) < k and h < Flip do
7     s ← Sref; p ← 0
8     /* Inversion partielle de la solution, à une fréquence : flip */
9     repeat
10    s ← Inverser (s,s[Flip * p + h])
11    p ← p + 1
12    until Flip * p + h > n
13    /* Insertion de la solution générée dans la liste de solution */
14    ensemble_K_solutions ← ensemble_K_solution ∪ {s}
15    h ← h + 1
16  end
17  return ensemble_K_solutions

```

Fonction DéterminationDesZonesDeRecherche

2.4 EHO & EWSA : Deux algorithmes basés essaim d'éléphants

2.4.1 Inspiration des éléphants

Les éléphants sont des mammifères sociaux qui présentent des structures sociales complexes ayant une multitude de particularités surprenantes.

Dans un premier temps, on se contentera d'exhiber les caractéristiques individuelles suivantes [6] :

- Bonne mémoire.
- Font preuve d'une intelligence avancée (reconnaissance de soi, la conscience de soi).
- Capacité d'apprendre et de distinguer plusieurs discriminations visuelles et acoustiques.
- Capacité à utiliser et manipuler des outils de la vie réelle.
- Possession de systèmes de détection et de communication très avancés.

Quant aux caractéristiques de groupe, nous citons [5] :

- Un groupe d'éléphants est composé de plusieurs clans.
- Un clan est dirigé par une matriarche (souvent l'éléphant le plus vieux du troupeau).
- Les femelles préfèrent vivre dans des groupes familiaux (clan).
- Les éléphants mâles ont tendance à quitter leur groupe familial en grandissant.
- Les éléphants mâles loin de leur groupe familial peuvent rester en contact avec les éléphants de leur clan par le biais de vibrations basses fréquences.

Méthodes de communication [6]:

Les éléphants utilisent leurs sens de l'ouïe, de l'odorat, de la vision, du toucher et une capacité exceptionnelle à détecter les vibrations pour communiquer entre eux ainsi qu'avec les autres espèces. Il existe deux types courants de communication entre éléphants:

1. Communication à longue distance: (jusqu'à 10-12 km)

-*Les communications sismiques*, causées par le grondement et le mouvement des éléphants. Leur réception se fait à travers leurs mécano-récepteurs dans les orteils ou les pieds et la pointe de leurs trompes.

-*Les communications sonores* produisent une gamme de signaux sonores (infrasons) que leurs trompes peuvent amplifier. Leurs oreilles font office de parabole pour la réception des sons de basses fréquences d'autres éléphants lointains.

-*Les communications chimiques (olfactives)*, ils utilisent leurs trompes pour sentir l'air, ou pour explorer le sol et les arbres, ainsi que pour renifler d'autres éléphants.

2. Communication à courte distance:

-*Les communications visuelles*, la vision des éléphants atteint une portée maximale de 46-100 m. Ils utilisent la tête, les yeux, la bouche, les oreilles, la trompe et tout le corps pour l'échange de messages.

-*Les communications tactiles*, Les éléphants sont des animaux extrêmement tactiles. Ils communiquent par le toucher en utilisant leurs trompes, leurs défenses, leurs pieds, leurs queues, ...etc

2.4.2 EHO : Elephant Herding Optimization

Analogie entre les éléphants et EHO [5]

La méta-heuristique EHO simule l'évolution du mouvement des éléphants. Elle représente un comportement simplifié des groupes d'éléphants se basant sur les règles idéalisées présentées dans le tableau qui suit :

Clans d'éléphants	EHO
Les éléphants parcouruent la nature à la recherche de meilleures sources de nourriture et d'eau	Les éléphants artificiels se déplacent dans l'espace des solutions cherchant la meilleure solution globale au problème à traiter.
La quantité de nourriture et d'eau permet aux éléphants d'évaluer la qualité de l'environnement	La qualité de la solution est évaluée à travers la fonction objectif.
Les éléphants vivent en clans où chaque clan est composé de plusieurs éléphants.	Plusieurs clans (nClan) sont créés avec un nombre fixe d'éléphants N , une solution est affectée à chaque éléphant.
Les éléphants de chaque clan vivent sous la direction d'une <i>matriarche</i> qui guide leurs déplacements.	L'éléphant possédant la meilleure solution du clan (meilleure locale) guide le clan.
Un nombre déterminé d'éléphants mâles quittera leur groupe familial et vivra solitairement loin du groupe d'éléphants principal à chaque génération.	Le pire éléphant en termes d'évaluation, devra se détacher de son clan et explorer une nouvelle zone à chaque itération.

Table 2.2: Analogie entre les caractéristiques des éléphants et EHO.

Paramètres et fonctions [5]

Cette méthode repose sur un certain nombre de paramètres empiriques, qui sont :

- **nClan** : le nombre de clans.
- **N** : le nombre d'éléphants d'un clan.
- α : le taux d'influence de la matriarche sur un éléphant.
- β : le taux d'influence du centre de gravité du clan sur la matriarche.

Pour le calcul des positions futures des éléphants et de la dynamique des clans, l'algorithme est basé sur trois fonctions principales, soient :

- Pour chaque éléphant i du clan c , sa prochaine position calculée par l'équation suivante :

$$X_{new,c,i} = X_{c,i} + \alpha * (X_{Best,c} - X_{c,i}) * r \quad (2.1)$$

Avec: $r \in [0, 1]$: distribution uniforme.

$X_{c,i}$ et $X_{new,c,i}$: positions (avant et après) du $i^{\text{ème}}$ éléphant dans le clan c .

$\alpha \in [0,1]$: taux d'influence de la matriarche sur le $i^{\text{ème}}$ éléphant.

- Pour l'éléphant matriarche du clan c , sa position future est calculée comme suit:

$$X_{Best,c} = \beta * X_{center,c} \quad (2.2)$$

Avec: $X_{center,c}$: le centre de gravité du clan c .

β : le taux d'influence du centre de gravité du clan sur la matriarche.

Sachant que le centre de gravité est donné par la formule suivante:

$$X_{center,c} = \frac{1}{N} * \sum_{i=1}^N X_{c,i} \quad (2.3)$$

- Pour le pire éléphant (*Worst*) qui quittera le clan, sa position est déterminée par l'estimation:

$$X_{Worst,c} = X_{min} + (X_{max} - X_{min}) * rand \quad (2.4)$$

Avec: $rand \in [0, 1]$: distribution stochastique uniforme.

X_{min} et X_{max} : limites (inférieure, supérieure) de la position d'un éléphant.

Pseudo code de EHO [5]

```

1 Algorithme : EHO
2
3   Input : MaxGen : le nombre maximum de génération,
4   Output : meilleure solution
5   begin
6     Initialisation
7     t = 1; /* Initialisation du compteur de génération */ 
8     initPopulation(); /* initialisation des solutions des éléphants. */
9     while t < MaxGen do
10    /* Tri de tous les éléphants selon la fonction d'évaluation.
11       */
12    TriElephant();
13    /* Mise à jour de la position des éléphants de chaque clan. */
14    MAJPositionClan();
15    /* Mise à jour de la position du pire éléphant de chaque clan.
16       */
17    OperateurSéparation();
18    /* Evaluation de la population selon leur nouvelle solution.
19       */
20    EvaluationPopulation();
21    t ← t + 1;
22  end
23 end

```

Algorithme 2 : EHO Générique

```

1 Fonction : MAJPositionClan
2
3   Input : nClan : nombre de clans, N : nombre d'éléphants d'un clan
4   /* pour chaque clan de la population */
5   for c ← 1 to nClan do
6     /* pour chaque éléphant du clan c */
7     for i ← 1 to N do
8       Mise à jour de  $X_{c,i}$  et calcul de  $X_{new,c,i}$  selon Eq. (2.1).
9       if  $X_{c,i} = X_{Best,c}$  then
10        | Mise à jour de  $X_{c,i}$  et calcul de  $X_{new,c,i}$  selon Eq. (2.2).
11      end
12    end
13  end

```

Fonction MAJPositionClan

```

1 Fonction : OpérationSéparation
2
3   Input : nClan : nombre de clan de la population
4   /* pour chaque clan de la population */
5   for c ← 1 to nClan do
6     | Remplacer le pire éléphant du clan c selon Eq. (2.4).
7   end

```

Fonction OpérationSéparation

2.4.3 EWSA : Elephant Swarm Water Search Algorithm

Analogie entre les éléphants et EWSA [6]

L'algorithme évolutionnaire EWSA proposé par *S Mandal*, simule aussi l'évolution du mouvement des éléphants mais pour la recherche d'eau. La métaphore entre le milieu naturel et la résolution de problèmes est présentée dans le tableau 2.3 suivant :

Groupes d'éléphants	EWSA
Les éléphants sont à la recherche de meilleures sources d'eau	Les éléphants cherchent la meilleure solution de l'espace des solutions.
L'endroit où se trouve un éléphant dans son territoire.	La position d'une solution dans l'espace de recherche.
Le déplacement d'un éléphant se fait selon sa vitesse.	Le changement de position d'une solution dans l'espace de recherche est calculé sur la base de la vitesse de l'éléphant artificiel.
Un meilleur niveau d'eau dénote une meilleure zone pour la survie des éléphants.	Une meilleure solution est déterminée par une meilleure évaluation par la fonction objectif.
Les éléphants forment plusieurs groupes, formant un essaim d'éléphants. Chaque groupe est composé d'un certain nombre d'éléphants	Les groupes (ou clans) d'éléphants sont représentés par un éléphant unique nommé chef du clan.
Capacité de communication à grande distance et à courte distance des éléphants.	Recherche globale et locale des éléphants.

Chaque fois qu'un groupe d'éléphants trouve une source d'eau, il communique aux autres groupes de l'essaim la quantité d'eau trouvée.	les groupes d'éléphants communiquent entre eux les solutions trouvées et leur évaluation.
Les groupes d'éléphants à la recherche d'une source d'eau, gardent en mémoire la meilleure source trouvée en plus de celle trouvée par d'autres groupes communiquant avec eux	Chaque groupe d'éléphants se souvient de sa propre meilleure solution (locale), et de la meilleure solution de tout l'essaim (globale).
La proximité physique entre groupes d'éléphants et d'autres facteurs tels que l'atténuation du signal à grande distance influent la recherche d'eau.	La recherche de solution locales et globales est contrôlée par une Constante probabiliste p .

Table 2.3: Analogie entre les caractéristiques des éléphants et EWSA.

Paramètres et fonctions [6]

L'ensemble des paramètres empiriques de cette méthode sont :

- N : le nombre de groupes.
- p : la constante de commutation (pour basculer entre la recherche globale et locale).
- w^t : le poids d'inertie à l'itération t (pour équilibrer entre exploration et exploitation).

Afin de calculer la position et vitesse suivantes des éléphants, l'algorithme repose sur un ensemble de fonctions. Mais avant, nous devons définir le format dans lequel sont représentées les positions et vitesses des groupes d'éléphants, nous avons:

$V_{i,d}^t = (v_{i1}, v_{i2}, \dots, v_{id})$: la vitesse du i^{eme} groupe d'éléphant à l'itération t .
 $X_{i,d}^t = (x_{i1}, x_{i2}, \dots, x_{id})$: la position du i^{eme} groupe d'éléphant à l'itération t .
 $P_{Best,i,d}^t = (p_{i1}, p_{i2}, \dots, p_{id})$: la meilleure position du i^{eme} groupe d'éléphant à l'itération t .
 $G_{Best,d}^t = (g_1, g_2, \dots, g_d)$: la meilleure position de tous l'essaim d'éléphant à l'itération t .
 X_{max} et X_{min} : limite supérieure et limite inférieure des positions (dans l'environnement).
 d : la dimension d'une solution.

- Pour chaque groupe d'éléphants i , sa vitesse est donnée par la formule:

$$V_{i,d}^{t+1} = \begin{cases} V_{i,d}^t * w^t + rand(1, d) \odot (G_{best,d}^t - X_{i,d}^t) & \text{si } random > p \\ V_{i,d}^t * w^t + rand(1, d) \odot (P_{best,i,d}^t - X_{i,d}^t) & \text{si } random \leq p \end{cases} \quad (2.5)$$

Ainsi on obtient sa position par l'équation:

$$X_{i,d}^{t+1} = V_{i,d}^{t+1} + X_{i,d}^t \quad (2.6)$$

Avec:

$rand(1, d) \in [0,1]$: tableau de dimension d de valeurs aléatoires.

w^t : le poids d'inertie à l'itération t .

$random \in [0,1]$: un chiffre aléatoire entre $[0,1]$.

p : constante de commutation.
 \odot : multiplication élément par élément.

- Le poids d'inertie à l'itération t , w^t peut être calculé de diverses manières, soient les trois principales utilisées par S.Mandal sont :

- Poids d'inertie constant / Constant Inertia Weight (CIW):

$$w^t = \text{constante} \quad (\text{généralement} = 0.5) \quad (2.7)$$

- Poids d'inertie aléatoire / Random Inertia Weight (RIW):

$$w^t = 0.5 + \text{rand}/2 \quad (\text{rand} \in [0, 1]) \quad (2.8)$$

- Poids d'inertie décroissant linéairement/ Linearly Decreasing Inertia Weight(LDIW):

$$w^t = w_{\max} - \frac{w_{\max} - w_{\min}}{t_{\max}} * t \quad (2.9)$$

Avec:

w_{\max} et w_{\min} : la valeur maximale et minimale d'inertie.

t : le numéro de l'itération en court.

t_{\max} : le nombre maximal d'itérations.

Pseudo code de EWSA [6]

```

1 Algorithme : EWSA
2
3   Input :  $N$  : nombre de groupes d'éléphants,  $p$  : constante de commutation,
4      $X_{min}$  : borne minimale d'une solution,  $X_{max}$  : borne maximale d'une
5     solution,
6      $t_{max}$  : nombre maximal d'itérations
7   Output :  $G_{Best,d}, f(G_{Best,d})$  : meilleure solution et évaluation.
8
9   begin
10    | for  $i \leftarrow 1$  to  $N$  do
11      |   /* Initialisation de la position et vitesse du  $i^{ème}$  groupe. */
12      |   Initialisation  $X_{i,d}$  et  $V_{i,d}$ ;
13      |   /* Initialisation du PBest du  $i^{ème}$  groupe. */
14      |    $P_{Best,i,d} = X_{i,d}$ ;
15      |   /* Evaluation de la position du  $i^{ème}$  groupe. */
16      |   Evaluation  $f(X_{i,d})$ ;
17    | end
18    |  $G_{best,d} = \text{Min}(f)$ ; /* Initialisation du GBest. */
19    | Initialisation  $w^t$  selon Eq. (2.9)
20
21    | /* Tant que le nombre d'itérations maximal n'est pas atteint */
22    | for  $t \leftarrow 1$  to  $t_{max}$  do
23      |   /* Pour chaque groupe d'éléphants */
24      |   for  $i \leftarrow 1$  to  $N$  do
25        |     | if  $random > p$  then
26        |       |   /* Mise à jour de la vitesse selon le GBest */
27        |       |   Mise à jour de la vitesse  $V_{i,d}$  1ère partie de l' Eq. (2.5)
28        |     | end
29        |     | else
30        |       |   /* Mise à jour de la vitesse selon le PBest (locale) */
31        |       |   Mise à jour de la vitesse  $V_{i,d}$  2ème partie de l' Eq. (2.5)
32        |     | end
33        |     | /* Mise à jour de la position du  $i^{ème}$  groupe. */
34        |     | Mise à jour de la position  $X_{i,d}$  avec Eq. (2.6);
35        |     | /* Evaluation de la position du  $i^{ème}$  groupe. */
36        |     | Evaluation  $f(X_{i,d})$ ;
37        |     | if  $f(X_{i,d}) < f(P_{Best,i,d}^t)$  then
38        |       |       |  $P_{Best,i,d}^t = X_{i,d}$  /* Mise à jour du PBest du  $i^{ème}$  groupe. */
39        |     | end
40        |     | if  $f(P_{Best,i,d}^t) < f(G_{Best,d}^t)$  then
41        |       |       |  $G_{Best,d}^t = P_{Best,i,d}^t$  /* Mise à jour du GBest */
42        |     | end
43    |   | end
44  | end
45  | Retourner ( $G_{Best,d}^t, f(G_{Best,d}^t)$ )
46
47 end

```

Algorithme 3 : EWSA Générique

2.4.4 Remarque

L'ossature des deux algorithmes EHO et EWSA est différente. EHO, la version classique prend en compte la composition d'un clan de plusieurs éléphants contrairement à EWSA qui réduit chaque clan en une seule unité.

Aussi, EHO possède deux facteurs de diversification qui sont le multi-clan et l'opérateur de séparation, mais une intensification ne menant pas toujours à une convergence. À l'opposé de l'approche EWSA qui fournit un certain équilibre entre diversification et intensification malgré les risques de convergence prématurés en vue de sa représentation des clans.

Ces deux méta-heuristiques ont prouvé leurs efficacité, suite à nombreuses expérimentations et comparaisons avec d'autres approches comme par exemple : CS (Cuckoo Search) [42], BA (Bat Algorithm) [43], FPA (Flower Pollination Algorithm) [44] dans le cas de EWSA. Et BBO (Biogeography-Based Optimization) [45], DE (Differential Evolution) [46] et GA (Genetic Algorithm) [47] pour EHO.

2.5 Algorithme Génétique [7]

La première description du processus des algorithmes génétiques a été donnée par Holland en 1975. Puis Goldberg en 1989 les a utilisés pour résoudre des problèmes concrets d'optimisation [7].

2.5.1 Inspiration de la génétique

Les algorithmes génétiques sont des algorithmes évolutionnaires se caractérisant par leur inspiration de l'évolution naturelle des espèces, regroupant les principales composantes de l'ADN, telles que :

- **Chromosome**, c'est une suite d'informations structurées dont l'ordre est important.
- **Gène**, il s'agit d'une unité d'un chromosome, celle-ci renferme une information appelée *Allèle*.
- **Allèle** est une information possédant une position bien définie dans le chromosome. Il s'agit de la valeur que peut prendre un gène et elle est propre à chaque individu.

2.5.2 Métaphore entre l'ADN et un algorithme génétique

L'algorithme génétique a emprunté grand nombre de concepts propres à l'ADN en particulier et à l'évolution des espèces en général, citons les similitudes dans le tableau suivant:

Nature	Algorithme génétique
Une société d'individus.	L'espace de recherche composé de toutes les solutions potentielles.
Une population d'individus appartenant à la société.	Un ensemble de solutions pris de l'espace de recherche.
Le but est la sélection des meilleurs gènes pour former le meilleur individu.	Le but est de trouver la meilleure solution de l'espace des solutions.
Un chromosome permet d'identifier un individu et ses propriétés.	Une solution représente un individu de la population.
Un chromosome regroupe plusieurs informations génétiques	Une solution est formée d'une suite de valeurs.
La reproduction entre deux individus donne naissance à des enfants.	L'opération de croisement entre deux solutions permet d'injecter de nouvelles solutions dans la population.
Pendant la création de l'ADN fils quelques modifications de gènes ou anomalies peuvent survenir.	Après la création de la solution fils, elle peut subir une mutation.
L'accouplement de parents ayant les meilleurs caractères héréditaires sont plus susceptibles de donner naissance à un fils doté des meilleurs gènes	La sélection des meilleurs parents à chaque itération augmente les chances d'atteindre la solution optimale.

Table 2.4: Analogie entre les caractéristiques de l'ADN et GA.

2.5.3 Paramètres et fonctions

Opération de croisement

Le croisement est appliqué sur deux solutions parents et donne naissance à deux solutions fils, Cette opération consiste à fragmenter les solutions parents pour ensuite les regrouper différemment.

Un point de croisement Généralement, ce point est pris aléatoirement, divisant chaque solution en deux fragments, les deux nouveaux individus prendront chacun une partie des gènes de chaque parent. Le processus est illustré dans la figure 2.2.



Figure 2.2: Croisement en un point.

Deux points de croisement Pour deux points de croisement, chaque solution parent devra être fragmentée en trois, ainsi chaque solution fils héritera de trois fragments (1 d'un parent et 2 de l'autre), comme représentés dans la figure 2.3 ci-dessous.



Figure 2.3: Croisement en deux points.

Opération de mutation

La mutation apporte l'aléa nécessaire à une exploration efficace de l'espace des solutions en permettant de quitter les extrêmes locaux. Il existe plusieurs façons de procéder à une mutation:

Inversion d'un gène Cette méthode consiste en l'inversement ou la troncature d'un certain nombre fixe ou aléatoire de gènes d'une solution. La figure 2.4 en est un exemple.



Figure 2.4: Mutation avec inversement.

Permutation de gènes Comme le montre la figure 2.5, la permutation de gènes nécessite d'abord la sélection de deux gènes sur l'individu (chromosome) à muter. Puis vient l'échange des deux valeurs (allèles) correspondantes, cela produit un nouvel individu.



Figure 2.5: Mutation avec permutation.

2.5.4 Pseudo code (GA incrémental)

```

1 Algorithme : GA
2
3   Input :  $N$  : taille de la population,  $MaxGen$  : nombre maximal de générations
4   Output : meilleure solution
5   begin
6      $K \leftarrow 0$ ;
7      $P_k \leftarrow \text{GénérationSolutions}(N)$ ;
8      $\text{Evaluation}(P_k)$ ; /* Évaluation de tous les individus de la
9       population */

10    while  $\neg \text{BonneSolution} \& k < MaxGen$  do
11      /* Sélection des deux meilleurs parents de la population */
12      ( $Parent1, Parent2$ )  $\leftarrow \text{MeilleurIndividus}(P_K)$ ;
13
14      /* Calcul de la position du point de croisement. */
15       $i \leftarrow \text{PointCroisement}(\text{taille}(\text{solution}))$ ;
16
17      /* Croisement des solutions parents */
18       $Fils1 \leftarrow \text{Croisement}(Parent1, Parent2, i)$ ;
19       $Fils2 \leftarrow \text{Croisement}(Parent2, Parent1, i)$ ;
20
21      /* Évaluation des solutions fils */
22       $\text{Evaluation}(Fils1)$ ;  $\text{Evaluation}(Fils2)$ ;
23
24      /* Détermination du nombre de mutations. */
25       $j \leftarrow \text{NombreMutation}(\text{taille}(\text{solution}))$ ;
26
27      /* Mutation des solutions fils */
28       $FilsMutant1 \leftarrow \text{Mutation}(Fils1, j)$ ;
29       $FilsMutant2 \leftarrow \text{Mutation}(Fils2, j)$ ;
30
31      /* Évaluation des solutions fils mutantes */
32       $\text{Evaluation}(FilsMutant1)$ ;  $\text{Evaluation}(FilsMutant2)$ ;
33
34       $K \leftarrow K + 1$ ;
35
36      /* Mise à jour de la population, suppression des solutions
37       parents et insertion des solutions fils et fils mutantes.
38       */
39       $K \leftarrow K + 1$ ;  $\text{Insertion}(P_k, Newindividus)$ ;
40
41    end
42
43 end

```

Algorithme 4 : Algorithme Génétique

2.6 Conclusion

Comme on a pu le constater à travers ce chapitre, le comportement collectif des éléphants et des abeilles ainsi que la complexité de leur psychologie ont donné lieu à de puissants algorithmes de résolution orientés espaces des solutions. Ces derniers ont prouvé leur performance et leur efficacité face à de nombreux problèmes. Ainsi, vu notre problématique, qui rappelons-le, consiste à détecter des cibles dans un espace complexe inconnu à l'aide de robots mobiles, nous avons choisi ces algorithmes intelligents pour la résolution de notre problème compte tenu de leur robustesse. Le chapitre qui suit portera

sur la modélisation de notre solution sur laquelle se baseront nos différentes approches mono-BSO, multi-BSO, EHO ainsi que EWSA.

Chapter 3

Chapitre 3 : Modélisation du problème de recherche de cibles

3.1 Introduction

L'étape de modélisation d'un algorithme basé intelligence en essaim est centrale et requiert un effort cérébral considérable pour les problèmes complexes.

Ce chapitre est dédié à la description détaillée de notre modélisation incluant la représentation de l'espace de recherche, de la solution, de la fonction objectif et de tous les autres composants de l'environnement de recherche. Par la suite, nous passerons en revue des méthodes de la littérature sur l'évitement d'obstacles avant d'enchaîner avec la méthode sélectionnée et l'argumentation de son choix. Puis nous nous pencherons sur le principe de fonctionnement de l'algorithme génétique que nous utiliserons comme outil de paramétrage des méta-heuristiques de recherche de cibles.

3.2 Modélisation de l'environnement de recherche

Nous avons modélisé notre environnement de recherche par une représentation 2D dite "*grid based*", sous forme d'une grille ou matrice comme le montre la figure 3.1. Notre environnement est caractérisé par une surface préalablement fixée, afin que les bordures délimitent la zone de recherche, pour cela la forme de l'environnement est carrée telle que :

$$\text{Surface} = \text{Taille}_{\text{Coté}} * \text{Taille}_{\text{Coté}}. \quad (3.1)$$

Dans cette grille, chaque case contient une valeur ainsi qu'une position / coordonnée (x,y) , composée d'une abscisse (x) et d'une ordonnée (y). Plusieurs valeurs sont admissibles dans les cases, celles-ci permettent de distinguer les objets comme suit :

- **Cible** : Dans ce cas la valeur assignée à cette position est égale à **1**.
- **Portée d'une cible** : La valeur est bornée par **0** et **1**, ces derniers exclus.
(valeur(x,y) $\in]0,1[$)
- **Obstacle** : Sa valeur est égale à **-1**.
- **Zone neutre** : Prend la valeur **0**.

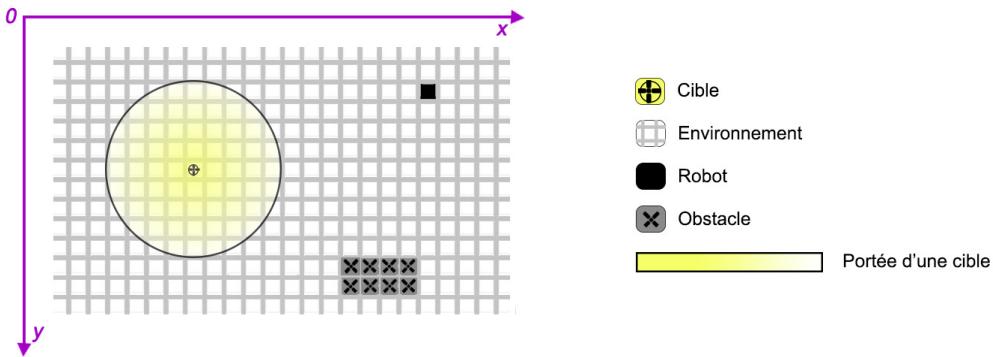


Figure 3.1: Représentation de l'environnement.

3.2.1 Représentation de la solution

Le but étant de trouver les cibles dans un environnement inconnu. L'espace des solutions à explorer n'est autre que notre environnement 2D composé de positions que nous appellerons aussi "solutions".

Une solution optimale est une position géographique de coordonnée (x,y) décrite par la valeur de sa case. La figure 3.2 représente une solution dans l'environnement de recherche.

Dans le cas de multiples cibles on devra trouver autant de solutions optimales que de cibles, telles que chaque solution correspond à une case de valeur 1.

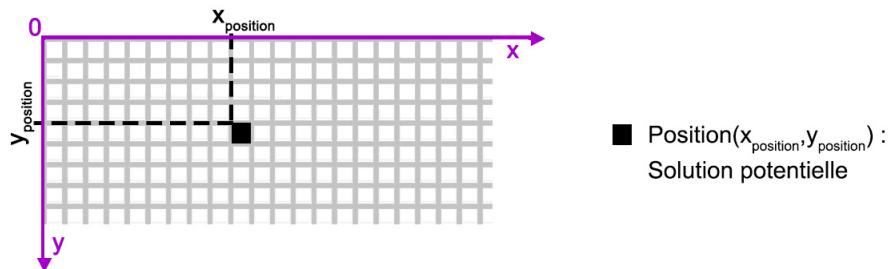


Figure 3.2: Représentation d'une solution dans notre modélisation.

Une solution doit satisfaire deux contraintes. D'abord celle de la bornitude par les dimensions de l'environnement de recherche, ce dernier étant carré de taille $Taille_{Côté}$, une solution est telle que:

$$\begin{cases} 0 \leq Position.x < Taille_{Côté} \\ 0 \leq Position.y < Taille_{Côté} \end{cases} \quad (3.2)$$

Puis en tenant compte qu'une position contenant un obstacle n'est pas une solution admissible, soit :

$$environnement.get(Position.x, Position.y) \neq -1 \quad (3.3)$$

3.2.2 Fonction objectif

Les solutions ont besoin d'être évaluées afin de mesurer leurs distances par rapport à une cible. Ainsi, on pourra choisir la plus proche à chaque itération, notre fonction objectif varie selon l'intervalle borné suivant : [0,1].

Elle est égale à la valeur captée par les capteurs du robot, telle que plus la valeur se rapproche du 1 plus le robot est proche de la cible, car son émission est plus importante. Comme nous le verrons plus bas (Section 3.3.3), l'évaluation se fait sur les cases visibles par le robot, cela à travers son champ de vision.

3.2.3 Les cibles

Une cible occupe une case unique, mais elle possède aussi une portée fixe relative à l'émission de signaux perceptibles par les robots.

Comme on peut le voir dans la figure 3.3, la portée décroît à partir de la position de la cible valant "1", jusqu'à la valeur "0" lorsque la distance par rapport à la cible excède la taille de la portée. Autrement dit, les valeurs représentatives des émissions de la cible sont inversement proportionnelles à la distance des positions faisant partie de la portée.

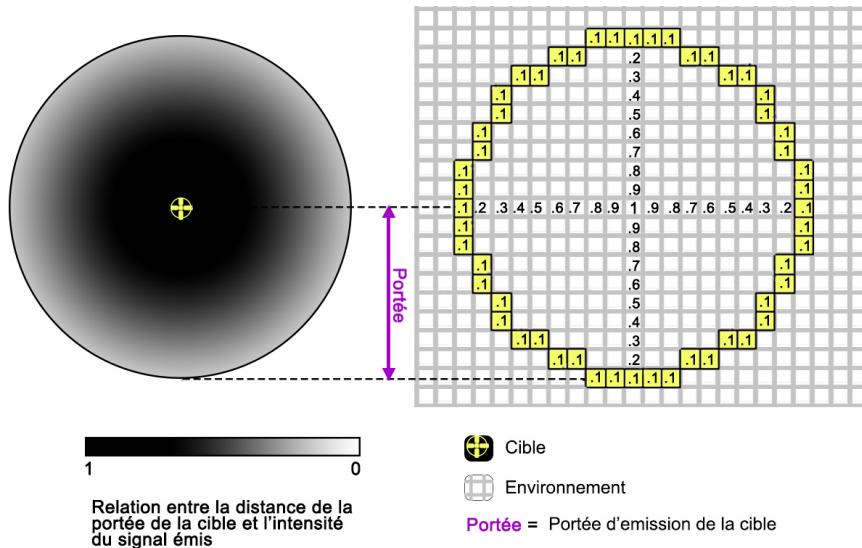


Figure 3.3: Représentation de la portée d'une cible dans notre modélisation.

Remarque: Une fois qu'une cible est atteinte par un robot, elle est désactivée, elle arrête alors d'émettre des signaux. Nous avons modélisé cette action par la mise à zéro des cases de la grille (environnement) constituant la portée de cette cible.

3.2.4 Les obstacles

Un obstacle est considéré comme un objet fixe (statique) occupant une certaine surface de l'environnement empêchant les robots de s'y positionner ou de la traverser.

Les environnements peuvent être catégorisés en trois (3) types selon la densité des obstacles, soient :

- **Environnements simples (sans obstacles)**, ceux-là ne contiennent que la ou les cible(s).
- **Environnements avec obstacles**, comportant un nombre réduit d'obstacle de forme rectangulaire ainsi que la ou les cible(s).

La taille d'un obstacle est comprise entre 2% et 4% de la taille de l'environnement. Quant au nombre d'obstacles il appartient à l'intervalle [15 , 25] obstacles.

- **Environnements complexes** : ils possèdent plus d'obstacles et sont plus grands en termes de superficie que ceux de la catégorie précédente, en plus de la ou les cible(s).

Dans ces environnements, la taille d'un obstacle est comprise entre 4% et 6% de la taille de l'environnement avec une densité de [25 , 35] obstacles.

Ainsi, le choix d'une bonne stratégie d'évitement d'obstacles adaptée à notre modélisation devient crucial.

3.3 Stratégies d'évitement d'obstacles

Une stratégie d'évitement d'obstacles est nécessaire pour la navigation des robots dans des environnements à obstacles et complexes. À noter pour ce qui suit que le but est de trouver une trajectoire sûre (sans obstacles) entre deux positions, c'est-à-dire entre la position initiale d'un robot et sa position destination ou but.

3.3.1 Description des approches existantes

Les techniques les plus utilisées pour l'évitement d'obstacles sont décrites ci-dessous:

Les algorithmes Bug [8, 9]

Les algorithmes Bug1 et Bug2 sont des méthodes anciennes et simples. Ils réduisent le robot en un point dans un plan 2D détectant les obstacles via capteurs tactiles. Ils sont basés sur deux comportements : "*déplacement vers le but*" et "*suivi d'une limite*".

Bug1 , tant que le robot n'a pas rencontré d'obstacle, il suit le comportement "*déplacement vers le but*", il se dirige tout droit vers le but (T), lorsqu'il rencontre un obstacle en un point (H_i) il bascule vers le comportement "*suivi d'une limite*" en faisant un tour complet sur l'obstacle. À partir de là il détermine le point du périmètre de l'obstacle (L_i) le plus proche de la position but et s'y positionne. Ce processus se répète jusqu'à atteindre la destination. (voir la figure 3.4).

Bug2 exploite la première solution prometteuse qu'il trouve. Durant le comportement "*déplacement vers le but*", le robot se déplace selon des lignes droites reliant la position initiale (S) du robot et la position but (T). Quand un obstacle est rencontré, le robot adopte le comportement "*suivi d'une limite*", consistant à faire le tour de l'obstacle, jusqu'à atteindre un nouveau point (H_i) qui appartient à la droite (S , T) (voir la figure 3.4). Le processus est répété tant que la position but n'est pas atteint par le robot.

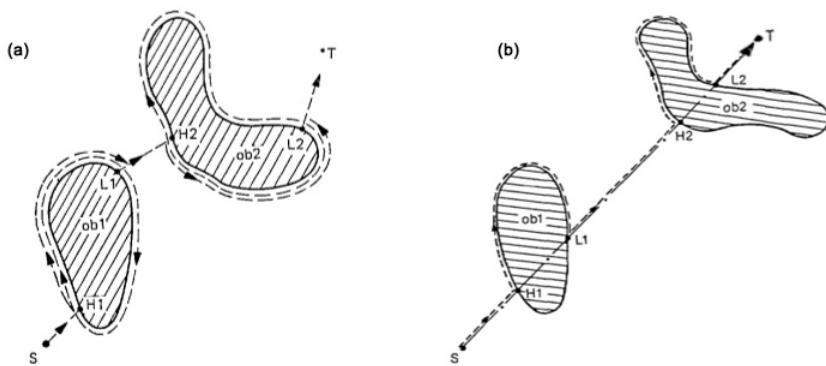


Figure 3.4: Calcul du chemin du robot par les algorithmes Bug : (a) Bug1, (b) Bug2 [9].

La méthode basée sur les champs de potentiel PF (Potential Field) [8]

Cette méthode a été proposée pour la première fois par Oussama Khatib [48]. Elle considère le robot comme une particule plongée dans un champ de potentiel, celui-ci régie par deux forces, soient :

Force attractive $U_{attract}$: générée par le but.

Force répulsive $U_{répuls}$: générée par les obstacles.

Ainsi, le robot calcule d'une manière itérative le mouvement à suivre, suivant une direction résultante des sommes de différents champs potentiels (voir la figure 3.5) et qui est donnée par la fonction :

$$F = -\nabla(U_{attract} + U_{répuls}) \quad (3.4)$$

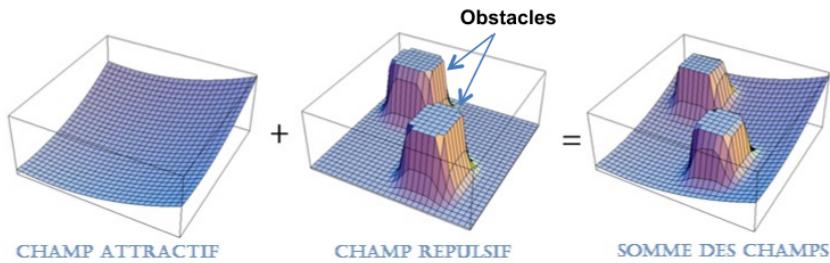


Figure 3.5: Champs de potentiel pour un environnement contenant deux obstacles et une position but.[10]

La méthode basée sur la fenêtre dynamique DW (Dynamic Window) [8]

La fenêtre dynamique est une méthode proposée par Burgardand et Thruncite [49]. Elle vise à choisir un couple (v,w) représentant respectivement la vitesse linéaire et angulaire du robot permettant d'éviter les obstacles perçus localement. Sur la base des différents couples possibles, DW opte pour le couple de vitesses le plus pertinent. Comme le montre la figure 3.6, la zone de recherche est limitée par une fenêtre dynamique basée sur les limitations dynamiques du robot et les vitesses admissibles qui peuvent être atteintes durant un laps de temps donné.

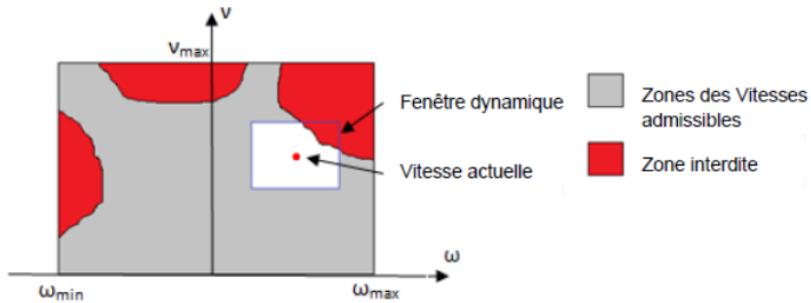


Figure 3.6: L'approche de la fenêtre dynamique montrant les régions admissibles et interdites par rapport à la fenêtre dynamique des vitesses atteignables par le robot [8].

La logique floue

La logique floue est une branche de l'intelligence artificielle et des mathématiques qui a été appliquée à des problèmes de commandes (Bühler 1997) [50]. Cette technique se base sur la déduction de deux commandes : la vitesse linéaire et angulaire selon les variables

floues des données en entrées, qui sont : la vitesse (v) du robot, et l'angle (α) entre le robot et l'obstacle, en plus d'une base de règles afin d'éviter les obstacles [51].

L'échantillonnage de l'espace d'entrée ISS (Input Space Sample)

À partir d'un état initial du robot, le modèle prédictif du système est utilisé pour la génération d'un ensemble de trajectoires. Celles-ci peuvent être triées selon une fonction de coût. L'exemple d'un ensemble de trajectoires générées en utilisant la technique par échantillonnage de l'espace d'entrées est illustré dans la figure 3.7.

Par la suite, selon les obstacles visibles par le robot, certaines trajectoires seront bannies de l'ensemble des trajectoires admissibles [8], comme il est le cas de $S1, S2, S7, S8$ et $S9$.

Les travaux de *Kelly* et *Stentz* [52] font partie des premiers travaux basés sur cette technique.

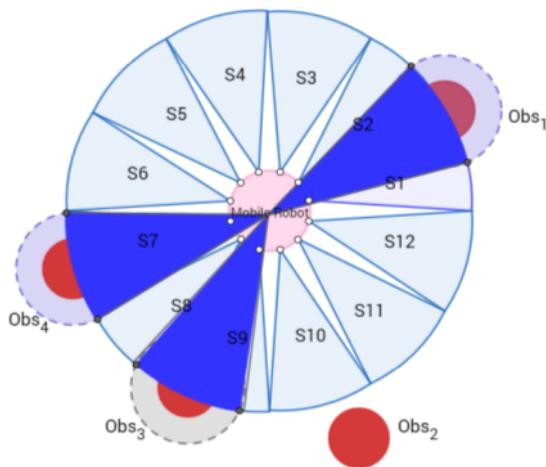


Figure 3.7: Ensemble de trajectoires générées par échantillonnage de l'espace d'entrées [11].

3.3.2 Discussion et choix

Les algorithmes bug1 et bug2 sont faciles à mettre en œuvre et peu gourmands en termes de mémoires. Cependant, ils nécessitent des déplacements supplémentaires et souvent la trajectoire trouvée est loin d'être optimale.

Les méthodes de champ de potentiel et de logique floue, sont facilement adaptables aux formules des méta-heuristiques disposant d'une vitesse telles que EWSA.

Cependant, elles sont difficiles à mettre en pratique pour d'autres types de méta-heuristiques dont BSO et EHO, car le calcul des positions est indépendant de la direction de l'ancienne position vers la nouvelle.

De plus, elles ne permettent pas de détecter un passage entre des obstacles assez proches, encore pire, le robot risque de ne jamais atteindre la position but, si ce dernier est à proximité directe d'un obstacle.

D'autre part, la méthode de la fenêtre dynamique (DW) n'est pas sujette aux limites précédemment citées. Mais elle peut s'avérer être une recherche exhaustive d'une trajectoire optimale selon la dimension de la fenêtre, car pour une fenêtre de taille (x, y) nous aurons $x * y$ paires (v, w) à tester.

Enfin, la stratégie d'échantillonnage, n'est pas concernée par les problèmes cités ci-dessus dans le cadre de notre modélisation. Elle est simple et efficace, mais surtout elle peut être directement adaptée ou intégrée dans d'autres concepts ou algorithmes.

Après avoir passé en revue les avantages et inconvénients de chaque approche face à notre modélisation, notre choix s'est porté sur la stratégie d'**échantillonnage**.

3.3.3 Paramètres de la méthode d'échantillonnage

L'échantillonnage de l'espace local des robots est régi par plusieurs paramètres devant être fixés au préalable, soient :

Champ de vision

L'espace local d'un robot est divisé en quatre zones de 90° , comme le montre la figure 3.8 : "NE: Nord-Est, NW: Nord-Ouest, SW: Sud-Ouest, SE: Sud-Est".

Le champ de vision d'un robot est d'un angle de 90° , choisi selon la zone contenant la droite reliant la position du robot à la position but.

Afin de déterminer la zone concernée, nous devons effectuer deux simples opérations de soustraction sur les coordonnées du robot (x_R, y_R) et celles de la position destination (x_{but}, y_{but}). Les zones relatives aux résultats de ces opérations sont représentées dans le schéma ci-dessous.

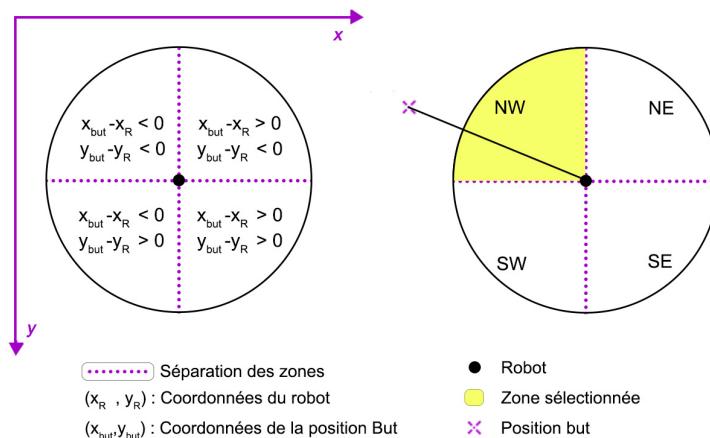


Figure 3.8: Détermination de l'angle de vue d'un robot dans notre modélisation.

Portée locale d'un robot

Chaque robot possède une portée locale, celle-ci forme une surface ronde dont le robot est le centre, le rayon de ce cercle est limité par la portée des capteurs présents sur le robot.

Dans notre modélisation, cette portée locale est fixée à une distance de dix positions du robot. La zone maximale visible par chacun de nos robots est représentée dans la figure 3.9 qui suit:

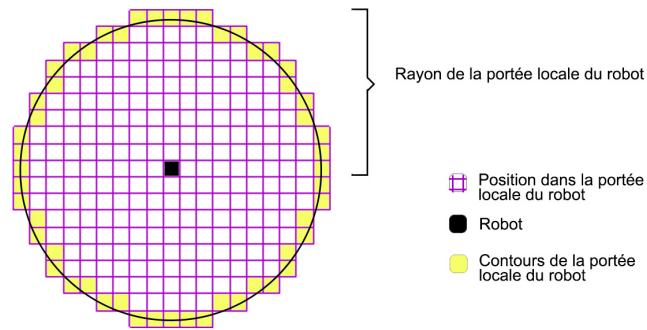


Figure 3.9: Représentation de la portée locale d'un robot dans notre modélisation.

Il est à noter que les robots ont cette portée de vue durant tout le déplacement, c'est-à-dire que tout le long de leur trajectoire, une vérification de cette surface accessible est effectuée.

Nos robots prennent place dans les meilleures positions visibles dans leurs portées, cela à travers le processus suivant :

Input : P : position du robot,
 $portee$: Portée des capteurs = 10,
 env : Matrice de dimension $M \times M$

Réultat : meilleurePosition : Meilleure position maximisant la fonction objectif

```

1 meilleureLocale ← env.valeur( $P_x, P_y$ );
2 meilleurePosition ← Position( $P_x, P_y$ );
3 for  $i \leftarrow -portee$  to  $+portee$  do
4   for  $j \leftarrow -portee$  to  $+portee$  do
5     if env.valide( $P_x + i, P_y + j$ ) And  $i^2 + j^2 \leq portee^2$  then
6       b ← env.valeur( $P_x + i, P_y + j$ );
7       if meilleureLocale < b then
8         meilleureLocale ← b;
9         meilleurePosition ← Position( $P_x + i, P_y + j$ );
10      end
11    end
12  end
13 end
14 return meilleurePosition;
```

Algorithme 5 : Meilleure position dans la portée du robot

Trajectoire

Une trajectoire est modélisée par une droite reliant deux positions successives, cette droite est un ensemble de cases (points) extraits selon l'équation de la droite, *Bresenham* [53] l'a exploité pour en extraire l'algorithme 6 suivant :

```

Input :  $x_0, y_0, x_1, y_1$ 
1  $dx = \text{Abs}(x_1 - x_0); dy = -\text{Abs}(y_1 - y_0);$ 
2 if  $x_0 < x_1$  then  $sx = 1;$ 
3 else  $sx = -1;$ 
4 if  $y_0 < y_1$  then  $sy = 1;$ 
5 else  $sy = -1;$ 
6  $err = dx + dy;$ 
7 while True do
8   setPoint ( $x_0, y_0$ );
9    $e2 = 2 * err;$ 
10  if  $e2 >= dy$  then if  $x_0 == x_1$ 
    then break;
11   $err += dy; x_0 += sx;$ 
12  if  $e2 >= dx$  then if  $y_0 == y_1$ 
    then break;
13   $err += dx; y_0 += sy;$ 
14 end

```

Algorithme 6 : line-drawing algorithm [53]

Grâce à cet algorithme de Bresenham, nous pouvons au lieu de chercher à dessiner la droite à partir d'une position initiale et une finale comme le montre la figure 3.10, l'exploiter pour vérifier si une trajectoire comporte des obstacles ou non.

Critères de choix de la trajectoire

Une fois que nous sommes en mesure de déterminer si une trajectoire est admissible (sans obstacles) ou interdite (contient des obstacles), vient l'étape de sélection de la trajectoire admissible la plus adéquate.

Cette seconde étape revient à choisir la meilleure trajectoire parmi celles admissibles en minimisant la distance entre la position choisie (accessible) et la position destination de notre robot. Les deux étapes sont illustrées dans la figure 3.11 ci-dessous:

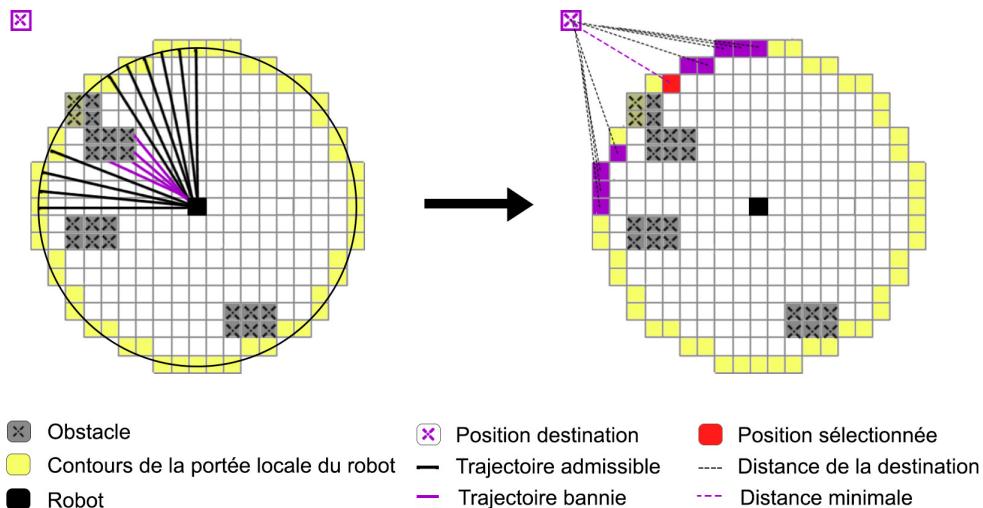


Figure 3.11: Choix de la trajectoire à suivre par le robot dans un environnement à obstacle.

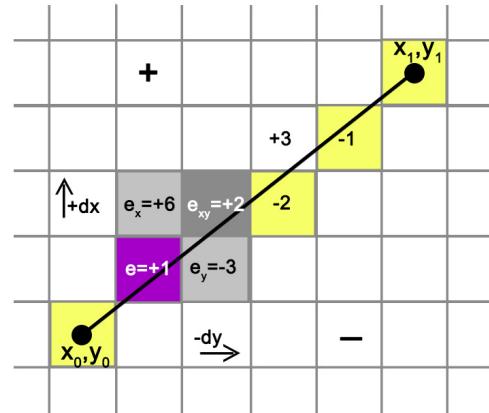


Figure 3.10: Sélection des points(cases) d'une droite avec l'algorithme de BRESENHAM.

Remarque: Si la position de destination est à une distance inférieure à la portée du robot (soit inférieure à dix cases.), le même procédé est effectué avec une portée égale à la distance entre le robot et la position destination.

Autres cas de figure Plusieurs cas de figure existent selon la position des obstacles par rapport à celle du robot, pour cela nous avons étudié toutes les possibilités pour garantir le bon fonctionnement de notre approche dans tous les environnements.

La figure 3.12 représente deux cas de figure se produisant souvent dans des environnements complexes. Lorsque aucune trajectoire admissible n'est trouvée dans le champ de vision initial du robot (90°), on l'élargit aux zones adjacentes de 90° allant ainsi à un champ de vision qui équivaut les 270° . Ceci simulera une rotation du robot sur lui-même (voir schéma de droite).

Si aucune trajectoire admissible n'est trouvée par le robot, une nouvelle position destination est demandée.

Aussi dans le cas où la position destination est alignée sur un des deux axes avec la position actuelle du robot (c'est-à-dire qu'ils possèdent la même abscisse ou même ordonnée), notre robot aura une vue sur les 180° dont la droite (position du robot - position destination) est commune (schéma de gauche).

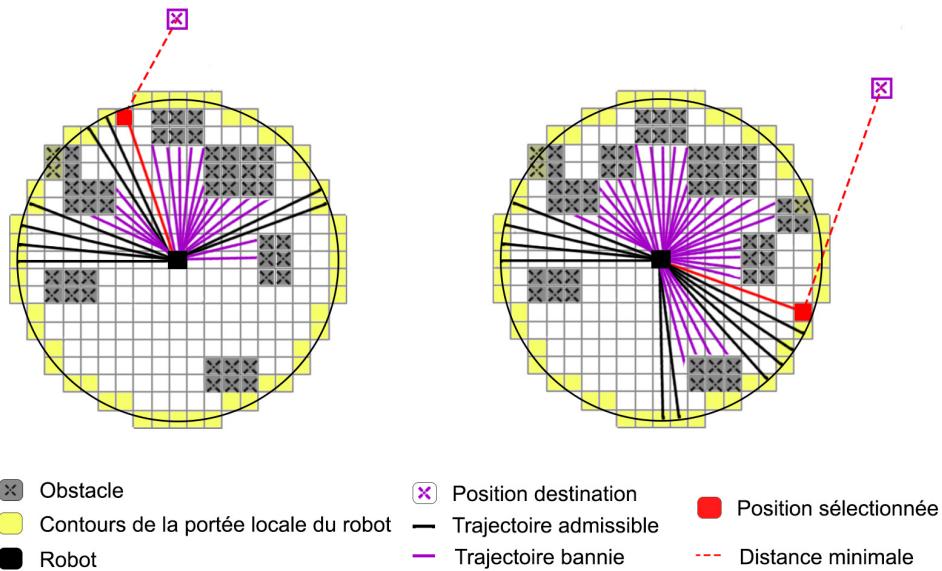


Figure 3.12: Choix de la trajectoire à suivre par le robot dans des situations complexes.

3.4 Auto-Paramétrage avec GA

3.4.1 Modélisation

Solution

Une solution pour notre algorithme génétique est un paramétrage pour une de nos approches de résolution. Elle est représentée par un vecteur de paramètres.

Vu que nos méta-heuristiques n'ont ni le même nombre de paramètres, ni les mêmes plages de valeurs de ces paramètres, la taille de la solution ainsi que les contraintes singulières relatives à chaque approche seront fixées, celles-ci sont représentées dans le tableau 3.1 ci-dessous :

Paramètres	Paramètre 1	Paramètre 2	Paramètre 3	Paramètre 4
BSO	Flip	nbrBees	MaxChance	-
Min	1	1	1	-
Max	100	25	4	-
MBSO	Flip	nbrBees	MaxChance	nbSwarm
Min	1	1	1	1
Max	100	5	4	5
EHO	nbrClan	nbrElephant	Alpha	Beta
Min	1	1	0.1	0.1
Max	5	5	0.9	0.9
ESWSA	nbElephant	P	inertia Weight	-
Min	1	0.1	0.1	-
Max	25	0.9	0.9	-

Table 3.1: Contraintes sur les paramètres des méta-heuristiques.

Espace des solutions

L'espace des solutions diffère selon l'approche à paramétrier. Pour une approche donnée, Il s'agit de l'ensemble des combinaisons possibles des différentes valeurs de chaque paramètre de cette approche.

Comme présenté dans le tableau 3.1, les domaines de chaque paramètre sont clairement définis, ainsi l'espace des solutions de notre algorithme génétique comporte:

- Pour BSO : $100 \times 25 \times 4 = 10\,000$ solutions.
- Pour MBSO : $100 \times 5 \times 4 \times 5 = 10\,000$ solutions.
- Pour EHO : $5 \times 5 \times 9 \times 9 = 2\,025$ solutions.
- Pour ESWSA : $25 \times 9 \times 9 = 2\,025$ solutions.

Fonction objectif

Rappelons que l'objectif de notre algorithme génétique est de trouver la meilleure combinaison de paramètre pour nos approches de recherche de cibles. C'est pourquoi notre évaluation se résume en une exécution d'une approche de recherche, compte à notre fonction objectif, elle se décompose en deux sous fonctions objectif:

1. Meilleure en termes de nombre de cibles trouvées, qui est à maximiser.
2. Meilleure en termes de nombre d'itérations, celle-ci est à minimiser.

Ces deux fonctions suivent une hiérarchie conforme à l'ordre dans lequel nous les avons citées, telles que la meilleure solution est celle qui permet de trouver le maximum de cibles d'abord, mais aussi qui minimise le nombre d'itérations effectué pour les atteindre.

Population

La taille de la population N initiale est un paramètre à régler par expérimentations, il dépend aussi de l'espace des solutions à explorer.

Croisement

Un seul point de croisement est choisi, il est défini de manière aléatoire selon la contrainte suivante:

$$0 < \text{pointCroisement} < \text{nbrParamtres} \quad (3.5)$$

Une fois le point de croisement déterminé, nous passons à la formation des solutions fils, grâce à l'opérateur de concaténation. La figure 3.13 suivante explicite le déroulement d'un croisement.



Figure 3.13: Croisement de deux solutions.

Mutation

Une mutation est le remplacement d'un ou plusieurs paramètres de notre solution par une valeur aléatoire toujours en respectant les contraintes d'intervalle de chaque paramètre cité dans le tableau 3.1. On choisit de faire une seule mutation aléatoire explicitée dans la figure 3.14 suivante:



Figure 3.14: Mutation d'une solution.

3.4.2 Fonctionnement

La mét-heuristique GA comporte une population de N individus dont nous essayons d'améliorer les gènes pour obtenir le meilleur individu au fil des générations. L'organigramme de la figure 3.15 décrit ce processus en détail.

Tout d'abord les bornes (inférieurs et supérieurs) de chaque paramètre pour chaque approche à paramétrier doivent être initialisées, afin que les valeurs générées pour chaque individu soient cohérentes avec l'espace des solutions. Puis, les N individus qui vont constituer notre population sont générés et évalués un à un selon la fonction objectif décrite plus haut, avant leur insertion dans la population P.

Le processus de paramétrage est mis en route et ne s'arrête qu'une fois le nombre maximal de générations **MaxGen** atteint. L'auto-paramétrage suit les étapes ci-contre:

1. Les deux meilleures solutions de la population **P** de la " $t^{ième}$ " génération sont sélectionnées, soient " $S1$ " et " $S2$ ".
2. Effectuation du croisement des deux solutions " $S1$ " et " $S2$ " selon le point de croisement **pointCroisement**, produisant ainsi deux solutions enfants: " $S1_{fils}$ " et " $S2_{fils}$ ".
3. Évaluation des solutions enfants " $S1_{fils}$ " et " $S2_{fils}$ ".
4. Application d'un nombre **nbrMutation** de mutations aux deux solutions fils " $S1_{fils}$ " et " $S2_{fils}$ ", il en résulte les deux solutions mutantes: " $S1'_{fils}$ " et " $S2'_{fils}$ ".

5. Évaluation des solutions obtenues " $S1'_{fils}$ " et " $S2'_{fils}$ ".
6. Insertion des quatre nouvelles solutions " $S1_{fils}$ ", " $S2_{fils}$ ", " $S1'_{fils}$ " et " $S2'_{fils}$ " dans la population \mathbf{P} .
7. Incrémentation de l'indice de génération " t ", afin de passer à la génération suivante.

Parmi notre solution finale, nous sélectionnons les cinq meilleures solutions de la population, pour en calculer la moyenne.

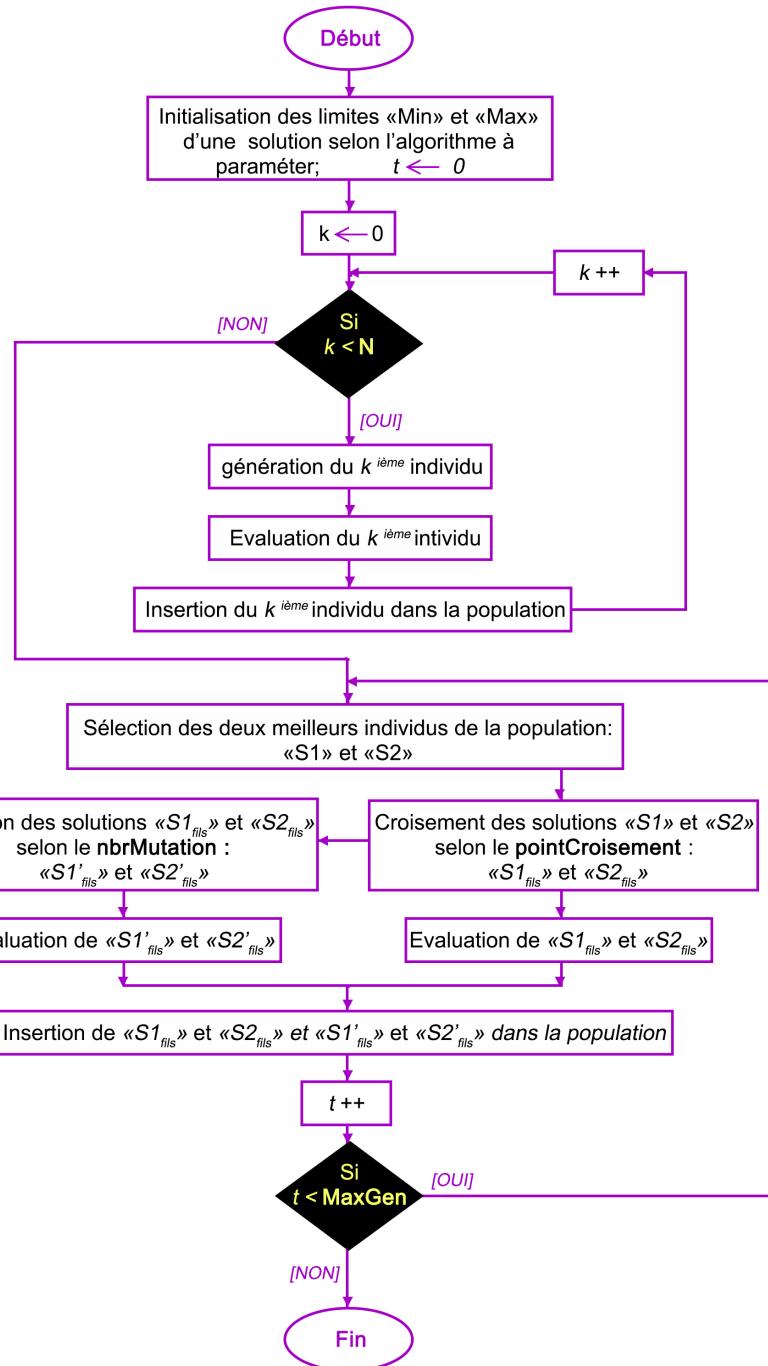


Figure 3.15: Organigramme du mode de fonctionnement de la méthode GA.

3.5 Conclusion

Ce chapitre a fait l'objet de la modélisation de tous les aspects de notre problème, c'est à partir de cette modélisation que nous avons choisi une stratégie d'évitement d'obstacles applicable à toutes nos méthodes de résolution basées essaims. Tous les détails relatifs à son fonctionnement y sont présentés. Enfin, nous avons explicité l'usage de l'algorithme Génétique pour le paramétrage automatique. Ces parties communes à l'ensemble de nos approches de recherche de cibles définies, nous nous attelons à présenter nos approches de recherche de cibles, commençant par BSO dont il est question dans le chapitre suivant.

Chapter 4

Chapitre 4 : BSO pour le problème de recherche de cibles

4.1 Introduction

Dans ce chapitre, nous présentons notre approche de résolution basée sur l'algorithme BSO. Dans un premier temps, le robot simule le comportement d'une seule abeille alors que dans une seconde implémentation, le robot simule tout un essaim d'abeilles, l'approche dans ce cas est appelée Multi-BSO. Le principe de fonctionnement de chaque algorithme, à savoir BSO, Multi-BSO sera détaillé.

4.2 Mono-BSO vs Multi-BSO

les robots constituent un seul groupe qui simule le comportement des abeilles. Il existe donc une communication de type stigmergique entre les robots, concrétisée à l'aide de la simulation de la danse des abeilles. Ce qui amène à déduire qu'il existe une intelligence collective entre les robots déployés et qu'ils interagissent de manière coopérative pour la recherche des cibles.

Nos deux approches BSO et Multi-BSO reposent certes toutes les deux sur les mêmes concepts de base, mais sont utilisées de manières bien différentes, telles que pour:

- **Mono-BSO**, chaque robot se comporte comme une seule abeille. Il occupe une case ayant une certaine position dans l'environnement et peut tester la qualité de sa position en tant que solution, grâce à la fonction objectif.
- **Multi-BSO**, comme son nom l'indique consiste en l'application de BSO par plusieurs essaims d'abeilles. Dans ce cas, chaque robot se comporte comme un essaim d'abeilles lui permettant de se déplacer à la recherche d'une solution optimale.

4.3 Adaptation de BSO pour le problème de la recherche de cibles

Au deuxième chapitre, nous avons exposé l'algorithme générique BSO pour la résolution d'un problème complexe quelconque. Dans cette section, nous nous intéressons à son application au problème de la recherche de cibles. Chaque composant spécifique à l'algorithme comme la solution, la fonction objectif, le paramètre Flip et la table Dance sera adapté à ce problème.

4.3.1 Solution

Une solution est la position de coordonnées (x, y) dans laquelle se trouve une abeille. Cette position correspond à une case ayant une certaine valeur.

4.3.2 Fonction objectif

Chaque abeille est responsable de l'évaluation de sa solution, en vérifiant la valeur contenue dans la case où elle se trouve. Si une cible est détectée la valeur trouvée est alors de 1.

4.3.3 Flip

Il s'agit du paramètre de diversification, dans l'environnement que nous traitons pour ce problème, le **flip** permet de déterminer la zone de recherche globale à partir d'une position initiale I , cela à travers le calcul des positions à une distance R (rayon) de la position I .

$$R = \text{TailleCoté} / \text{flip} \quad (4.1)$$

Avec: TailleCoté : Taille du coté de l'environnement.

Les positions générées constituent le cercle de rayon R ayant pour centre la position initiale I . La figure 4.1 schématisé ce processus.

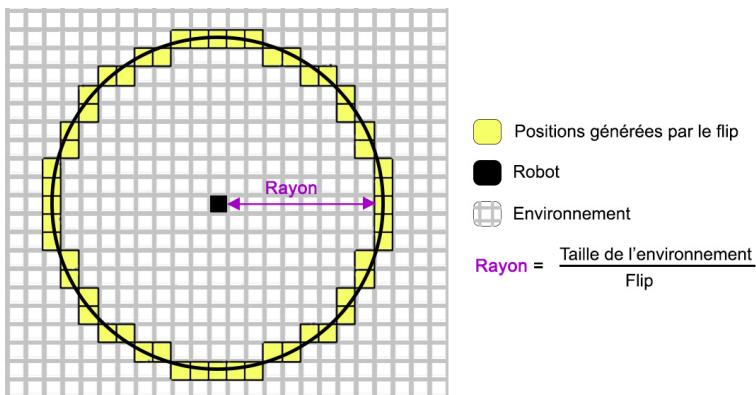


Figure 4.1: Représentation de la méthode de génération de solutions avec le flip.

4.3.4 MaxChance

Une solution possède un nombre maximum de chances pour être sélectionnée comme solution de référence. Ce paramètre est aussi exploité afin de détecter la stagnation.

4.3.5 Table Dance

À chaque itération les abeilles déposent leurs meilleures solutions dans une table appelée "*Table Dance*". Cette table sera exploitée pour choisir la solution de référence de la prochaine itération. Ce choix est basé sur l'un des deux critères suivants:

Critère de qualité (Best In Quality)

Parmi les solutions présentes dans la table **Dance** la meilleure en terme de qualité est celle possédant la valeur la plus élevée après évaluation. Un exemple de sélection de la meilleure solution est illustré dans la figure 4.2 suivante:

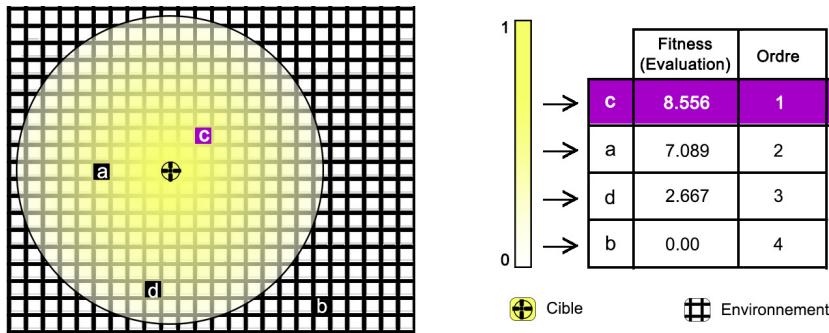


Figure 4.2: Méthode de sélection de la meilleure solution en termes de Qualité.

Critère de Diversité (Best In Diversity)

En cas de stagnation, BSO a recours au choix de la meilleure solution en termes de diversité appartenant à la table *Dance* et ne figurant pas encore dans la liste *Tabou*. Ce mécanisme permet l'exploration des zones distantes tout en évitant de tomber dans un minimum local.

Pour cela nous devons calculer la distance entre deux solutions, la formule choisie est la distance euclidienne, donnée comme suit:

$$\text{Distance}(S_1, S_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (4.2)$$

S_1 : Solution 1 avec coordonnées (x_1, y_1)

S_2 : Solution 2 avec coordonnées (x_2, y_2)

Le choix de la solution la plus diverse se fait en sélectionnant la solution dont la distance minimale par rapport aux autres solutions est maximale.

La méthode de sélection est schématisée dans la figure 4.3 ci-contre:

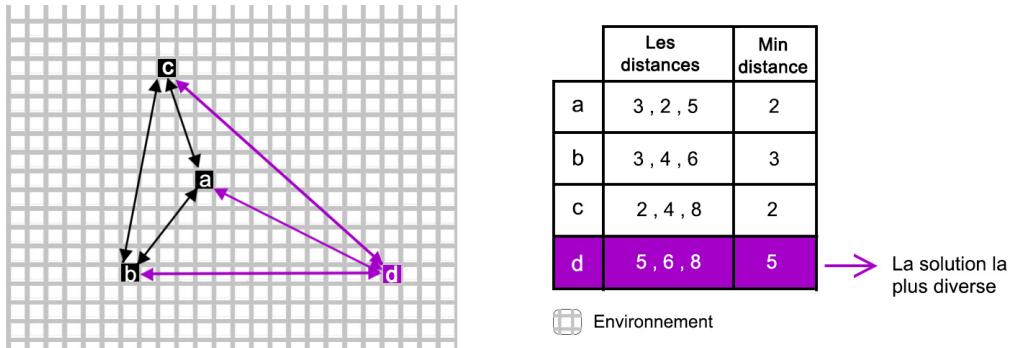


Figure 4.3: Méthode de sélection de la meilleure solution en termes de Diversité.

4.3.6 Fonctionnement du Mono-BSO

L'approche Mono-BSO est constituée d'un seul essaim d'abeilles où chaque robot se comporte comme une seule abeille. Les différentes étapes du Mono-BSO pour la recherche de cibles sont détaillées dans ce qui suit, suivi d'un organigramme 4.4 résumant ces derniers.

Initialisation de la solution de référence (Sref)

Une solution ($\text{position}(x,y)$) est générée aléatoirement et conformément aux contraintes énoncées dans notre modélisation (voir section 3.2.1).

Insertion de Sref dans la liste *Tabou*

La solution de référence (Sref) de l'itération courante " t " sera mise dans une liste *Tabou* qu'on définit comme suit:

Liste Tabou Lorsqu'une position a déjà servi pour une abeille, nous gardons trace son passage dans la liste "*Tabou*", cela permet de réduire la redondance (passage par les mêmes positions).

Génération des zones de recherche

À partir de la solution de référence (Sref) on génère d'autres solutions, ces dernières déterminent des zones équidistantes de Sref. La génération de ces zones de recherche se fait grâce à l'opérateur **Flip**.

Affectation des zones aux abeilles

Une fois les zones générées, On les affecte aux "**nbrBees**" abeilles de l'essaim de façon à ce que chaque abeille soit responsable de sa propre zone.

Recherche locale de chaque abeille

Chaque abeille effectue une recherche locale qui consiste en l'évaluation d'une surface ronde de l'environnement dont le rayon est de vingt cases, cela constitue l'étape d'intensification.

Dans le cas où la recherche locale apporte une amélioration, l'abeille prend cette nouvelle solution et réitère ce processus jusqu'à l'absence d'amélioration.

Déplacement des abeilles

Lors de la recherche locale les abeilles peuvent trouver de meilleures solutions que celles qui leur a été affectées, c'est pourquoi elles se déplacent vers ces solutions plus prometteuses dans l'environnement, ce qui simule le déplacement du robot. Chaque déplacement d'une position (solution) à une autre se fait à travers la stratégie d'évitement d'obstacles (stratégie d'échantillonnage).

Insertion des meilleures solutions dans la table *Dance*

Lorsqu'une abeille termine sa recherche locale elle communique meilleure solution au reste de l'essaim, cela en l'inscrivant dans la table *Dance* commune à l'essaim. Rappelons que le but étant de trouver les cibles donc une bonne solution est celle qui évaluée comme étant proche ou même très proche d'une cible. Lorsqu'une cible est atteinte on incrémente le compteur relatif au nombres de cibles trouvées, et si le nombre objectif de cibles "**nbrCible**" est égalé, la recherche prend fin avec succès.

Choix de la nouvelle Sref

Si à l'itération courant " t " le nombre de cibles recherchées n'est pas atteint, une nouvelle solution de référence doit être choisie pour la prochaine itération.

Il existe alors deux critères possibles, le critère prioritaire est celui de qualité. Une solution prise selon le critère de qualité peut demeurer comme Sref au fil des itérations pour un nombre "**MaxChances**" de fois. Dans le cas où toutes les chances données à cette Sref sont épuisées (stagnation), le choix de la nouvelle Sref se fera selon le second critère, celui de diversité.

Critère d'arrêt de la recherche

BSO continuera de réitérer les étapes citées plus haut à partir de 4.3.6 jusqu'à atteindre le nombre maximum d'itérations "**MaxIter**" ou le nombre de cibles recherchées "**nbrCible**".

Remarque: Contrairement aux autres méta-heuristiques, BSO n'utilise pas seulement la fonction d'évaluation comme critère de sélection de solution, mais aussi la diversité comme nous avons pu le constater dans à l'étape . C'est un mécanisme pour remédier au problème de stagnation dans les minima locaux.

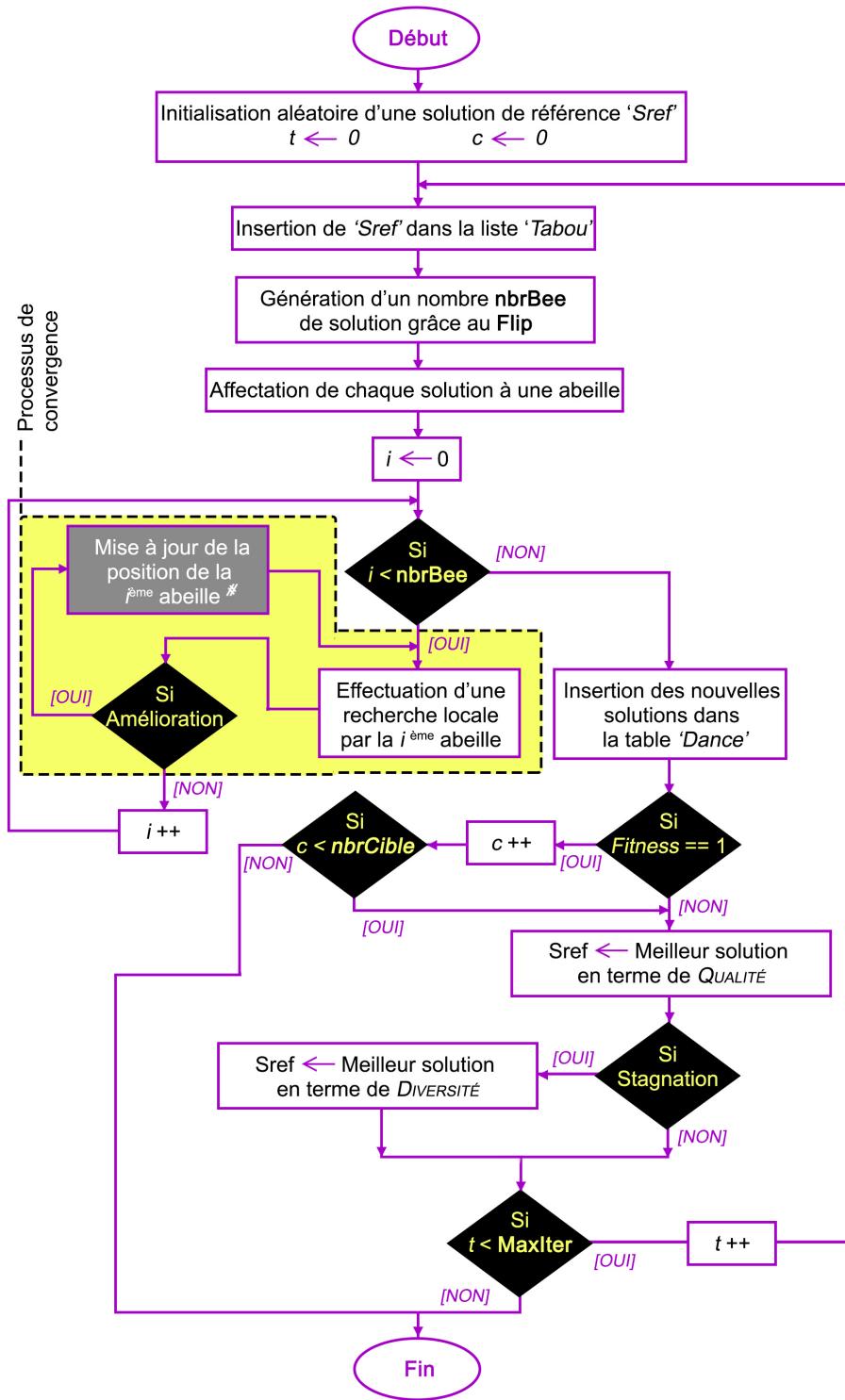


Figure 4.4: Organigramme du mode de fonctionnement de l'approche mono-BSO.

4.4 Multi-BSO : fonctionnement

L'approche Multi-BSO quant à elle, consiste à associer à chaque robot, un essaim d'abeilles. Ils sont donc indépendants les uns des autres, mais ils sont en mode de coopération pour la recherches des cibles. Ils suivent donc le modèle d'interaction du tableau noir des systèmes multi-agents. Le processus de recherche de cibles implémenté à l'aide de Multi-BSO est décrit dans ce qui suit :

4.4.1 Initialisation des solutions de référence (Srefs)

Un nombre "**nbrSwarm**" de solutions (x,y) est généré aléatoirement dans l'espace des solutions, de telle sorte à respecter les contraintes citées dans notre modélisation (voir section 3.2.1). Chaque position Sref est attribuée à un robot.

4.4.2 Insertion de Sref dans la Liste Tabou

Chaque essaim d'abeilles relatif à un robot dépose sa solution de référence (Sref) dans la liste *Tabou*. Cette Liste *Tabou* est commune à tous les robots, elle constitue leur moyen de communication (inter-essaim) permettant de réduire la redondance et repassage sur les mêmes solutions.

4.4.3 Génération des zones de recherche

À partir de la solution de référence (Sref) de chaque robot, les zones de recherche sont générées grâce à l'opération de "**Flip**". Chaque zone est définie par une position dans l'espace de recherche.

4.4.4 Affectation de chaque zone à une abeille

Pour chaque robot disposant de "**nbrBees**" abeilles, chaque zone de recherche calculée précédemment est affectée à une abeille de son essaim, celle-ci y prend position.

4.4.5 Recherche locale pour chaque abeille

Chaque abeille effectue une recherche locale à partir de la zone qu'elle s'est vue affectée, cette recherche consiste en l'évaluation des solutions (positions) de son voisinage à travers la fonction objectif.

À noter que tant qu'une abeille améliore la qualité de sa solution la recherche locale est réitérée de manière récursive.

4.4.6 Insertion des solutions dans les tables *Dance*

Chaque essaim possède sa propre table *Dance*, celle-ci permet la communication intra-essaim de telle sorte que chaque abeille y insère sa meilleure solution locale (de sa zone), afin d'en informer le reste de l'essaim.

4.4.7 Test de la 1^{ère} condition d'arrêt

Si une des abeilles a atteint une cible (solution optimale), le nombre de cibles trouvées " c " est mis à jour. Dans le cas où le nombre de cibles recherchées "**nbrCible**" est égalé, alors la mission touche à sa fin.

4.4.8 Choix de la nouvelle solution de référence

À priori la meilleure solution en matière de qualité est sélectionnée à partir de la table *Dance* comme nouvelle solution de référence (Sref). Sauf en cas de stagnation où une même solution est prise pour solution de référence au delà de "MaxChances" fois, la solution de référence (Sref) est alors choisie selon le critère de diversité.

4.4.9 Déplacement des robots

Chaque robot se déplace de son ancienne position vers la nouvelle position calculées (Sref) en employant notre stratégie d'évitement d'obstacles.

4.4.10 Test de la 2^{ème} condition d'arrêt

Le nombre d'itérations " t " est incrémenté puis comparé au nombre maximum d'itérations "MaxItér" accordé à la recherche.

L'approche Multi-BSO peut alors être perçue comme l'application de Mono-BSO par chaque robot. L'organigramme de son fonctionnement est résumé dans la figure 4.5 ci-dessous.

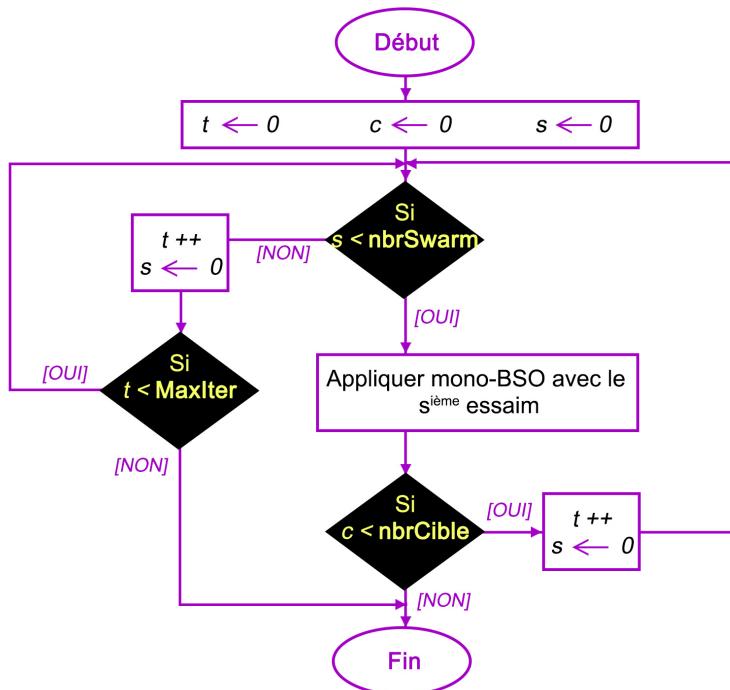


Figure 4.5: Organigramme du mode de fonctionnement de l'approche multi-BSO.

4.5 Conclusion

À travers ce chapitre on a pu mettre en évidence deux approches inspirées du comportement des abeilles BSO et Multi-BSO. Ces algorithmes intelligents reposent sur les mêmes paramètres et fonctions de base avec un mode d'emploi légèrement différent pour Multi-BSO afin de permettre la coordination des essaims (équipes) d'abeilles.

Tout en restant dans la même dynamique des techniques inspirées des comportements d'animaux, nous nous penchons dans le chapitre suivant sur deux approches simulant le comportement des éléphants qui sont très différents des abeilles de par leurs tailles et mécanismes sociaux.

Chapter 5

Chapitre 5 : Algorithmes des éléphants pour le problème de recherche de cibles

5.1 Introduction

La deuxième contribution majeure de ce projet est l'adaptation des deux algorithmes basés essaim d'éléphants vus précédemment, à savoir EHO et EWSA pour le problème de la recherche de cibles. Il sera question dans ce chapitre d'effectuer une adaptation de l'ensemble des concepts de ces deux méthodes de résolution à la modélisation exhibée au niveau du chapitre 3, ainsi que les différentes étapes de leur fonctionnement.

5.2 Adaptation de EHO pour le problème de la recherche de cibles

L'algorithme EHO fut présenté de manière générale au niveau du chapitre 2. Nous allons maintenant nous intéresser à son adaptation au problème de recherche de cibles. Les aspects far de cette approche seront redéfinies comme suit:

5.2.1 Solution

Une solution est la position de coordonnées (x, y) dans laquelle se trouve un éléphant.

5.2.2 Fonction objectif

La fonction objectif est comme définie pour l'approche BSO, à la différence près que pour EHO ce sont les éléphants qui sont responsables de l'évaluation des solutions.

5.2.3 Éléphant

Un éléphant est un robot, il occupe une case unique dans l'environnement à chaque instant " t ", la mise à jour de la position d'un éléphant par rapport à son clan suit l'équation 2.1 pour les deux dimensions (x, y) , comme suit:

$$\begin{aligned} x_{new,c,i} &= x_{c,i} + \alpha * (x_{Best,c} - x_{c,i}) * r \\ y_{new,c,i} &= y_{c,i} + \alpha * (y_{Best,c} - y_{c,i}) * r \end{aligned} \tag{5.1}$$

La figure 5.1 ci-dessous représente l'influence de la meilleure solution sur le déplacement des autres éléphants du clan selon différents degrés α .

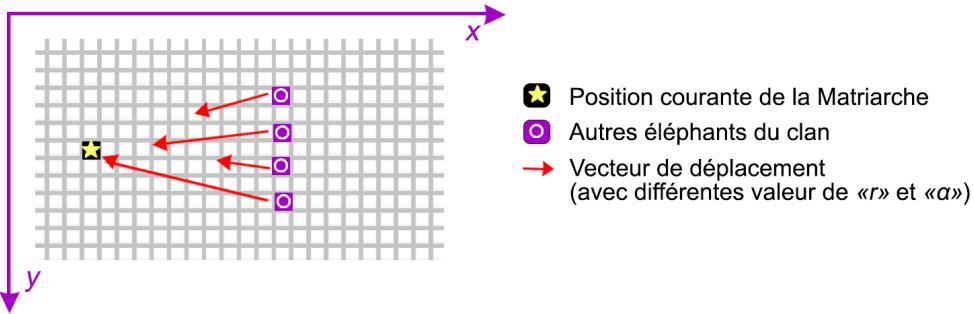


Figure 5.1: Représentation de la méthode de mise à jour des positions des éléphants.

5.2.4 Clan

Les éléphants d'un même clan sont généralement regroupés dans une même zone, ainsi les positions initiales des éléphants sont générées comme suit:

1. Pour chaque clan générer une position initiale aléatoirement.
2. A partir de chaque position générée, positionner les **nbrElephant - 1** autres éléphants sur une surface ronde de rayon inférieur ou égale à une certaine limite (5 cases).

La figure 5.2 ci-dessous, représente un exemple de positions générées pour les éléphants d'un même clan:

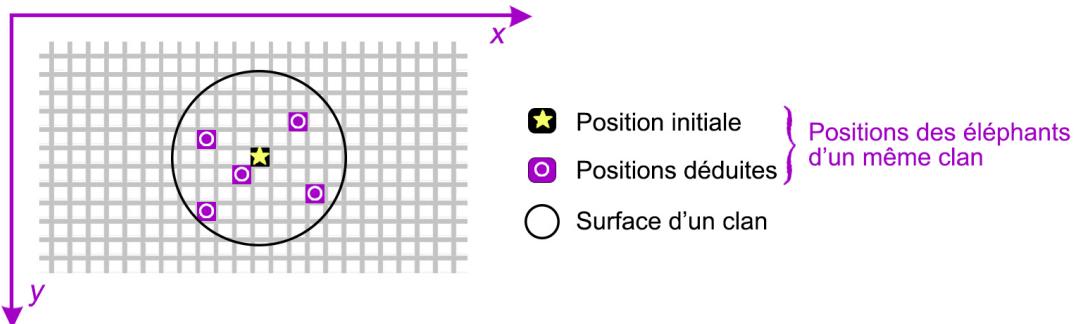


Figure 5.2: Représentation de technique de génération des positions des éléphants d'un même clan.

5.2.5 Chef de clan (Matriarche)

L'éléphant chef de clan appelé "Matriarche" est celui qui possède la meilleure solution de son clan, la mise à jour de cette position dépend du centre de gravité du clan, comme présentée dans l'équation 2.2, elle est appliquée de la même manière à chacune des coordonnées *x* et *y*, comme suit:

$$\begin{aligned} x_{Best,c} &= \beta * x_{center,c} \\ y_{Best,c} &= \beta * y_{center,c} \end{aligned} \quad (5.2)$$

La figure 5.3 montre le processus de mise à jour de la position de la "Matriarche".

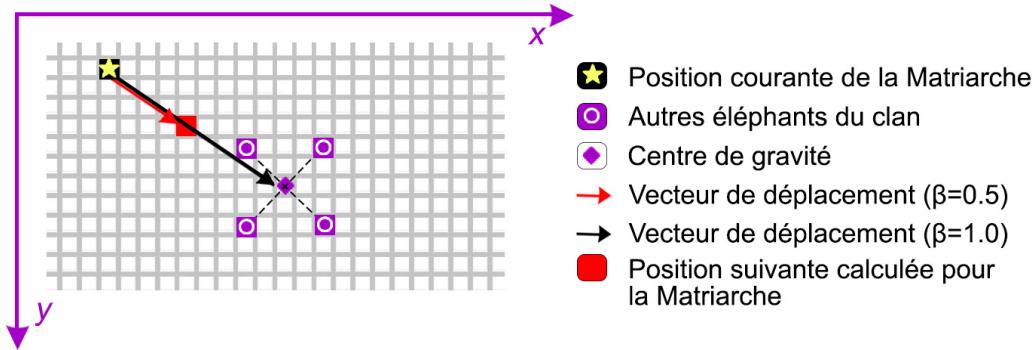


Figure 5.3: Représentation de la méthode de mise à jour de la position du meilleur éléphant/robot.

5.2.6 Éléphant mâle

Dans chaque clan, l’éléphant évalué comme détenant la pire solution est appelé “*pire éléphant*”, il possède une fonction de mise à jour particulière décrite par l’équation 2.4, qui pour notre environnement 2D de taille $taille_{Coté}$ est adaptée par l’équation:

$$\begin{aligned} x_{Worst,c} &= (taille_{Coté} - 1) * rand \\ y_{Worst,c} &= (taille_{Coté} - 1) * rand \end{aligned} \quad (5.3)$$

Celle-ci favorise la diversification afin d’explorer une plus grande surface de l’environnement. Un exemple d’application de cette équation est présentée dans la figure 5.4 qui suit:

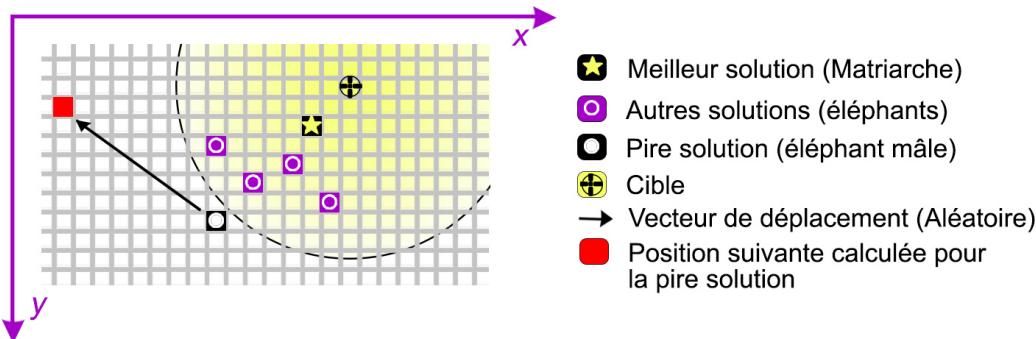


Figure 5.4: Représentation de la méthode de mise à jour de la position du pire éléphant/robot.

5.3 EHO : Fonctionnement

La méthode EHO repose sur la présence de plusieurs clans de robots éléphants où chaque clan possède un chef aussi appelé “*meilleur éléphant*” ou “*Matriarche*” et un pire éléphant devant se séparer du clan. Les différents clans coopèrent dans le but commun de recherche de cibles, deux types de communication sont utilisés, d’abord la communication entre robot d’un même clan sur les meilleures solutions trouvées, et en second lieu la communication entre clans relative au nombre de cibles, celle-ci faisant office de tableau noir. L’organigramme de la figure 5.5 décrit son fonctionnement.

5.3.1 Initialisation des positions des clans

Pour commencer, nous initialisons les positions des "**nbrClan**" clans de manière aléatoire dans notre espace des solutions, toujours en respectant les contraintes concernant les bornes et les obstacles de ce dernier.

5.3.2 Initialisation des positions des éléphants

À partir des positions des clans, sont déduites les positions des "**nbrEle**" éléphants de chaque clan comme décrit dans la section 5.2.4.

5.3.3 Évaluation des solutions

Chaque clan "*Cl*" se voit évaluer les solutions que portent tous ses éléphants. Dès qu'une nouvelle cible est trouvée, le nombre de cible détectée "*c*" est incrémenté. Ainsi si le nombre total de cibles "**nbrCible**" de notre environnement est atteint, la mission de recherche est un succès et donc interrompue.

5.3.4 Tri des clans

Chaque clan parmi les "**nbrClan**", procède au tri de ses éléphants selon la fonction objective dans l'ordre décroissant de ses valeurs.

5.3.5 Mise à jour des positions des éléphants

Pour chaque éléphant "*ele*" du clan "*Cl*" on détermine la nouvelle position, pour cela deux paramètres sont nécessaires: La position du chef du clan et le paramètre empirique α .

La nouvelle position du meilleur éléphant de chaque clan est calculée à partir du centre de gravité du clan CG_{Cl} et du paramètre empirique β .

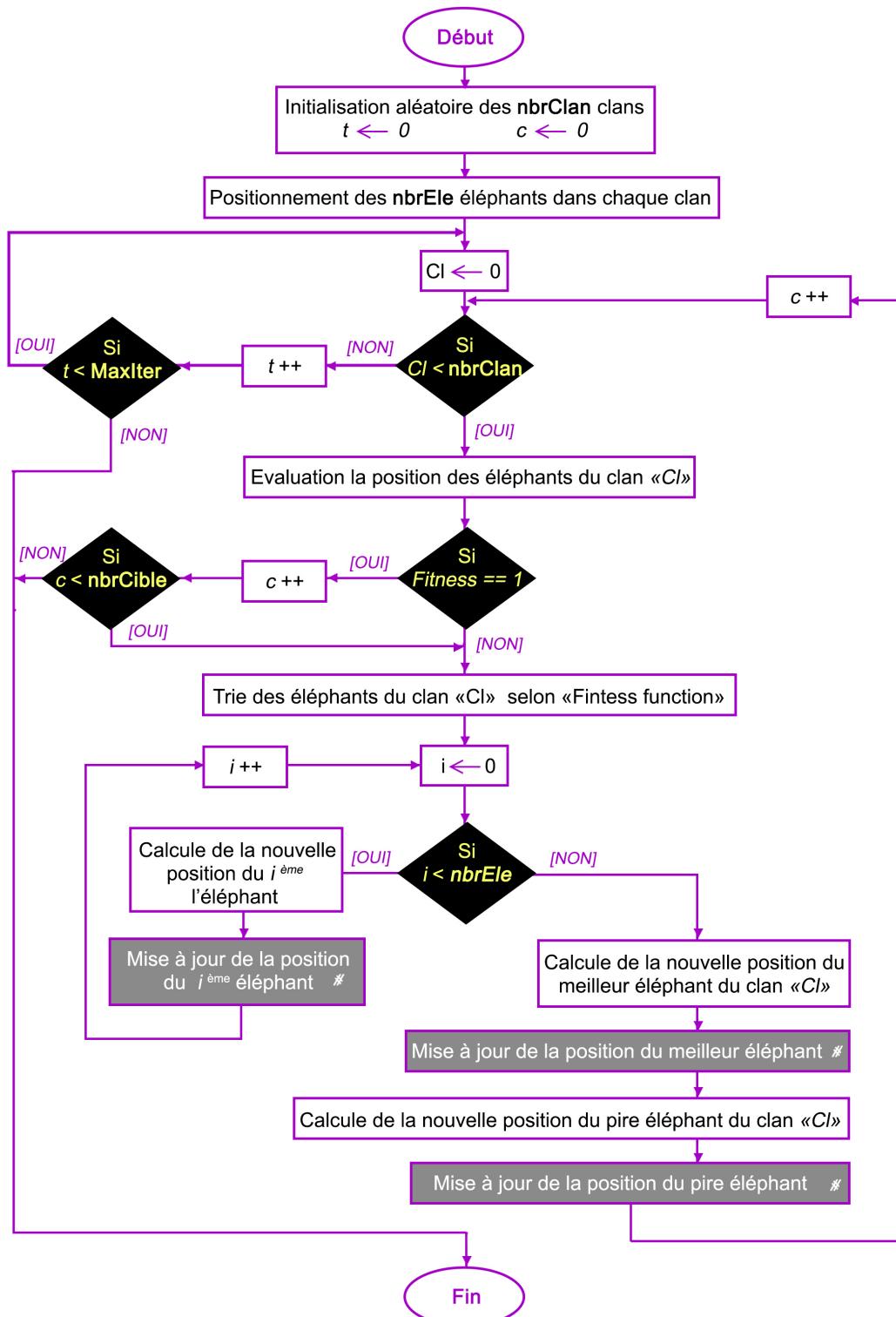
Génération de la prochaine position du pire éléphant de chaque clan grâce à l'opérateur de séparation de base aléatoire.

5.3.6 Déplacement des robots

Chaque robot éléphant "*ele*" se déplace vers la nouvelle position calculée via notre processus d'évitement d'obstacles.

5.3.7 Test de la condition d'arrêt

Incrémenter le nombre d'itérations effectuées "*t*". Le processus de recherche prend fin si le nombre maximum d'itérations "**MaxIter**" est atteint. Dans le cas contraire le processus reprend à partir de l'étape d'évaluation des solution (section 5.3.3).



* Appel de la stratégie d'évitement d'obstacle avec l'ancienne et la nouvelle position calculée

Figure 5.5: Organigramme du mode de fonctionnement de l'approche EHO.

5.4 Adaptation de ESWSA pour le problème de la recherche de cibles

5.4.1 Solution

Tout comme EHO, une solution pour ESWSA est la position de coordonnées (x, y) occupée par un éléphant se comportant comme un robot.

5.4.2 Fonction objectif

La fonction objectif est la même que pour EHO, où les éléphants sont responsables de l'évaluation des solutions.

5.4.3 Éléphant

Un éléphant simule un robot, il occupe une case dans la grille(environnement) ayant une position connue, il est doté d'une mémoire qui englobe sa meilleure position personnelle $pbest$ et la meilleure position globale $gbest$ du groupe d'éléphants. La dimension du vecteur de positions est alors égale à deux.

$$X_{i,2} = (x_{i1}, x_{i2}) \quad (5.4)$$

Pour des soucis de compréhension on a choisi la notation (x, y) avec i le numéro de l'éléphant

$$X_{i,2} = (x_i, y_i) \quad (5.5)$$

5.4.4 Outils de mémoire

"Pbest" dite aussi "personal best", spécifique à chaque éléphant de l'essaim, représente la position de l'éléphant dans l'environnement qui a maximisé la fonction objectif tout au long de la recherche donc celle qui est la plus proche d'une cible particulière, c'est la meilleure solution en qualité de l'éléphant.

$$\begin{aligned} Pbest_i &= (Pbest_{xi}, Pbest_{yi}) \\ Pbest_i^t &= \max_{1 \leq j \leq t} (f(X_i^j)) \end{aligned} \quad (5.6)$$

La mise à jour du $Pbest$ d'un éléphant au fil des itérations est représenté dans la figure 5.6 suivante:

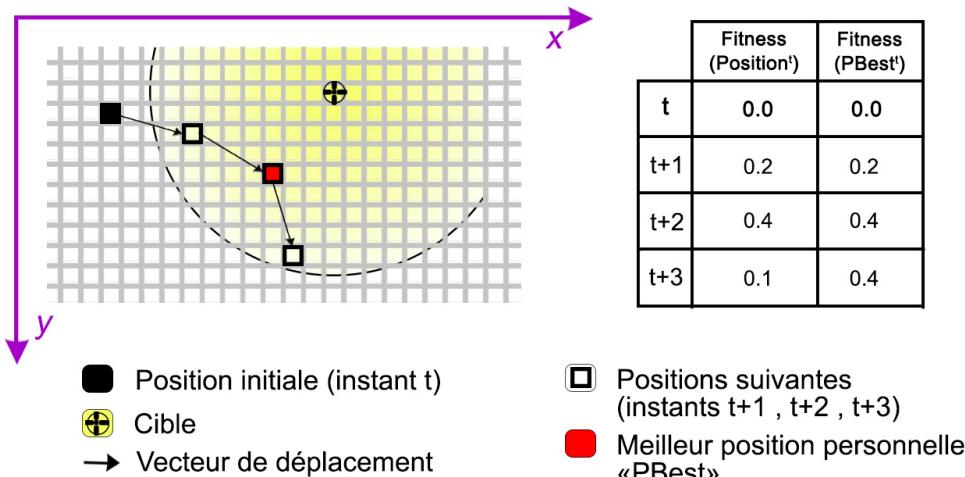


Figure 5.6: Représentation de la méthode de mise à jours du Pbest d'un éléphant.

Global best () ou encore meilleure position globale, est une position spécifique à l'essaim, c'est la meilleure position trouvée par l'essaim et par conséquent la plus proche par d'une certaine cible de tout l'essaim. On peut dire que le *Gbest* consiste à trouver le max en termes de qualité des *Pbest* de chaque éléphant.

Chaque éléphant a connaissance de la valeur de *Gbest* à chaque itération, c'est une connaissance commune à tous.

$$\begin{aligned} Gbest &= (Gbest_x, Gbest_y) \\ Gbest^t &= \max(f(Pbest_i^t)) \end{aligned} \quad (5.7)$$

5.4.5 Vélocité

la vélocité combine la vitesse et la direction qui détermine la prochaine position de l'éléphant. La notation de la vélocité dans un environnement bidimensionnelle est la suivante:

$$V_{i,2} = (v_{xi}, v_{yi}) \quad (5.8)$$

5.4.6 Mécanisme de déplacement

Mise à jour de la vélocité de l'éléphant

La mise à jour de la vélocité est sujette à plusieurs paramètres dont la position *Pbest* et *Gbest*, explicité dans la figure 5.7, selon la constante *p* l'éléphant choisira de suivre sa meilleure position ou la meilleure du groupe, ce qui lui permet d'éviter de stagner dans des optimums locaux.

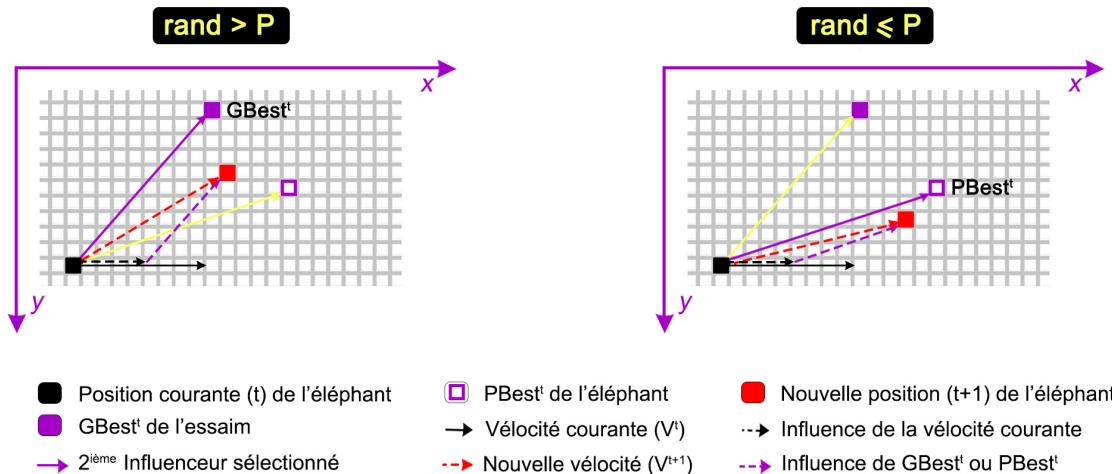


Figure 5.7: Représentation de la mise à jour de la vélocité d'un éléphant dans l'approche EWSA.

Vu qu'on se trouve dans un environnement bi-dimensionnelle on devra calculer deux vélocités, une selon les abscisses et une selon les ordonnées. Le calcul de celle des abscisses se fait avec la première composantes des positions $X_{i,2}$, $Pbest_{i,2}$, $Gbest_{i,2}$, alors que celle des ordonnées se fait avec leur seconde composante on explicite dans les deux équations qui suivent le calcul de la vélocité.

$$v_{xi}^{t+1} = \begin{cases} v_{xi}^t * w^t + rand * (Gbest_x^t - x_i^t) & \text{si } random > p \\ v_{xi}^t * w^t + rand * (Pbest_{xi}^t - x_i^t) & \text{si } random \leq p \end{cases} \quad (5.9)$$

$$v_{yi}^{t+1} = \begin{cases} v_{yi}^t * w^t + rand * (Gbest_y^t * y_i^t) & \text{si } random > p \\ v_{yi}^t * w^t + rand * (Pbest_{yi}^t * y_i^t) & \text{si } random \leq p \end{cases} \quad (5.10)$$

Notre environnement a des bornes fixes, si on choisit de garder cette formule de calcul pour la vélocité on risque de tomber dans des vélocités trop grandes ce qui perturbe l'organisation et le comportement du groupe d'éléphants. Une solution est de borner la vélocité. Soient les bornes de l'environnement représentées par le point $MaxP$

$$\begin{aligned} MaxP &= (MaxP_x, MaxP_y) \\ v_{xi}^{t+1} &= v_{xi}^{t+1} \bmod MaxP_x \\ v_{yi}^{t+1} &= v_{yi}^{t+1} \bmod MaxP_y \end{aligned} \quad (5.11)$$

Mise à jour de la position de l'éléphant

Tout comme la vélocité la mise à jour de la position $X_{i,2}$ de l'éléphant i se fait sur deux niveaux celui des abscisses et celui des ordonnées, l'équation de mise à jour est la suivante.

$$\begin{aligned} x_i^{t+1} &= v_{xi}^{t+1} + x_i^t \\ y_i^{t+1} &= v_{yi}^{t+1} + y_i^t \end{aligned} \quad (5.12)$$

Toujours pour éviter les valeurs aberrantes on procède de la même manière qu'avec la vélocité en bornant la position soit :

$$\begin{aligned} MaxP &= (MaxP_x, MaxP_y) \\ x_i^{t+1} &= x_i^{t+1} \bmod MaxP_x \\ y_i^{t+1} &= y_i^{t+1} \bmod MaxP_y \end{aligned} \quad (5.13)$$

Ainsi la position de l'éléphant est calculée en tenant compte de sa connaissance qui est sa position à l'instant t afin de calculer celle de l'instant $t + 1$.

Mise à jour de w^t (poids d'inertie)

Le poids d'inertie est un paramètre qui influence la vélocité et ainsi le calcul des positions des éléphants. Comme ça a été explicité dans le chapitre 2 voir section 3 page 21, l'équation choisie calcule le poids d'inertie selon le nombre d'itérations t avec l'initialisation suivante:

$$\begin{aligned} w_{max} &= 0.9 \\ w_{min} &= 0.1 \\ t_{max} &= 1000 \\ w^t &= w_{max} - \frac{w_{max} - w_{min}}{t_{max}} * t \end{aligned} \quad (5.14)$$

w^t joue un rôle important dans la convergence des éléphants vers la solution, il détermine le taux de vélocité mémoire, la vélocité à l'instant t , qui influencera la prochaine vélocité à l'instant $t + 1$. Il permet alors de contrôler et varier la vitesse et la trajectoire afin d'explorer au mieux l'environnement de recherche.

5.5 EWSA : Fonctionnement

Dans EWSA chaque éléphant se comporte comme un robot indépendant du groupe mais coopératif, ayant une mémoire qui se présente en sa *pbest* et *gbest* qui sont nécessaire pour lui lors des choix de trajectoires possibles. L'organisation des éléphants lors de la recherche suit un schéma dont on explicite les étapes dans la figure 5.8 qui résume le processus suivant :

5.5.1 Initialisation

Comme toute mét-heuristique, EWSA commence par une étape d'initialisation déterminante pour le déroulement de la recherche à savoir :

Paramètres empiriques

Cette étape consiste à choisir les bonnes valeurs des paramètres empiriques, ils sont souvent le sujet d'un réglage pour augmenter l'efficacité et performance des approches basées essaim. Les paramètres de EWSA sont : T_{max} , $nbrEle$, P , W^t , $MaxP$, $MinP$.

Positions initiales des éléphants

Les positions initiales des éléphants sont aussi un facteur influant de la recherche. On initialise alors chaque éléphant à une position aléatoire appartenant à l'environnement respectant les contraintes énoncées dans le chapitre 3.

Pbest pour chaque éléphant

Après avoir choisi les positions des éléphants, on doit alors évaluer la qualité de cette position. Cette première évaluation permet d'initialiser les *Pbest* (meilleure position personnelle en termes de qualité) de chaque éléphant de l'essaim.

Gbest

Comme on a pu l'expliquer plus haut dans 5.4.4, la position *Gbest* (meilleure position globale de tout l'essaim) est initialisée en trouvant le maximum des positions *pbest* évaluées en termes de qualité.

Une fois les initialisations faites le processus de EWSA se met en marche selon ce qui suit :

5.5.2 Mise à jour de la vitesse pour chaque éléphant

Chaque éléphant de l'essaim se voit mettre à jour sa vitesse, cela se fait avec *le choix d'un nombre aléatoire* qui détermine l'équation à choisir. Si ce nombre aléatoire est inférieur au paramètre empirique *p* alors la vitesse sera calculée par rapport à sa position *Pbest*, sinon ca sera selon le *Gbest*. Autrement dit l'éléphant choisira de suivre sa position *Pbest* ou bien *Gbest* à chaque itération.

5.5.3 Calcul de la prochaine position de chaque éléphant

Une fois la vitesse calculée, elle nous permet de mettre à jour la position de chaque éléphant selon l'équation de mise à jour, en additionnant la vitesse à la position précédente de l'éléphant.

5.5.4 Déplacement des robots

Vu qu'un éléphant simule un robot, son déplacement se fait en employant la stratégie d'évitement d'obstacles en cas de présence d'obstacles dans l'environnement.

5.5.5 Évaluation des positions de chaque éléphant

Un éléphant a un angle de vue lui permettant de voir autours sur une superficie de dix cases, ces cases seront évaluées selon la fonction objectif, l'éléphant gardera la position de la meilleure case.

5.5.6 Mise à jour de Pbest de chaque éléphant

Pendant la recherche, la position *Pbest* est susceptible de changer c'est pourquoi après chaque évaluation un test est effectué pour savoir si on doit mettre à jour la *pbest* de l'éléphant c'est à dire qu'on a trouvé une meilleure position nous rapprochant d'une cible.

5.5.7 Évaluation des positions des éléphants

Un éléphant a un angle de vue lui permettant de voir autours sur une superficie de dix cases, ces cases seront évaluées selon la fonction objective, l'éléphant gardera la position de la meilleure case.

Si une cible est détecté, le nombre de cibles trouvées "c" est alors incrémenté.

5.5.8 Mise à jour du Pbest de chaque éléphant

Pendant la recherche, la position *Pbest* est susceptible de changer, c'est pourquoi après chaque évaluation un test est effectué pour savoir si la mise à jour du *Pbest* de l'éléphant est nécessaire. Cela signifie qu'on a trouvé une meilleure position nous rapprochons d'une cible. »»»> 6a4c1102eced6c45748b34579ef1a970fd34048f

5.5.9 Mise à jour de Gbest

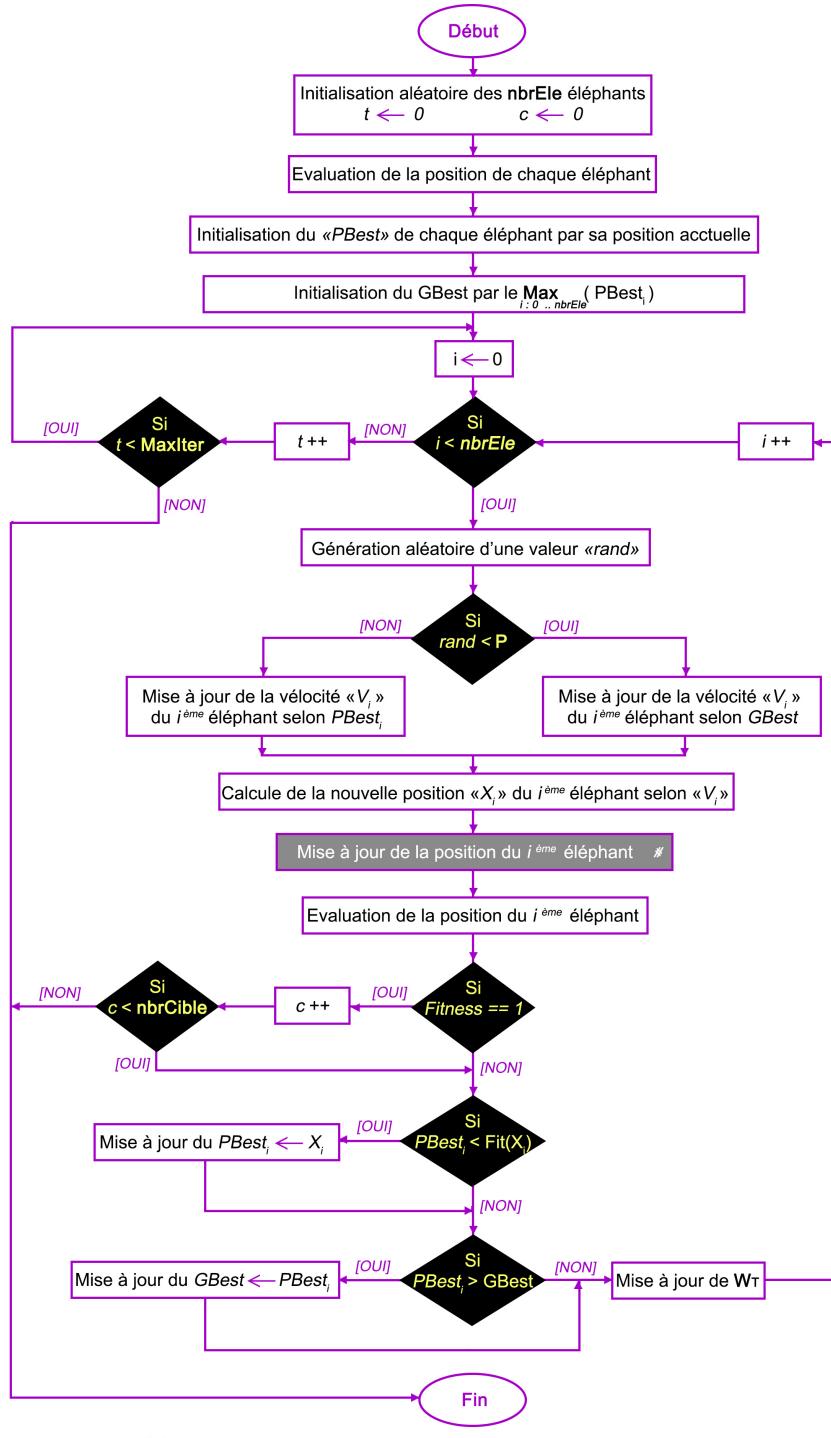
Après l'éventuelle mise à jour des positions *Pbest*, vient la mise à jour de la position *Gbest*, où un test est effectué également pour un potentiel changement du *Gbest*.

5.5.10 Mise à jour du W^t (poids d'inertie)

Le poids d'inertie est relatif à chaque itération dont on calcule sa valeur selon l'itération courante.

5.5.11 Critère d'arrêt de la recherche

Le processus de ESWSA s'exécute en boucle, en incrémentant le nombre d'itération "*t*", jusqu'à la rencontre de toutes les "nbrCible" cibles ou bien jusqu'à atteindre le nombre maximum d'itérations "MaxIter".



Appel de la stratégie d'évitement d'obstacle avec l'ancienne et la nouvelle position calculée

Figure 5.8: Organigramme du mode de fonctionnement de l'approche EWSA.

5.6 Conclusion

Les deux algorithmes présentés reposent sur le comportement des éléphants, mais vu sous deux angles bien différents. C'est pourquoi leur fonctionnement n'a rien en commun exploitant chacun un aspect différent relatif aux éléphants et groupes d'éléphants. Le dernier chapitre sera consacré à la réalisation et expérimentations de toutes des approches citées dans les chapitres précédents.

Chapter 6

Chapitre 6 : Validation Expérimentale

6.1 Introduction

Les expérimentations sont essentielles pour la validation de nos algorithmes et approches, cette étape nécessite de déterminer les caractéristiques des machines utilisées ainsi que la description de la partie logicielle dont nous avons eu besoin, mais surtout une bonne organisation des expérimentations. C'est justement l'objet de ce chapitre, qui est structuré comme suit:

- Description de l'environnement de développement.
- Expérimentations relatives au paramétrage des approches de recherche.
- Expérimentations sur les environnements, dont l'organisation est détaillée, suivie des résultats obtenus regroupés par types d'expérimentations et par types d'environnements.
- Présentation du simulateur accompagné de son manuel d'utilisation.

6.2 Environnement de développement

Comme pour toute expérimentation l'environnement de test est un élément clé pour la bonne interprétation des résultats obtenus.

6.2.1 Matériel

Nous avons utilisé deux machines (Laptop) possédant les caractéristiques et capacités suivantes:

<u>Machine 1 :</u>	<u>Machine 2 :</u>
Mémoire RAM: 8 Go	Mémoire RAM: 8 Go
Processeur: Intel® Core™ i5-6200U CPU 2.30GHz x 4	Processeur: Intel® Core™ i5-3317U @ 1.70Hz 1.70GHz
Carte graphique: Intel® HD Graphics 520 (Skylake GT2) x86/MMX/SSE2	Carte graphique: Intel® HD Graphics 520 .
Type de l'OS: UBUNTU 14.04 LTS 32-bit	Type de l'OS: UBUNTU 18.04 LTS 64-bit

6.2.2 Logiciels

Le développement de nos algorithmes a été exclusivement fait sous système d'exploitation **Linux** (*Ubuntu*), celui-ci munit des logiciels suivants:

IntelliJ IDEA est un environnement de développement de logiciels informatiques orienté **Java**, supportant une large palette de plugins. C'est un des nombreux produits développés par la compagnie *JetBrains*. celle-ci nous a été mise à disposition via une licence étudiant.

Nous l'avons utilisé pour le développement de notre solution de recherche de cibles.

- **Java** est un langage orienté objet académique qui a fait ses preuves, très largement utilisé pour ses nombreux avantages dont on cite: la portabilité, large diversité des librairies, conséquente documentation, ...etc.

Nous avons eu recours à une bibliothèque seulement, cela pour l'interface graphique en (**javaFx**) de notre simulateur en Java.

jfoenix-8.0.8.jar JFoenix¹ est une librairie mettant à disposition des composantes java qui implémentent Google Material Design, elle est open source, et dédiée aux applications java, spécialement les interfaces *javaFx*.

PyCharm est un environnement de développement de logiciels informatiques sous langage **Python**, aussi fournis par la compagnie *JetBrain*, nous avons obtenu ce logiciel sous licence étudiant. Nous l'avons principalement utilisé pour exploiter les résultats des expérimentations pour les traiter et traduire en graphe plus significatifs.

Python est un langage de programmation interprété qui a plus ou moins récemment fait surface. Il est de plus en plus utilisé en vue de sa simplicité et polyvalence.

6.3 Expérimentations relatives aux approches

6.3.1 Paramétrage du mini-GA incrémental

Notre Algorithme Génétique possède quatre paramètres empiriques comme décrits dans le chapitre 3, nous avons procédé à une suite de tests afin de trouver les meilleures valeurs de ces paramètres, sans passer par un réglage de paramètres exhaustive. . Pour cela nous avons fixé les bornes de chaque paramètre en prenant en compte la taille d'une solution (3D ou 4D).

Les meilleurs paramètres obtenus se présentent comme suit:

- Taille de la population : 20
- Nombre d'itérations : 30
- Nombre de mutations : 2
- Point de croisement : aléatoire.

6.3.2 Paramétrage des approches de recherche de cibles

Pour chaque taille d'environnement , portée des cibles , nombre de cibles nous avons paramétré nos méthodes à l'aide de l'algorithme génétique (mini-GA incrémental), la moyenne des cinq meilleurs paramètres obtenus est calculée afin de former le meilleur paramétrage

¹Lien : <https://github.com/jfoenixadmin/JFoenix>

Réglage de Paramètres de BSO

Flip, pour cette approche, nous avons constaté durant le réglage de ses paramètres que la variable *Flip* influence grandement l'évolution de la recherche. Une valeur de *Flip* trop grande empêche les abeilles d'explorer les zones distantes, de ce fait elle augmente les risques de stagnation, contrairement à des valeurs de *Flip* trop petites qui favorisent l'exploration en dispersant trop les abeilles, ce qui peut mener à une recherche désorganisée.

nbBees: Le nombre d'abeille *nbBees* est un paramètre à double tranchant, car un grand nombre d'abeilles peut certes augmenter les chances de réussite, mais il amplifie considérablement le temps d'exécution.

maxChances prit avec une valeur trop grande cause une perte de temps et d'effort des abeilles, ce qui favorise une stagnation. En revanche une valeur trop petite augmente les chances de passer à coté d'une solution intéressante.

Les meilleurs paramètres obtenus pour l'algorithme BSO sont présentés dans le tableau suivant:

Réglage de Paramètres de Multi-BSO

Flip, ce paramètre dans l'approche Multi-BSO ne doit pas être trop petit car cela causerait une exploration trop grande et les abeilles se verront dispersées et perdraient leur organisation en groupes, d'autre part des valeurs exagérément grandes ralentiraient l'avancement de la recherche.

nbSwarms & nbBees & MaxChances: cette méthode se distingue de la précédente par le nombre de groupes d'abeilles *nbSwarms*, dont les valeurs ont le même impacte que le nombre d'abeilles *nbBees* dans BSO.

D'ailleurs le nombre d'abeilles *nbBees* par groupe et le paramètre *MaxChance* ont tous deux le même effet sur l'évolution de la recherche que sur BSO.

Le tableau ci-dessous résume les meilleurs paramètres trouvés pour l'approche Multi-BSO:

Réglage de Paramètres de EHO

nbClan & nbEle Le nombre de clan **nbClan** et nombre d'éléphants par clan **nbEle** influencent directement les temps d'exécution et l'efficacité de la recherche, car des valeurs trop grandes (trop de clans et d'éléphants) augmentent les temps d'exécution, par contre leur affecter des valeurs trop petites réduit considérablement l'efficacité d'EHO.

Alpha (α) D'une part, si le paramètre α prend des valeurs trop grandes, les éléphants convergeront vers des positions proches de la meilleure solution trop vite, cela engendrera leur stagnation dans un minimum locale. D'autre part, une valeur trop petite ralentirait le processus de recherche car les déplacements entre deux itérations seront très petits.

Beta (β) Enfin, une valeur trop proche du 1 pour β peut empêcher la convergence vers la solution optimale, car l'éléphant *matriarche* sera retenu par le centre de gravité du clan qui n'est pas forcément dans la direction de la meilleure solution. Par contre avec une valeur trop petite (proche du 0) l'éléphant se détachera de son clan, ce qui causera de grands écarts dans les déplacements.

Les meilleurs paramètres de l'algorithme EHO sont donnés dans le tableau qui suit:

Réglage de Paramètres de EWSWA

Wt : Nous avons remarqué que quand le nombre d'itérations t augmente, la quantité du poids d'inertie W^t diminue. Autrement dit plus les éléphants avancent (nombre d'itérations) plus l'influence de l'ancienne vitesse diminue ce qui permet une meilleure exploration.

vélocité : L'initialisation de la vitesse influe sur la dynamique de la recherche. Car une initialisation aléatoire non nulle $V_i = (valx, valy)$ à de grandes valeurs, disperse les éléphants dans l'environnement. Contrairement à une initialisation nulle $V_i = (0,0)$ qui permet des déplacements à pas progressifs.

L'initialisation à valeurs aléatoires bornées moyennes a été la plus satisfaisante car elle privilégie l'exploration au début de la recherche.

p : Lors du paramétrage à l'aide du GA, nous avons remarqué que lorsque p est très petit, chaque éléphant ne prend en considération que sa meilleure solution personnelle ($Pbest_i$) cela ralentit le processus de recherche de cibles, car les éléphants ont tendance à repasser par les mêmes chemins et parfois stagner dans leur optimum local, comme le montre les figures 6.1, 6.2 et 6.3 pour une valeur de $p = 0$.

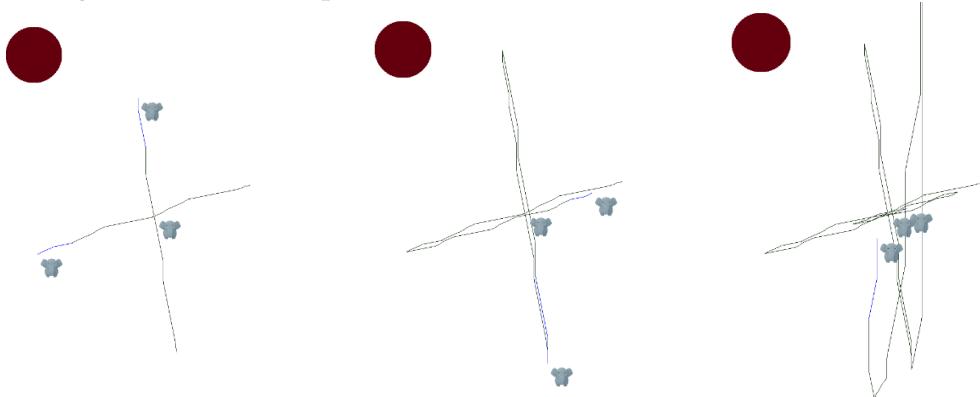


Figure 6.1: EWSWA à l'itération 4

Figure 6.2: EWSWA à l'itération 8

Figure 6.3: EWSWA à l'itération 16

Contrairement à une grande valeur de p , les éléphants suivent la même meilleure solution globale ($Gbest$), convergeant ainsi vers une même cible ce qui cause un manque d'intensification ne profitant pas de la meilleure solution personnelle de chaque éléphant ($Pbest_i$).

Dans le tableau ci-contre, sont exhibés les meilleurs paramètres obtenus pour l'algorithme EWSWA:

6.4 Expérimentations relatives à l'environnement

Nous avons sélectionné trois paramètres à expérimenter pour observer le comportement de nos approches de recherche basées essaims. Ces paramètres sont : la portée des cibles, la taille de l'environnement et le nombre de cibles recherchées. Nous avons effectué ces expérimentations pour trois types d'environnements soient : simples, avec obstacles et complexes.

6.4.1 Organisation des types d'expérimentations

Pour une question de clarté, nous avons jugé nécessaire de décrire comment est organisé chaque type d'expérimentation avons de passer aux résultats obtenus.

Organisation des expérimentations par rapport à la portée des cibles

Pour les expérimentations relatives à l'influence de la portée des cibles sur le comportement de nos algorithmes de recherche, nous avons fixé la taille de l'environnement à 500×500 positions et fait varier la portée d'un rayon de 10 positions à 100 positions avec un pas de 10.

Ainsi comme le montre la figure 6.4, les tests de ce type proviennent des exécutions sur 40 environnements (4 par portée), soit 400 exécutions pour le mono-cible et 400 autres pour le multi-cibles, pour chaque méta-heuristique (après paramétrage).

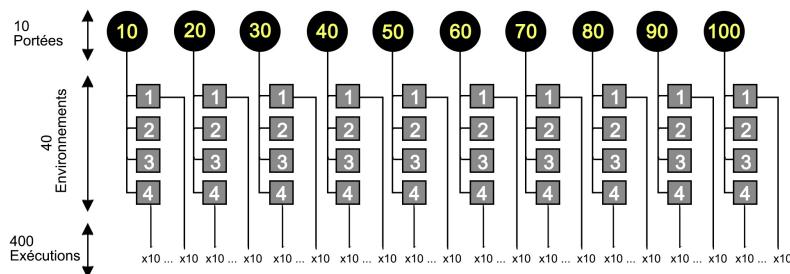


Figure 6.4: Représentation de l'organisation des expérimentations sur la portée des cibles.

Organisation des expérimentations par rapport à la taille de l'environnement

Pour les expérimentations sur l'influence de la taille de l'environnement sur le comportement de nos méta-heuristiques à la recherche de cibles, nous avons adapté la portée des cibles à chaque taille d'environnement conformément à l'équation suivante:

$$portee = \frac{taille_{Coté} \times 10}{100} = \frac{taille_{Coté}}{10} \quad (6.1)$$

Ce qui revient à :

$$S_{cible} = \frac{\sqrt{S_{env}}}{5} \times \pi \quad (6.2)$$

Avec :

- $taille_{Coté}$: taille du côté de l'environnement carré.
- S_{cible} : surface d'émission de la cible (rayon = portée).
- S_{env} : surface de l'environnement de recherche ($S_{env} = taille_{Coté}$).

Pour cela nous avons sélectionné dix différentes tailles d'environnement, les tailles du côté de ces environnements sont comme suit : 50, 100, 200, 400, 600, 800, 1000, 1500, 3000, 5000.

les surfaces respectives sont les suivantes: 2500, 10000, 40000, 160000, 360000, 640000, 1000000, 2250000, 9000000, 25000000.

Ainsi les portées correspondantes sont: 5, 10, 20, 40, 60, 80, 100, 150, 300, 500.

Les résultats des tests liés à ce type d'expérimentations sont le résultat de 40 exécutions par taille d'environnement (4 configurations pour chaque taille), ce qui revient à un total de 400 exécutions pour le mono-cible et 400 autres pour le multi-cibles, et ce pour chaque approche (après paramétrage). Le schéma 6.5 ci-dessous illustre cette organisation.

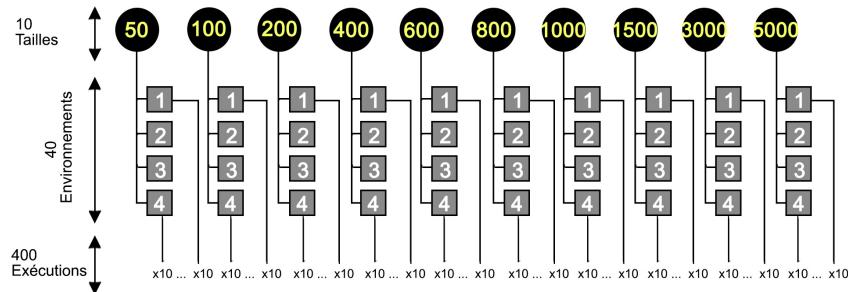


Figure 6.5: Représentation de l'organisation des expérimentations sur la taille des environnements.

Organisation des expérimentations par rapport au nombre de cibles

Ces expérimentations permettent d'étudier l'influence du nombre de cibles présentes dans nos environnements sur le comportement de nos métahéuristiques (BSO, EHO, EWSA et MBSO). Pour cela nous avons fixé la taille des environnements à 500×500 positions ainsi qu'une portée de cible égale à 50 (conformément à l'équation 6.1).

Nous avons pris le nombre de cibles compris entre 1 et 15 (bornes incluses) avec un pas de 2, ce qui nous donne les 8 nombres de cibles suivants : 1, 3, 5, 7, 9, 11, 13, 15.

De ce fait, les résultats des tests sont le fruit d'exécution sur 32 environnements (4 par nombre de cibles), ce qui correspond à 320 exécutions pour chacune des approches développées (après paramétrage).

La figure 6.6 ci-dessous schématisse cette organisation:

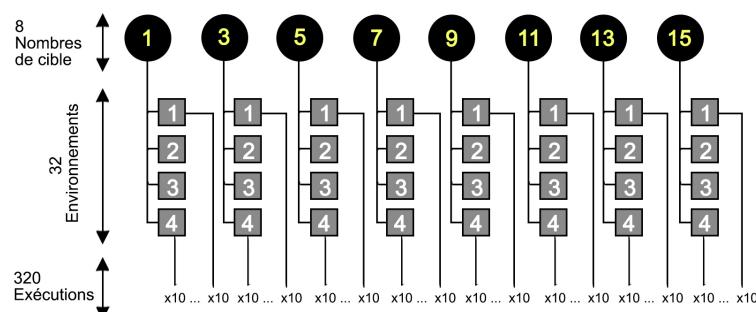


Figure 6.6: Représentation de l'organisation des expérimentations sur le nombre de cible.

6.4.2 Expérimentations par rapport à la portée des cibles

Taux de réussite

La figure 6.7 est un *diagramme à bandes* représentant l'évolution du taux de réussite de chacune de nos méta-heuristiques par rapport aux différentes portées des cibles.

Nous constatons que le taux de réussite est maximal (100%) quelle que soit la valeur de la portée (de 10 à 100), autrement dit les algorithmes implémentés arrivent toujours à trouver la ou les cible(s) sans atteindre le nombre maximal d'itérations qui est de 1000.

Il est à noter que ces mêmes résultats ont été obtenus pour les trois types d'environnements (simple, avec obstacles et complexe) de dimension 500×500 aussi bien dans les cas de mono-cible que de multi-cibles.

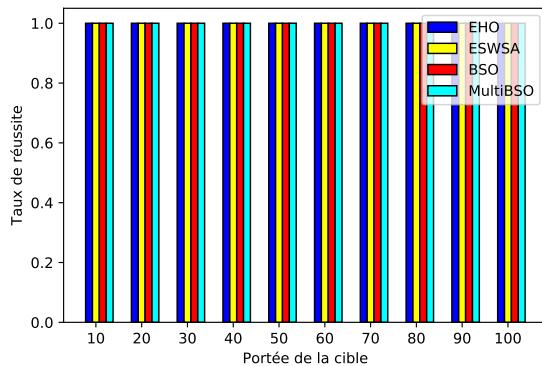


Figure 6.7: Comparaison de la variation du taux de réussite des algorithmes selon la portée des cibles.

Variation du nombre d'itérations et temps d'exécution

a- Environnements simples (sans obstacles)

- Mono-cible

Les figures 6.8 et 6.9 illustrent des *diagrammes à lignes brisées*, représentant respectivement la variation du nombre d'itérations et temps d'exécution de chacune de nos approches confrontées aux différentes portées de la cible dans un environnement sans obstacle.

On remarque que toutes les méthodes voient leur nombre d'itérations décroître avec l'élargissement de la portée de la cible. Pour les méthodes MBSO, EHO et EWSA le nombre d'itérations est estimé entre 12 et 16 pour une portée égale à 10, puis décroît progressivement jusqu'à atteindre environ 2 ou 3 itérations pour la portée maximale. Quant à BSO, il débute avec un peu plus d'itérations (moyenne de 33), avant de diminuer pour rejoindre les autres méthodes lorsque la portée est maximale.

Pour ce qui est des temps d'exécution nos quatre méthodes s'exécutent en des temps records de moins de 0.08 secondes. BSO et MBSO possèdent les meilleurs temps relativement stables et inférieurs à 0.02 secondes, suivies d'EHO atteignant les 0.05 secondes, enfin EWSA est le plus long en arrivant jusqu'à 0.08 secondes pour une portée égale à 100.

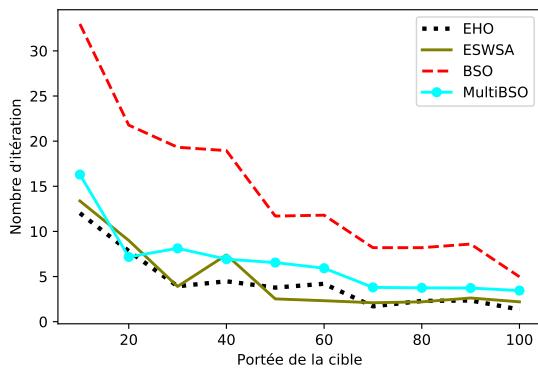


Figure 6.8: Comparaison de la variation du nombre d’itérations des algorithmes selon la portée de la cible (sans obstacles).

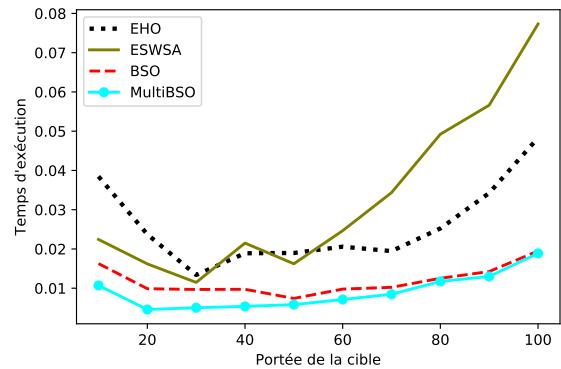


Figure 6.9: Comparaison de la variation du temps d’exécution des algorithmes selon la portée de la cible (sans obstacles).

- Multi-cibles

Les deux figures ci-dessous 6.10 et 6.11 sont représentatives de la variation du nombre d’itérations et temps d’exécution (dans cet ordre) de nos algorithmes, faces aux différentes portées des cibles dans des environnements sans obstacles.

Nous distinguons trois comportements par rapport au nombre d’itérations, le premier est propre à BSO, celui-ci détient le plus grand nombre d’itérations pour toutes les portées, malgré la réduction de ce nombre de 107 à 29 en vue de l’augmentation de la valeur de la portée des cibles. Compte au 2^{ème}, il est caractérisé par un nombre d’itérations avoisinant les 63 itérations pour une portée égale à 10 puis une diminution de ce nombre jusqu’à atteindre entre 4 et 8 itérations pour les cibles de portée 100; ce comportement concerne les deux méta-heuristiques EHO et ESWSA. Enfin le 3^{ème} comportement est celui de MBSO possédant le plus petit nombre d’itérations initial 31 qui diminue graduellement jusqu’à 9 itérations pour la portée égale à 100s.

En termes de temps d’exécution seul ESWSA subit une augmentation visible allant de 0.15 à 0.83 secondes pour les portées de 10 à 100 respectivement, les trois autres algorithmes ne dépassent pas les 0.23 secondes, cette valeur correspondant aux résultats d’EHO pour la portée de 100 positions, BSO et MBSO ayant des temps légèrement inférieurs.

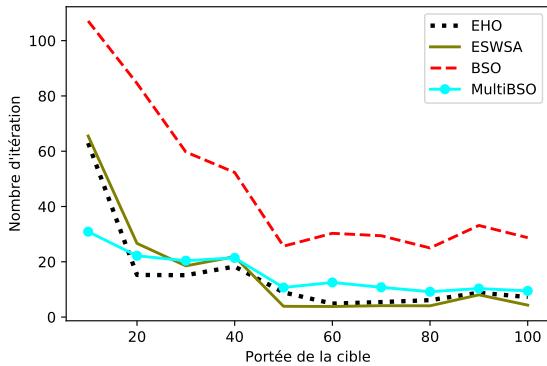


Figure 6.10: Comparaison de la variation du nombre d’itérations des algorithmes selon la portée des cibles (sans obstacles).

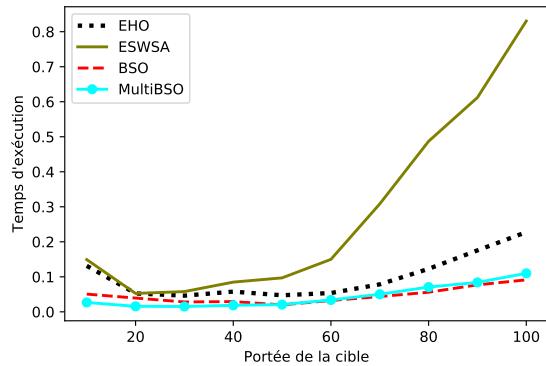


Figure 6.11: Comparaison de la variation du temps d’exécution des algorithmes selon la portée des cibles (sans obstacles).

b- Environnements avec obstacles

- Mono-cible

Les deux *diagrammes à lignes brisées* des figures 6.12 et 6.13 retracent la variation du nombre moyen d’itérations et temps moyens d’exécution de nos algorithmes faisant face à une portée croissante de la cible dans des environnements avec obstacles.

Pour la première valeur de portée (égale à 10) BSO est celui qui nécessite le plus d’itérations (52.68), suivi de EHO et MBSO dont le nombre d’itérations est d’environ 14, et enfin ESWSA qui démarre avec seulement 3.58 itérations. pour toutes les mét-heuristiques ce nombre évolue de manière inversement proportionnelle à la portée de la cible jusqu’à ce que la portée soit égale à 50 où il commence à se stabiliser autour de 9 itérations.

Les temps d’exécution sont minorés par 0.001 secondes et majorés par la valeur 0.11 secondes, ce qui est remarquablement rapide. On distingue une augmentation des temps d’exécution pour ESWSA, plus légère pour EHO suite à l’élargissement de la surface de la portée de la cible. Les deux autres approches restent moyennement stables n’excédant pas les 0.02 secondes.

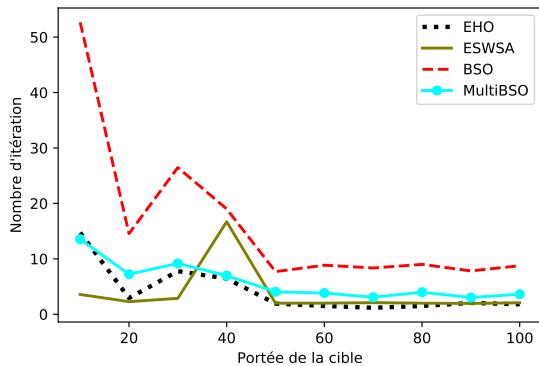


Figure 6.12: Comparaison de la variation du nombre d’itérations des algorithmes selon la portée de la cible (avec obstacles).

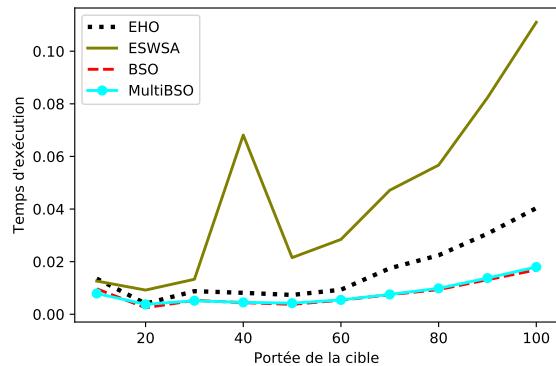


Figure 6.13: Comparaison de la variation du temps d’exécution des algorithmes selon la portée de la cible (avec obstacles).

- Multi-cibles

Les figures 6.14 et 6.15 reflètent la variation du nombre d'itérations et temps d'exécution de nos quatre approches de recherche face aux portées des cibles, cela dans des environnements avec obstacles.

Le nombre d'itérations diminue avec l'augmentation de la portée des cibles pour toutes les approches, néanmoins BSO se distingue par une diminution importante en passant d'une moyenne de 110.6 à 22.58 itérations. Contrairement à MBSO, EHO et EWSWA qui varient entre 30 et 3 itérations.

Pour ce qui est des temps d'exécution, nous remarquons que pour EWSWA la durée de recherche croît considérablement avec l'augmentation de la portée des cibles, comme nous pouvons le voir il passe de 0.06 à 2.73 secondes. Les autres algorithmes qui se comportent de manière similaire mais avec une marge moins importante passant de 0.01 et 0.23 secondes.

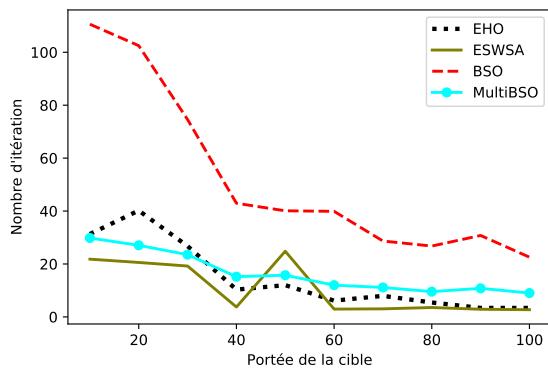


Figure 6.14: Comparaison de la variation du nombre d'itérations des algorithmes selon la portée des cibles (avec obstacles).

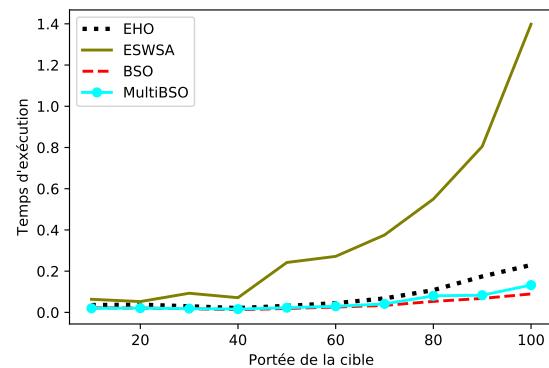


Figure 6.15: Comparaison de la variation du temps d'exécution des algorithmes selon la portée des cibles. (avec obstacles)

c- Environnements complexes

- Mono-cible

Les *diagrammes à lignes brisées* des figures 6.16 et 6.17, décrivent la variation du nombre d'itérations et temps d'exécution de nos approches, celles-ci confrontées aux diverses valeurs de portée de la cible recherchée et ce dans des environnements complexes.

Nous remarquons que le nombre d'itérations décroît avec l'augmentation de la portée de la cible, BSO débute avec le plus grand nombre de 60.83 pour en effectuer que 6.60 itérations lorsque la portée est à 100. EWSWA passe de 40.08 à 2.03 itérations. Quant à EHO, lui fluctue entre 22.13 et 1.48 itérations. Enfin MBSO possède la courbe descendante la plus régulière passant de 25 à 3.05 itérations.

Pour ce qui est des temps d'exécution, ceux de BSO, MBSO ainsi que EHO connaissent une légère augmentation de 0.02 à 0.04 secondes contrairement à ceux de EWSWA qui double passant de 0.06 à 0.12 secondes.

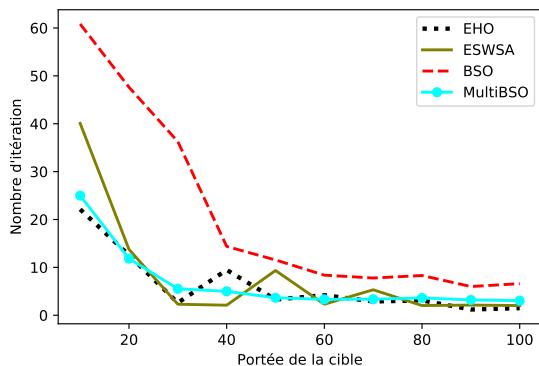


Figure 6.16: Comparaison de la variation du nombre d’itérations des algorithmes selon la portée de la cible (complexe).

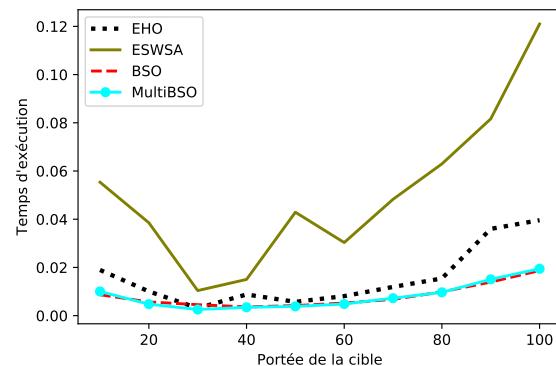


Figure 6.17: Comparaison de la variation du temps d’exécution des algorithmes selon la portée de la cible (complexe).

- Multi-cibles

À travers les figures 6.18 et 6.19, sont représentées respectivement la variation du nombre d’itérations et temps d’exécution de nos approches sous-mises aux dix valeurs de portée des cibles dans des environnements complexes.

Comme dans le mode mono-cible l’augmentation de la portée des cibles conduit à une baisse du nombre d’itérations. On note pour BSO une importante chute du nombre d’itérations allant de 110.6 à 22.58 suivi d’ EHO et MBSO qui se comportent de façon similaire avec un nombre d’itérations initial aux alentours de 80 qui dégringole aux environs de 8 itérations. Enfin ESWSA possède la baisse la moins flagrante passant de 46.88 à 3.08 itérations atteignant le nombre minimum d’itérations qui est de 3.

Une hausse du taux de croissance est observée pour les temps d’exécution de ESWSA (de 0.05 à 1.10 secondes). contrairement à ceux de BSO, MultiBSO et EHO où une stabilisation demeure avec une très légère augmentation pour EHO ne dépassant pas les 0.21 secondes, pour les portées entre 60 et 100.

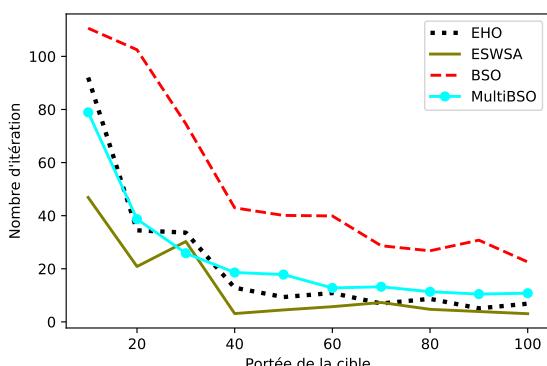


Figure 6.18: Comparaison de la variation du nombre d’itérations des algorithmes selon la portée des cibles (complexe).

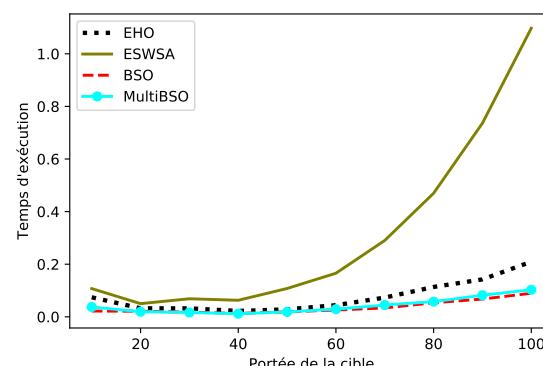


Figure 6.19: Comparaison de la variation du temps d’exécution des algorithmes selon la portée des cibles (complexe).

- Analyse relative à la portée des cibles

À travers les résultats commentés des cas du mono-cible et multi-cibles appliqués à toutes les approches, on peut retenir les points suivants :

- Même pour des petites portées des cibles, nos algorithmes arrivent à bout de la recherche avec succès.
- L'élargissement graduel de la portée des cibles a fait fléchir le nombre d'itérations.
- Malgré les temps d'exécution acceptables, Ils sont moyennement plus importants pour le mode multi-cibles.
- La densité des obstacles influe la recherche ce qui s'explique par l'augmentation des temps d'exécution et nombre d'itérations.

6.4.3 Expérimentations par rapport à la taille de l'environnement

Taux de réussite

a- Environnements simples (sans obstacles)

- Mono-cible

Le graphe de la figure 6.20 montre la variation du taux de réussite pour chaque approche développée, celles-ci sous-mises à divers tailles d'environnements sans obstacles.

Pour EHO et EWSWA un taux maximal de réussite est observé quelle que soit la taille de l'environnement. En revanche BSO et MBSO voient leur taux de succès maximal diminuer jusqu'à 60% pour BSO et 77% pour MBSO pour les deux dernières tailles c'est-à-dire 3000 et 5000.

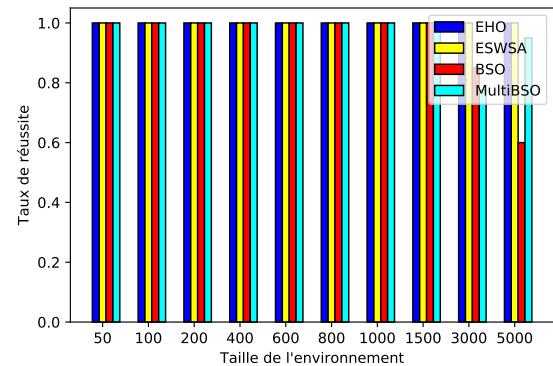


Figure 6.20: Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (sans obstacles).

- Multi-cibles

La figure 6.21 à droite décrit les taux de réussite dans la recherche des cibles, de chaque mét-heuristique en fonction de la variation de la taille des environnements sans obstacles.

Nous nous apercevons que comme pour le mono-cible, EHO et ESWSA atteignent le taux de 100% pour toutes les tailles, ce qui n'est pas le cas de BSO dont le taux de réussite décroît à partir de la 7^{ème} taille d'environnement avec 99% pour atteindre 51% pour les environnements de taille 5000 et MBSO dont les taux de succès des deux dernières tailles d'environnement sont respectivement de 85% et 79%.

b- Environnements avec obstacles

- Mono-cible

La figure 6.22 affiche la variation des taux de réussite de nos approches selon des tailles croissantes des environnements avec obstacles.

Pour les tailles entre 50 et 1500, tous nos algorithmes arrivent à obtenir les 100% de taux de succès, toutefois les environnements de dimensions 3000 connaissent une baisse de ce taux à 76% pour BSO mais toujours 100% pour les autres, enfin pour les plus grands environnements MBSO atteint un taux de 87% contre 55% pour BSO et 100% pour EHO et ESWSA.

- Multi-cibles

La figure 6.23 représente la variation des taux de succès de nos approches confrontées à divers tailles d'environnement avec obstacles.

EHO et ESWSA arrivent à maintenir les 100% de taux de réussite pour toutes les tailles testées, contrairement à MBSO qui relâche son taux de réussite à 83% au bout de l'avant-dernière taille (3000), et BSO qui déjà à partir des tailles 1000 baisse à 67% pour continuer à baisser jusqu'à 17%. Toujours dans la limite du nombre d'itérations maximal (1000).

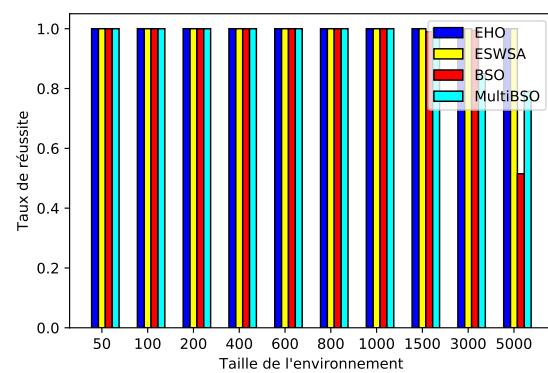


Figure 6.21: Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (sans obstacles).

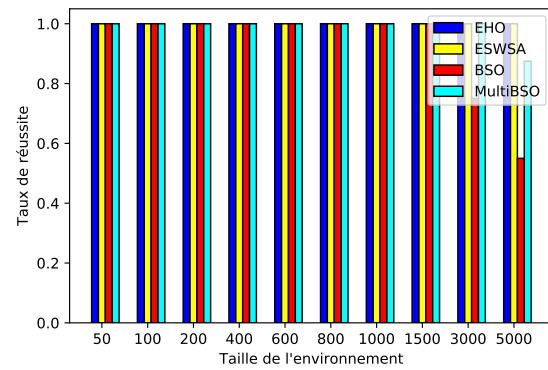


Figure 6.22: Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (avec obstacles).

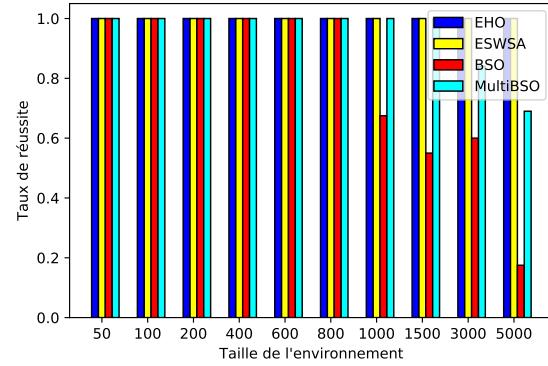


Figure 6.23: Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (avec obstacles).

c- Environnements complexes

- Mono-cible

Dans la figure 6.24 est représentée la variation des taux de réussite de l'ensemble de nos approches exécutées sur différentes tailles d'environnements complexes.

Pour toutes les tailles testées de 50 à 3000 la totalité de nos algorithmes atteignent les 100% de réussite, mais pour la dernière taille d'environnement (5000) seuls EHO et EWSA arrivent à garder le même taux de réussite, tels que MBSO décent à 90% et BSO chute à 60% car ils sont limités par le nombre d'itérations maximal.

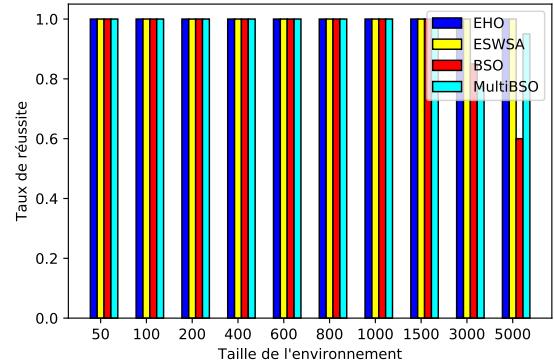


Figure 6.24: Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (complexe).

- Multi-cibles

Les *diagrammes à bandes* de la figure 6.25 décrivent comment les taux de succès de nos algorithmes varient faces aux tailles d'environnements complexes à la recherche de plusieurs cibles.

EHO et EWSA maintiennent les 100% de taux de réussite pour toutes les tailles d'environnement, idem pour MBSO sauf pour ce qui est de la dernière taille il n'arrive qu'à 71%. Par contre BSO décent sous la barre des 100% dès la taille 1000, à partir de là, son taux fluctue entre 36% et 98% car freiné par le nombre d'itérations Max.

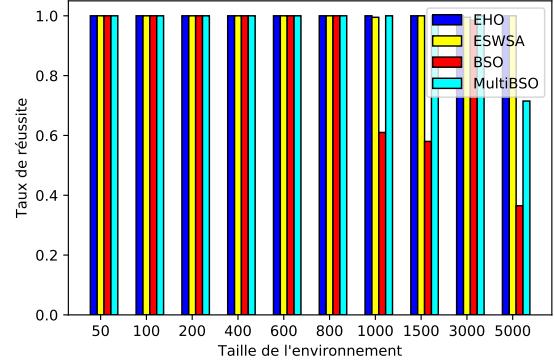


Figure 6.25: Comparaison de la variation du taux de réussite des algorithmes selon la taille de l'environnement (complexe).

Variation du nombre d'itérations et temps d'exécution

a- Environnements simples (sans obstacles)

- Mono-cible

Dans les figures 6.26 et 6.27 sont représentés la variation du nombre d'itérations et du temps d'exécution de nos méta-heuristiques pour l'ensemble des tailles d'environnement sans obstacles sélectionnées.

Les méta-heuristiques inspirées des éléphants détiennent le nombre d'itérations le plus réduit, celui-ci varie entre 1 et 10 itérations pour EHO et entre 2 et 13 itérations pour EWSA. Par contre les deux variantes de BSO connaissent une importante augmentation du nombre d'itérations de 4 à 574 itérations pour BSO et de 2 à 189 pour MBSO, suite à l'augmentation de la taille des environnements.

Pour ce qui est de l'aspect "temps", toutes les approches sont sujettes à une croissance proportionnelle à la croissance des tailles d'environnement à différents coefficients près, tels que nous pouvons les ordonner selon la vitesse d'augmentation des temps

d'exécution de l'algorithme le plus lent au plus rapide comme suit : ESWSA puis BSO suivit de MBSO et enfin EHO, ce dernier atteint un maximum de 4.6 secondes pour une taille = 5000.

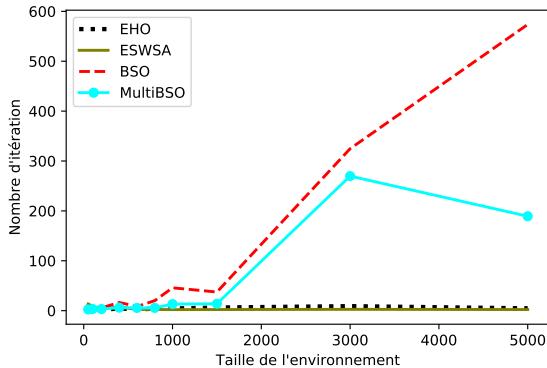


Figure 6.26: Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (sans obstacles).

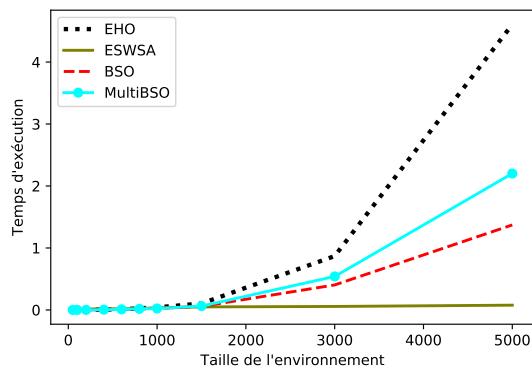


Figure 6.27: Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (sans obstacles).

- Multi-cibles

Les courbes des figures 6.28 et 6.29 illustrent la variation du nombre moyen d'itérations et temps moyens d'exécution de nos algorithmes vis-à-vis des différentes tailles d'environnement sans obstacles.

Ici aussi pour le mode multi-cibles EHO et ESWSA effectuent le moins d'itération, avec une légère augmentation allant de 2.43 jusqu'à 19.43 itérations pour EHO, et de 3.73 à 6.95 itérations pour ESWSA, cela pour l'agrandissement de la taille des environnements. Par ailleurs, BSO et MBSO subissent une conséquente élévation du nombre d'itérations allant de 9.88 à 991.18 itérations pour BSO et de 5.95 à 865.53 pour ce qui est de MBSO.

Quant aux temps d'exécution, globalement nous pouvons dire que les temps d'exécution des algorithmes croient avec l'augmentation de la taille des environnements à différentes vitesses, tels que BSO passe progressivement de 0.001 à 6.57 secondes suivie de MBSO avec des temps haussant de 0.001 à 11.75 secondes, puis EHO qui démarre à 0.001 pour atteindre les 19.43 secondes et enfin ESWSA qui enregistre les plus grands temps passant de 0.001 à 17.01 secondes.

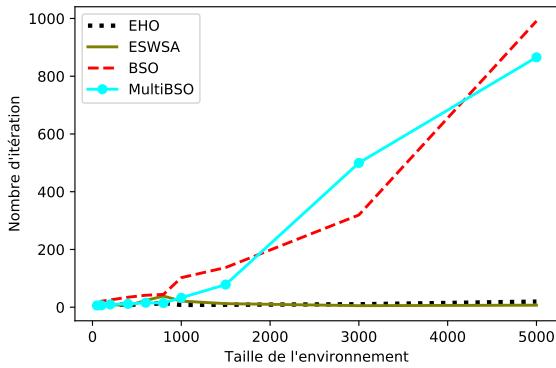


Figure 6.28: Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (sans obstacles).

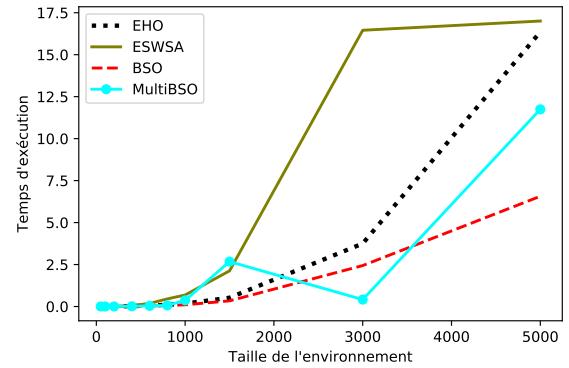


Figure 6.29: Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (sans obstacles).

b- Environnements avec obstacles

- Mono-cible

Les *diagrammes à lignes brisées* des figures 6.30 et 6.31 font apparaître la variation du nombre d'itérations et temps d'exécution de BSO, MBSO, EHO ainsi que EWSWA par rapport aux tailles des environnements avec obstacles.

La figure de gauche montre que BSO détient le plus grand nombre d'itérations pour l'ensemble des tailles d'environnement, avec une augmentation allant de 2.55 à 566.18 itérations. MBSO vient juste après avec des nombres d'itérations croissants tout en restant réduit jusqu'aux environnements de 3000×3000 où il arrive à 48.45 itérations qui par la suite grimpe vers 295.58 itérations pour les plus grands environnements. Enfin EHO et EWSWA disposent des nombres d'itérations les plus réduits, fluctuant entre 1.08 et 29.78 itérations.

Quant à la figure de droite, nous y discernons deux comportements tous deux croissants avec l'élargissement des surfaces des environnements, le 1^{er} englobe BSO, MBSO et EHO qui possèdent des temps d'exécution minimes (entre 0.001 et 3.83 secondes), et le 2^{ème} comportement concerne EWSWA avec des temps moyens plus ou moins élevés, allant de 0.001 à 15.56 secondes.

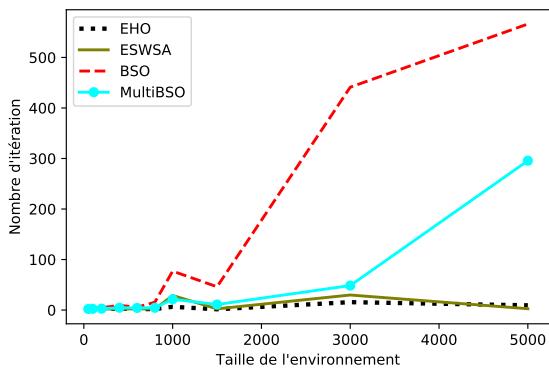


Figure 6.30: Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (avec obstacles).

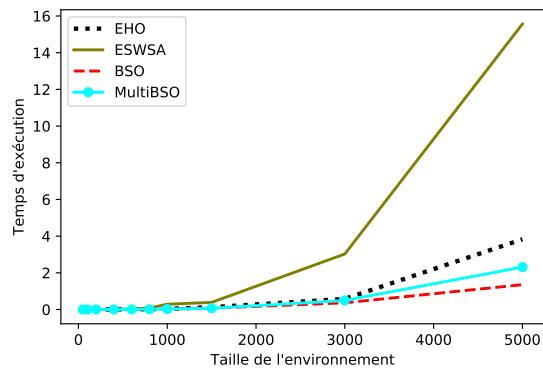


Figure 6.31: Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (avec obstacles).

- Multi-cibles

Les figures 6.32 et 6.33 présentées ci-dessous, retracent la variation du nombre d'itérations et temps d'exécution de nos algorithmes testés sur plusieurs tailles d'environnement avec obstacles à la recherche de plusieurs cibles.

BSO et MBSO se distinguent avec des nombres élevés d'itérations, BSO débutant de 9.50 itérations et arrivant à la limite maximale de 1000 itérations, et MBSO allons d'une moyenne de 5.40 à 920.68 itérations. D'autre part EHO et ESWSA possèdent le moins d'itérations (entre 2.45 et 22.18 itérations).

Côté temps d'exécution, nous pouvons les ordonner du plus rapide au plus long comme suit: BSO avec de 0.001 à 1.99 secondes suivie de MBSO avec de 0.01 à 5.67 secondes puis EHO avec des temps entre 2.45 et 20.50 secondes, et pour finir ESWSA dont les temps croissent de manière spectaculaire de 0.001 à 94.55 secondes.

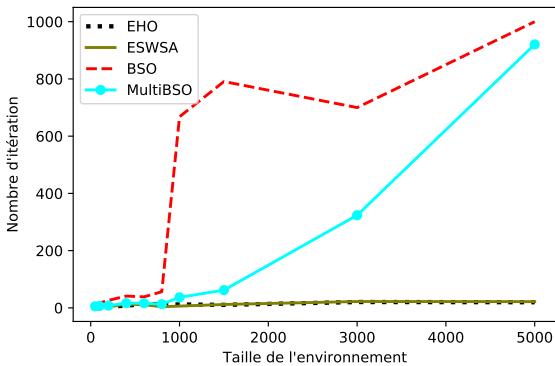


Figure 6.32: Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (avec obstacles).

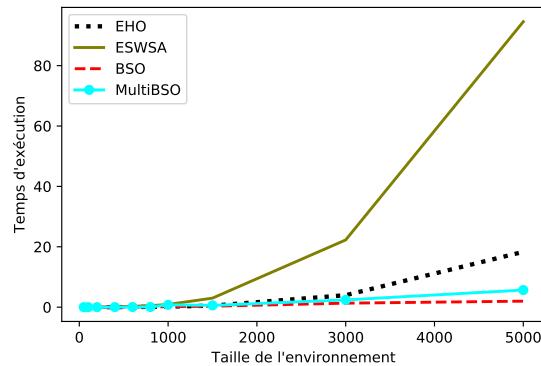


Figure 6.33: Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (avec obstacles).

c- Environnements complexes

- Mono-cible

Dans les figures 6.34 et 6.35 sont décrit la variation du nombre d'itérations et du temps d'exécution de nos algorithmes selon la taille des environnements complexes.

Nous pouvons regrouper les méta-heuristiques en deux groupes selon l'évolution de leur nombre d'itérations, le 1^{er} groupe est constitué de BSO et MBSO qui subit une importante croissance de ce nombre passant de 2.7 à 747.78 itérations pour BSO et de 2.05 à 444.45 itérations pour MBSO.

Le 2^{ème} groupe est celui de EHO et EWSA, ceux-ci croient de manière bien plus raisonnable, soient de 1.35 à 13.83 itérations pour EHO et de 1.05 à 70.45 itérations pour EWSA.

Toutes nos approches connaissent un accroissement des temps d'exécution avec l'élargissement des tailles d'environnement. EWSA est celui dont les temps ont le plus augmenté (de 0.001 à 12.32 secondes), suivi d'EHO avec des temps qui passent de 0.001 à 3.03 secondes, puis vient MBSO avec entre 0.001 et 1.92 secondes enfin le plus rapide est BSO, dont les temps sont entre 0.001 et 1 seconde.

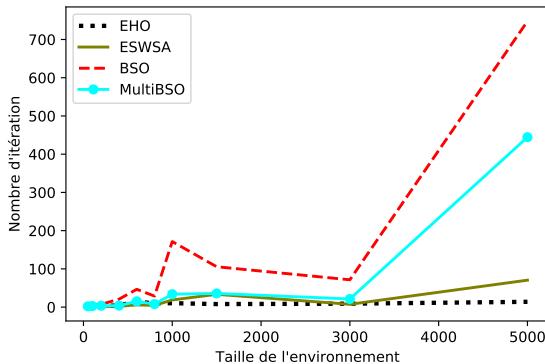


Figure 6.34: Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (complexe).

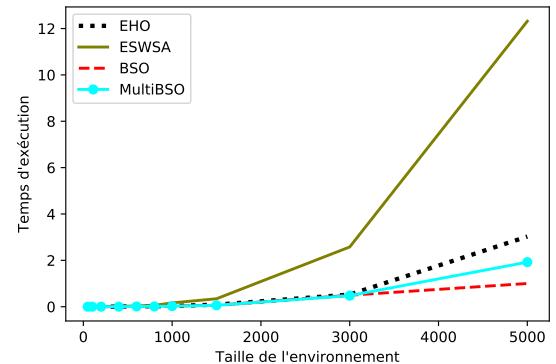


Figure 6.35: Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (complexe).

- Multi-cibles

Les courbes des figures 6.36 et 6.37 illustrent la variation du nombre moyen d'itération et temps moyen d'exécution de nos algorithmes vis-à-vis des différentes tailles d'environnement complexes.

Du point de vue du nombre d'itérations, ils commencent tous avec environ 3 à 12 itérations pour le plus petit environnement puis augmentent. Ici aussi c'est BSO et MBSO qui enregistrent des plus grands nombres avec un maximum de 1000 et 865.5 itérations respectivement, ensuite EHO va jusqu'à 29.3 itérations et enfin EWSA ne dépasse pas la moyenne de 13.65 itérations.

Quant aux temps d'exécution les 4 approches évoluent de la même manière que pour le mode mono-cible, mais avec des temps plus importants, allant de 0.001 à 93.31 secondes pour EWSA, grimpant de 0.001 à 15.55 secondes pour EHO, puis de 0.001 à 10.23 secondes pour MBSO, enfin pour BSO les temps vont de 0.001 à 4.79 secondes.

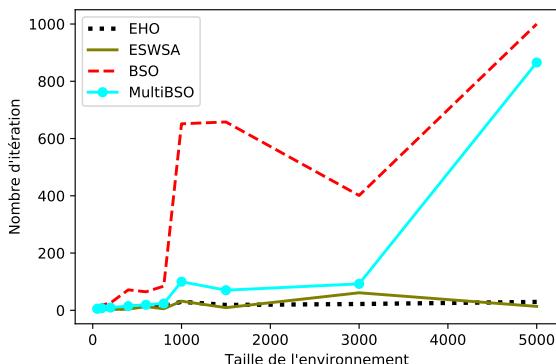


Figure 6.36: Comparaison de la variation du nombre d'itérations des algorithmes selon la taille de l'environnement (complexe).

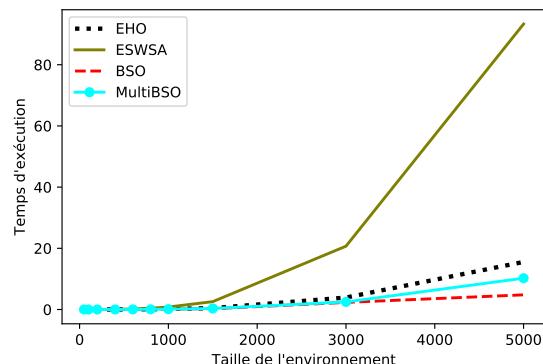


Figure 6.37: Comparaison de la variation du temps d'exécution des algorithmes selon la taille de l'environnement (complexe).

- Analyse relative à la taille des environnements

Suite aux résultats concernant la taille des environnements simples, avec obstacles et complexes selon les deux modes mono et multi-cibles, nous pouvons noter que:

- Contrairement aux deux approches inspirées des abeilles (BSO et MBSO), les deux autres méta-heuristiques à savoir : EHO et ESWSA atteignent le taux maximum de réussite dans leur recherche de la ou les cible(s) quelle que soit la taille de l'environnement à explorer.
- Pour tous les algorithmes le nombre d'itérations croît avec l'augmentation de la taille des environnements, mais à des vitesses différentes. Telles que EHO et ESWSA effectuent le moins d'itérations.
- Les temps restent relativement acceptables pour le mono-cible, par contre pour le mode multi-cible ils atteignent les 1min 30. C'est ESWSA le plus long.
- EHO et ESWSA minimisent le nombre d'itérations en raison des grands pas entre deux positions successives qu'ils génèrent, mais cela leur coûte cher en temps, ce qui est l'inverse de BSO et MBSO qui effectuent de petits pas dans des temps moindres.
- Pour les deux modes et pour toutes les approches, les temps d'exécution et nombre d'itérations croissent avec l'augmentation de la complexité des environnements.

6.4.4 Expérimentations par rapport au nombre de cibles

Taux de réussite

Les taux de réussite sont présentés sous forme de *diagramme à bandes* dans la figure 6.38, elle comporte les résultats de chaque méta-heuristique pour plusieurs nombres de cibles.

Nous nous apercevons qu'à l'unanimité toutes nos approches trouvent la totalité (100%) du nombre de cibles, quel que soit ce dernier entre 1 et 15 cibles (avec un pas de 2). Tout en respectant la limite du nombre d'itérations maximal (1000). Ces même résultats sont valables pour les trois types d'environnement étudiés.

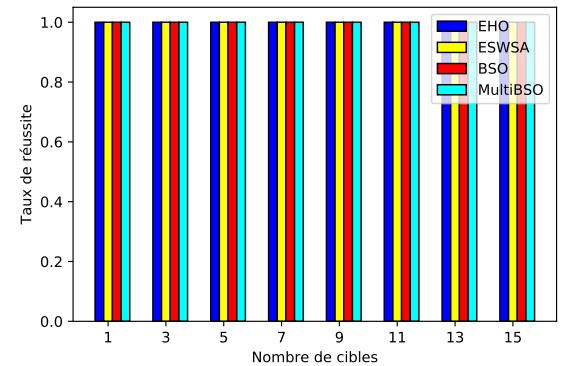


Figure 6.38: Comparaison de la variation du taux de réussite des algorithmes selon le nombre de cibles.

Variation du nombre d'itérations et temps d'exécution

a- Environnements simples (sans obstacles)

Les deux figures 6.39 et 6.40 ci-dessous retracent la variation du nombre moyen d'itération et temps moyen d'exécution de nos algorithmes confrontés à un nombre croissant de cibles dans des environnements sans obstacles.

Tous les algorithmes résolvent le problème de recherche de cibles en un nombre d'itérations variant entre 1 à 31 itérations à l'exception de BSO, qui vient bien après ses concurrents démarrant de 8 et arrivant jusqu'à 83 itérations. Notons que la meilleure performance est celle de ESWSA.

Nous avons obtenu des temps d'exécution remarquablement réduits pour l'ensemble des méthodes, certes les temps ont accru avec l'augmentation du nombre de cibles mais cela reste très raisonnable entre 0.02 et 0.05 secondes pour ESWSA, entre 0.001 et 0.06 secondes pour MBSO, de 0.01 à 0.07 secondes pour BSO et enfin de 0.01 à 0.12 secondes pour ce qui est d'EHO.

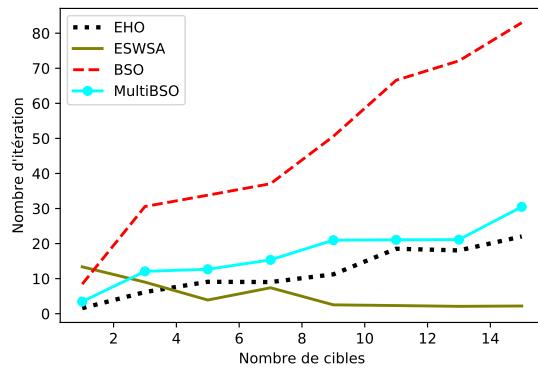


Figure 6.39: Comparaison de la variation du nombre d'itérations des algorithmes selon le nombre de cible (sans obstacles).

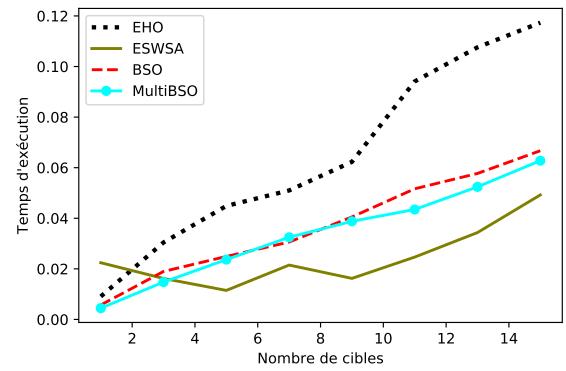


Figure 6.40: Comparaison de la variation du temps d'exécution des algorithmes selon le nombre de cible (sans obstacles).

b- Environnements avec obstacles

Les deux figures 6.41 et 6.42, décrivent la variation du nombre d'itérations et temps d'exécution de nos quatre approches à la recherche d'un nombre variable de cibles dans des environnements avec obstacles.

Le nombre d'itérations croît avec l'augmentation du nombre de cibles dans l'environnement, BSO connaît la plus grande croissance en passant de 23.53 à 104.43 itérations. MBSO fait moins d'itérations avec entre 6.10 et 26.08. EHO et EWSWA effectuent de 1.55 à 17.98 et de 2.58 à 7.58 itérations respectivement.

Les temps d'exécution sont très courts pour l'ensemble des méthodes, notons l'augmentation des temps de 0.02 jusqu'à 0.48 secondes pour EWSWA. Ce dernier étant considéré comme le plus long, puis de 0.01 à 0.07 secondes pour EHO et de 0.01 à 0.05 pour les algorithmes inspirés des abeilles (BSO et MBSO).

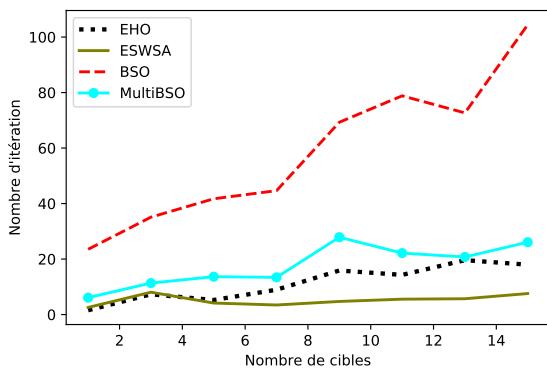


Figure 6.41: Comparaison de la variation du nombre d'itérations des algorithmes selon le nombre de cible (avec obstacles).

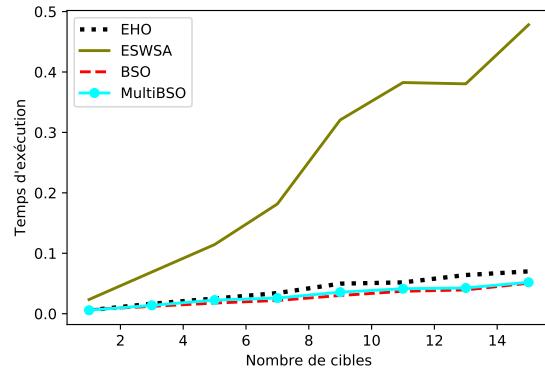


Figure 6.42: Comparaison de la variation du temps d'exécution des algorithmes selon le nombre de cible (avec obstacles).

c- Environnements complexes

Les *diagrammes à lignes brisées* des figures 6.43 et 6.44 mettent en relief l'influence du nombre de cibles recherchées sur le nombre d'itérations et temps d'exécution de nos algorithmes, dans des environnements complexes.

En observant le nombre d'itérations nous constatons que nos approches peuvent être classées selon leur rythme de croissance face au nombre de cibles recherchées. Telles que, BSO possède les plus grands nombres avec de 25.53 à 125.83 itérations, puis vient MBSO qui passe de 8.73 à 35.28 itérations, suivie d'EOH dont les nombres sont entre 1.8 et 21.3 itérations, non loin EWSWA avec entre 2.45 et 20.78 itérations.

Les algorithmes inspirés des abeilles possèdent des temps d'exécution bas, légèrement croissant avec l'augmentation du nombre de cibles passant de 0.01 à 0.05 secondes. EHO s'en rapproche avec une croissance de ces temps allant de 0.001 à 0.08 secondes. EWSWA quant à lui sort du lot avec une augmentation de 0.02 à 0.48 secondes.

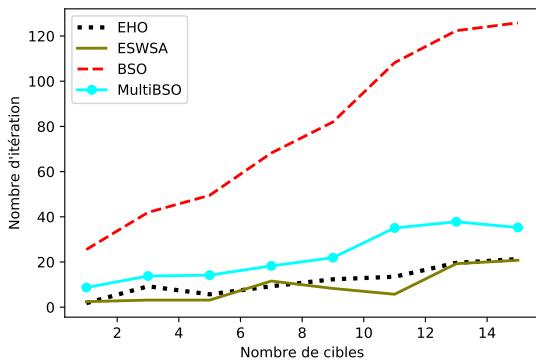


Figure 6.43: Comparaison de la variation du nombre d’itérations des algorithmes selon le nombre de cible (complexe).

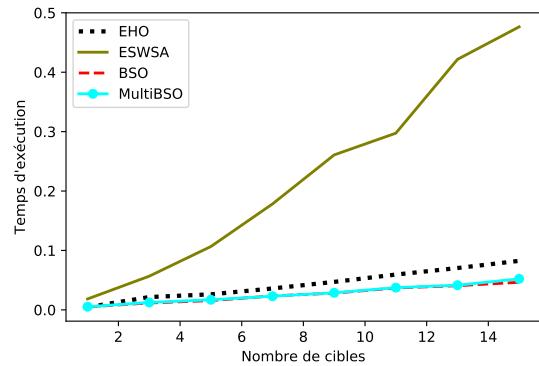


Figure 6.44: Comparaison de la variation du temps d’exécution des algorithmes selon le nombre de cible (complexe).

Analyse relative au nombre de cibles

D’après les résultats présentés ci-dessus par rapport au nombre de cibles recherchées dans les différents types d’environnements expérimentés, nous sommes en mesure de tirer les quelques conclusions qui suivent:

- Nos quatre méta-heuristiques ont fait leurs preuves de par leur capacité à trouver toutes les cibles qu’importe leur nombre (selon les conditions citées).
- Plus il y a de cible à chercher, plus les algorithmes font d’itération et plus le temps d’exécution croît.
- BSO détient le plus grand nombre d’itérations quel que soit l’environnement mais en contrepartie il possède les meilleurs temps d’exécution.
- La méta-heuristique EWSWA effectue des temps acceptables dans des environnements simples à grand nombre de cibles avec un nombre d’itérations record. En revanche lors de la présence d’obstacles les temps d’exécution deviennent beaucoup trop importants comparés aux autres approches (BSO, EHO, MBSO) détenant ainsi les pires temps.
- Quant aux deux autres algorithmes leur évolution en termes de temps et nombre d’itérations est bornée par celles de BSO et EWSWA.
- Le nombre d’itérations reste relativement acceptable pour 15 cibles, à noter qu’EWSWA, EHO et MBSO étaient plus performants sur ce plan.
- Les temps d’exécution étaient globalement satisfaisants pour tous les algorithmes, car n’excédant pas les 0.5 secondes.
- La densité des environnements en obstacles, joue un rôle important dans le comportement de nos approches, telle que, plus nos environnements sont complexes plus nos approches produisent d’efforts et prennent de temps à atteindre les cibles.

6.4.5 Comparaison des types d'environnement

La moyenne des temps d'exécution et nombre d'itérations croissent lors du passage d'environnements simples (sans obstacles) aux environnements avec obstacles, mais elles connaissent une considérable augmentation lorsqu'on a affaire à des environnements complexes.

Cela est dû à la complexité des environnements entravant et rendant plus difficile le mouvement des robots, ainsi le choix de la bonne trajectoire devient de plus en plus complexe ce qui impacte le temps d'exécution.

Pour ce qui est des taux de réussite, ils dépendent beaucoup plus des méthodes de recherche.

6.4.6 Comparaison de nos quatre approches

Nos quatre approches développées et testées ne possèdent pas les mêmes comportements face aux différentes variantes liées à l'environnement de recherche. Nous pouvons conclure que:

- L'approche MBSO est la plus stable, les variations du nombre d'itérations et de temps d'exécution sont harmonieux.
- EWSWA est le meilleur en termes de nombre d'itérations suivi de près par EHO.
- Les méta-heuristiques inspirées des abeilles sont les meilleures en termes de temps d'exécution.
- L'algorithme EWSWA atteint parfois des temps d'exécution peu raisonnables.
- Globalement le multi-swarming (Multi-BSO) a grandement amélioré BSO que ce soit dans les taux de réussite, le nombre d'itérations ou temps d'exécution.
- BSO et MBSO ont quelques lacunes par rapport aux très grands environnements.
- EHO possède le meilleur compromis entre taux de réussite (toujours à 100%), nombre d'itérations assez bas et temps d'exécution très raisonnables.

6.5 Simulateur

La réalisation de notre simulateur temps réels et interactif pour la visualisation de l'évolution des approches implémentées que ça soit BSO, MBSO, EHO ou encore EWSWA, a été mise en œuvre afin de faciliter la compréhension de notre travail.

6.5.1 Fonctionnement

L'affichage de l'environnement de recherche de notre simulateur passe par les étapes suivantes:

- Sélection de l'environnement à explorer sous sa forme matricielle comme décrite dans la modélisation de la section 3.2.
- Translation de l'espace de recherche sélectionné en une image PNG, en assignant aux obstacles et à la portée deux couleurs distinctes.
- Représentation de chaque position de l'environnement de recherche par un unique pixel dans l'image PNG.

Quant à l'étape de recherche des cibles, elle nécessite de retracer la trajectoire de chaque robot, en dessinant leurs chemins entre la position actuelle et la nouvelle position calculée par la méta-heuristique de recherche, le dessin suit l'algorithme de BRESENHAM [53].

Dans notre simulateur temps réel, pour une question de fluidité et de parallélisme il est nécessaire de faire appel au *multi-threading*, un *thread* pour l'exécution de la méta-heuristique et un autre pour l'interface de simulation, la coordination entre ces deux *threads* est illustrée par le schéma de la figure 6.45 qui suit.

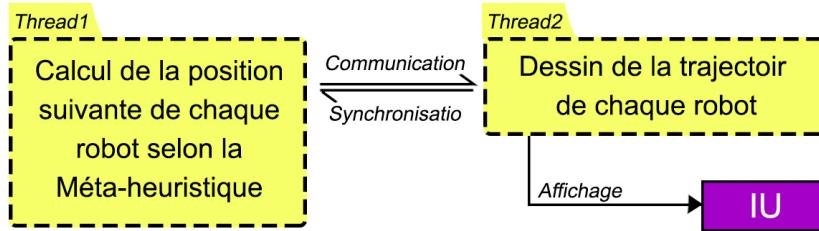


Figure 6.45: Schéma de communication du système multi-threads.

6.5.2 Interface

L'interface offre deux sections possibles, une dédiée au choix de l'environnement nous permettant de choisir un environnement selon nos préférences et paramètres, celle-ci est représentée dans la figure 6.46 ci-dessous:

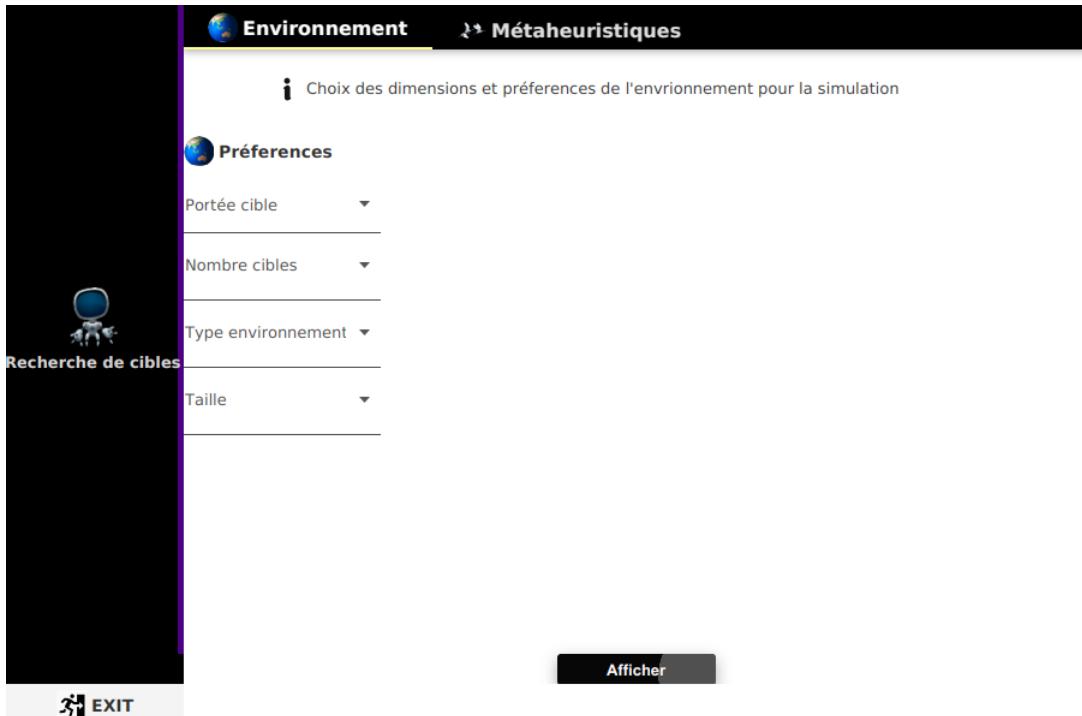


Figure 6.46: Présentation de l'interface

Et une autre section dédiée aux méta-heuristiques, qui englobe toutes les approches basées essaims traitées dans ce projet, afin d'être appliquées à l'environnement choisi dans la section Environnement. Cette section est visible dans la figure 6.47 qui suit:

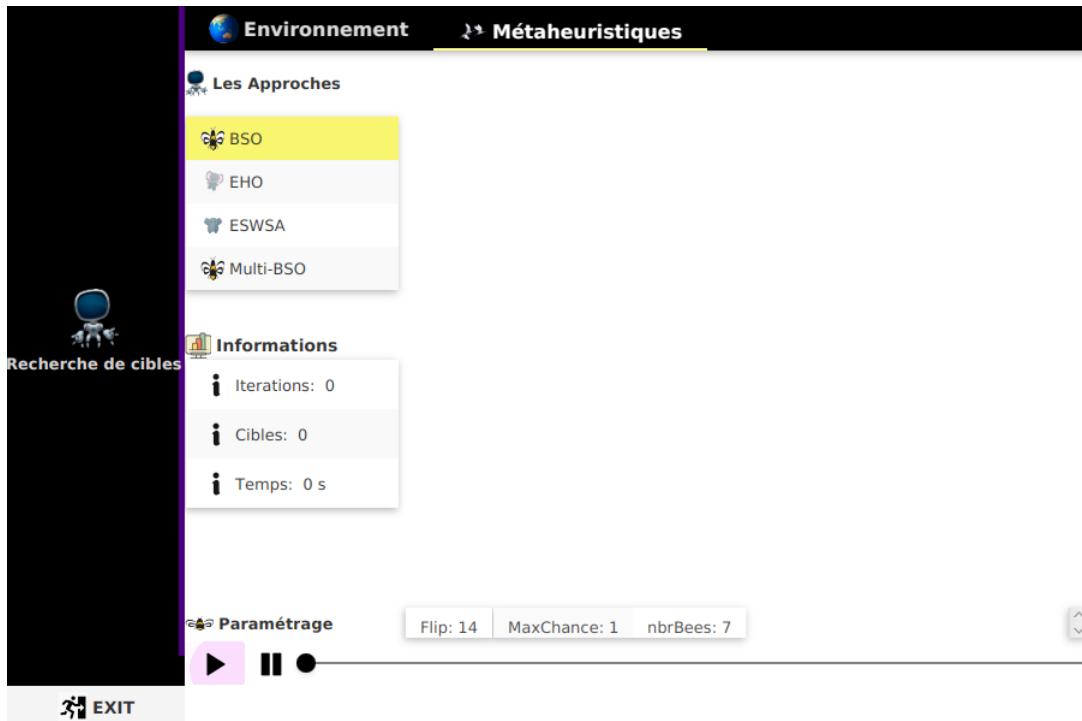


Figure 6.47: Présentation de la section Métaheuristiques

Environnement

La section Environnement permet à l'utilisateur de choisir un environnement selon les paramètres et préférences suivantes :

Nombre de cibles Nos environnements peuvent comporter une ou plusieurs cibles dont les positions sont générées de manière aléatoire.

Dans le cadre de notre simulation, nous nous sommes contentés de permettre deux choix pour le nombre de cibles qui sont :

- Mono-cible : une seule cible.
- Multi-cibles : avec 5 cibles.

Ce choix peut être augmenté selon le nombre de cibles qu'on souhaite avoir.

Portée des cibles Les portées de cibles disponibles dans l'application sont donc catégorisées comme suit:

- **Portée petite**, d'un rayon d'environ 5% de la taille de l'environnement.
- **Portée moyenne**, d'un rayon d'environ 15% de la taille de l'environnement.
- **Portée large**, d'un rayon d'environ 25% de la taille de l'environnement.

Types d'environnements Les types d'environnements simples, avec obstacles et complexes sont représentés dans la figure ??, sont mis à disposition. Dans le cas des deux derniers types d'environnements, il est à noter que les positions et distributions des obstacles sont générées aléatoirement.

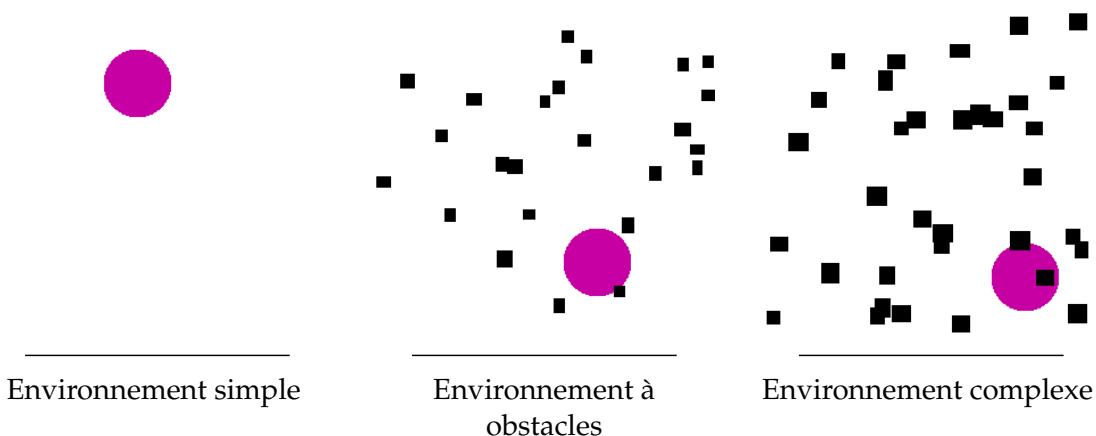


Figure 6.48: Les types d'environnements possibles

Taille d'environnement Les environnements varient selon leur taille, on a alors choisi à titre représentatif quelques tailles par classe, telles que:

- 100 et 400 qui sont assez petits.
 - 600 et 1000 dits de taille moyenne.
 - 1500 qui est assez grand.

Afficher La section environnement présentée dans la figure 6.49 comporte le bouton *afficher* qui sert à visualiser l'environnement choisi en fonction des préférences insérées dans les listes de choix multiples de chaque influent (portée, nombre cibles,...).

Ci-dessous un exemple d'environnement sélectionné:

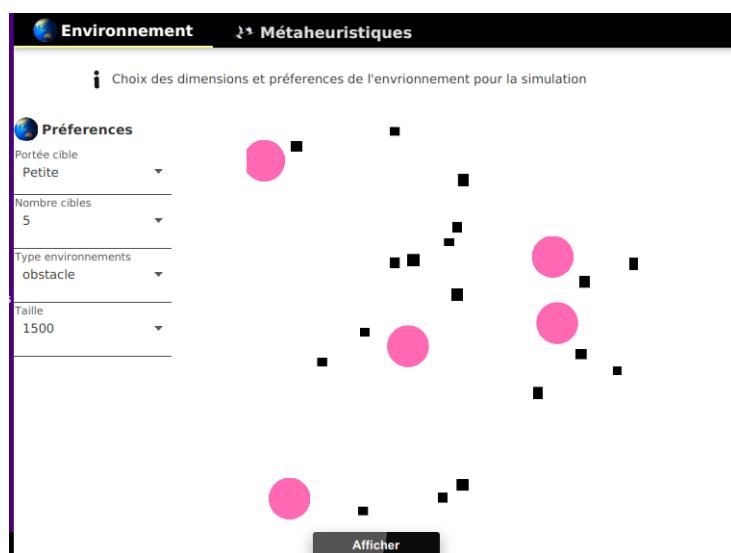


Figure 6.49: Affichage de l'environnement selon les préférences insérées

Métaheuristiques

Les approches Les approches basées essaims auxquels nous avons eu affaire tout au long de ce mémoire *BSO*, *EHO*, *ESWSA* et *Multi-BSO* y sont proposées sous forme d'une

liste permettant de choisir celle à exécuter. Une fois qu'on clique sur une approche deux possibilités pour le choix du paramétrage s'offrent à nous :

- Paramétrage avec algorithme génétique L'option de paramétrage avec l'algorithme génétique (Mini-GA incrémental) est accessible pour chaque approche, comme le montre la figure 6.50 ci-dessous.

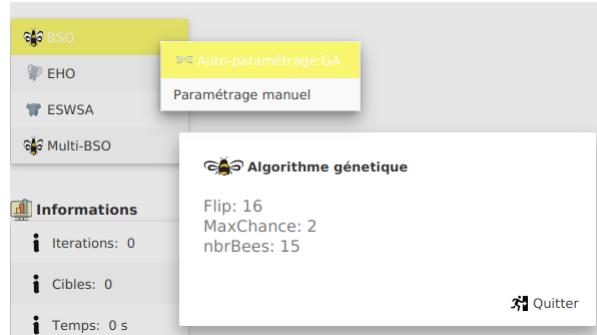


Figure 6.50: Auto-paramétrage avec l'algorithme génétique

- Paramétrage manuel Le paramétrage manuel se fait en choisissant pour chaque paramètre de l'approche basée essaim une valeur. Les figures 6.51 et 6.52 ci-dessous illustrent ces paramètres pour BSO.



Figure 6.51: Menu de paramétrage

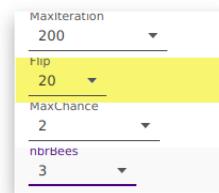


Figure 6.52: Menu de paramétrage manuel pour BSO

Les informations Ce sont les détails identiques pour chaque approche et qui sont: *Nombre d'itérations*, *Nombre de cibles trouvées*, *Temps d'exécution (sec)* comme présentés dans la figure 6.53.

Ces informations sont mises à jour en temps réel, c'est-à-dire elles sont modifiées à chaque itération (le nombre d'itérations inclus).

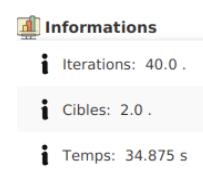


Figure 6.53: Informations durant l'exécution de BSO

Paramétrage C'est une liste horizontale comme illustré dans la figure 6.54, elle contient les paramètres de l'exécution de l'approche choisie, qu'elle soit générée par algorithme génétique ou bien manuellement.

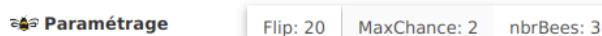


Figure 6.54: Liste des paramètres pour BSO

Start Le bouton "Start" ► , permet de débuter l'exécution après avoir choisi l'approche. Ainsi l'image de l'environnement et des robots (abeilles ou éléphants) sur l'environnement s'affichent.

Stop Le bouton "Stop" || ; permet d'arrêter et mettre fin à l'exécution de l'approche, afin de choisir une autre méta-heuristique à exécuter ou bien relancer la même avec d'autres paramètres.

Le slider de la figure 6.55 joue le rôle d'une barre de progression, tel qu'il s'incrémentera après chaque itération de l'approche en cours d'exécution. Une fois l'exécution terminée, il permet de revenir en arrière retracant ainsi la trajectoire des robots dans l'environnement. Comme pour un film ou une vidéo nous avons la possibilité d'avancer du début jusqu'à la fin, pour revoir l'exécution.



Figure 6.55: Le slider

L'illustration 6.56 suivante est un exemple d'une exécution de BSO dans un environnement multi-cibles de petites portées et avec obstacles.

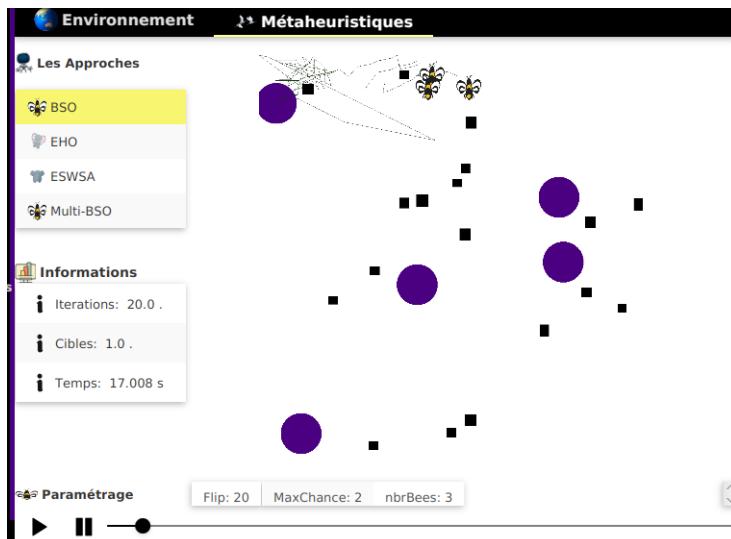


Figure 6.56: Exemple d'exécution de BSO.

Quitter Un bouton "Quitter" EXIT se trouve en bas à gauche de l'interface, il permet de sortir de l'application dès que nous le souhaitons.

6.6 Conclusion

À travers ce chapitre on a pu discuter et voir une panoplie de tests expérimentaux qui concernent toutes les approches de résolution d'intelligence en essaim implémentées qu'on cite :

les deux algorithmes basés essaim d'abeilles BSO, Multi-BSO.

Ou encore EHO, EWSA qui sont quant à eux basés groupe d'éléphants.

D'autres tests ont été présentés portant sur l'algorithme génétique qui ont comme objectif d'effectuer un réglage de paramètres des algorithmes implémentées.

Cette série de tests (23 040 exécutions) s'est portée sur plusieurs caractéristiques de l'environnement comme *La portée de la cible, la taille de l'environnement, nombre de cibles ainsi que la présence d'un nombre éventuel d'obstacles variant*.

L'analyse des différents tests nous a permis d'observer les changements de comportement des algorithmes ce qui nous ont aidé à extraire les avantages de chaque approche, ainsi que ses inconvénients en essayant de décortiquer les raisons pour lesquels ils ont eu lieu.

Ainsi on a pensé à un algorithme hybride qui peut éventuellement surpasser les algorithmes vus dans ce mémoire, cela en combinant les points forts des approches et remédiant aux points faibles.

Conclusion générale et perspectives

La détection de cibles reste un problème attractif, sujet à de multiples propositions de résolution qui ont souvent recours aux systèmes multi-robots, ces derniers se trouvent dans des environnements complexes et inconnus, ils doivent alors se munir d'une stratégie d'évitement d'obstacles pour faire face au monde inconnu dans lequel ils se lancent à la recherche des cibles, agissant ainsi comme des fouilleurs.

Quel que soit le domaine d'application, le besoin de méthodes de recherche efficaces, optimisées et rapides devient vite indispensable, d'où notre intervention à l'apport d'une solution à ce problème.

Pour ce faire il nous a fallu suivre une méthodologie de travail, telle qu'en premier lieu nous avons cherché à nous familiariser avec le domaine et problème auquel on s'attaque et ses nombreuses variantes dues à certains paramètres propres à la recherche de cibles. Cette étape nous a permis de définir dans quelle branche notre présent projet prend place. Par la suite nous avons présenté une synthèse de quelques travaux relatifs à notre problématique, d'une part pour prendre connaissance des lacunes recentrées et limitations de ces derniers pour potentiellement y remédier, et d'autre part, afin de mieux cerner le problème et nous informer des méthodes les plus adaptées à sa résolution.

En second lieu nous sommes penchées sur les outils de résolution basés intelligence en essaim choisis à savoir les méta-heuristiques s'inspirant des abeilles (BSO et Multi-BSO) et des éléphants (EHO et EWSA) ou encore de la génétique (GA). Ce choix fut motivé par les caractéristiques qu'elles profèrent et qui s'adaptent au mieux à la formulation du problème auquel nous sommes confrontées.

Par ailleurs, nous avons clairement défini notre modélisation par rapport à l'environnement, les cibles et les robots. Ensuite on a choisi une stratégie d'évitement d'obstacles conforme à cette modélisation, pour enfin décrire formellement le fonctionnement des quatre approches de recherche qui constituent nos deux contributions majeures, ainsi que l'algorithme génétique mis à profit pour le réglage automatique des paramètres. À l'issue de cette phase nous sommes passé à l'implémentation de nos algorithmes sur machine.

Aussi par souci de visualisation du suivi de la recherche étape par étape, nous avons mis en place un simulateur temps réel de la recherche, exploitant pour cela le multi-threading pour une application fluide et agréable.

Enfin, l'évaluation de nos approches a été appuyé par un bon nombre d'expérimentations, nous permettant d'effectuer une comparaison fiable de leur efficacité. Suite à cela nous pouvons dire que les résultats obtenus étaient satisfaisants. D'ailleurs l'analyse des résultats nous a permis de mettre le doigt sur les avantages ainsi que les points faibles ou inconvénients de chaque approche, d'où notre ambition d'exploiter cela à travers une hybridation des quatre méthodes basées essaim ne retenant que les atouts de chacune, pour des résultats d'une qualité encore plus satisfaisante.

Pour résumer, dans le présent travail nous avons développé:

- Un générateur d'environnement de recherche.
- L'approche BSO pour la recherche de cibles.
- L'approche MBSO pour la recherche de cibles.

- L'approche EHO pour la recherche de cibles.
- L'approche EWSA pour la recherche de cibles.
- L'algorithme GA de paramétrage.
- La stratégie d'évitement d'obstacles et de simulation du déplacement des robots.
- L'auto-génération des expérimentations en graphiques.
- Un simulateur de recherche temps réel.

Néanmoins, nous pensons déjà à quelques améliorations que nous avons en perspective, dont nous citerons:

- L'adaptation de la recherche aux cibles mobiles.
- L'adaptation de la stratégie d'évitement d'obstacles aux obstacles mobiles (comme des être humain, animaux, voitures, ...)
- Réalisation de la version multi-agents de nos approches afin de bénéficier d'un parallélisme et d'une coopération plus réaliste.
- Hybridation des avantages de chaque approche.
- Partitionnement des environnements en affectant des équipes à chaque partition.

Bibliographie

- [1] Madhubhashi Senanayake, Ilankaikone Senthooran, Jan Carlo Barca, Hoam Chung, Joarder Kamruzzaman, and Manzur Murshed. Search and tracking algorithms for swarms of robots: A survey. *Robotics and Autonomous Systems*, 75:422–434, jan 2016.
- [2] Cyril Robin and Simon Lacroix. Multi-robot Target Detection and Tracking: Taxonomy and Survey. *Autonomous Robots*, 40(4):pp.729–760, April 2016.
- [3] Yun-Qian Miao, Alaa Khamis, and Mohamed S. Kamel. Applying anti-flocking model in mobile surveillance systems. In *2010 International Conference on Autonomous and Intelligent Systems, AIS 2010*. IEEE, jun 2010.
- [4] Habiba Drias, Souhila Sadeg, and Safa Yahi. Cooperative bees swarm for solving the maximum weighted satisfiability problem. In *Computational Intelligence and Bio-inspired Systems*, pages 318–325. Springer Berlin Heidelberg, 2005.
- [5] Gai-Ge Wang, Suash Deb, and Leandro dos S. Coelho. Elephant herding optimization. In *2015 3rd International Symposium on Computational and Business Intelligence (ISCBI)*. IEEE, dec 2015.
- [6] Sudip Mandal. Elephant swarm water search algorithm for global optimization. *Sādhanā*, 43(1), jan 2018.
- [7] Tarek Chaari. A genetic algorithm for robust scheduling : application of hybrid flow shop scheduling problem. 03 2010.
- [8] Sara Bouraine. Contribution à la planification de mouvements en environnements dynamiques pour des robots mobiles type voiture : Cas robucar. *Blida: Université de Blida*, page 218, 2016.
- [9] Vladimir Lumelsky and Alexander Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Transactions on Automatic Control*, 31(11):1058–1063, nov 1986.
- [10] Fabio Morbidi. Localisation et navigation de robots, 2014.
- [11] Ibraheem Kasim Ibraheem and Fatin Hassan Ajeil. Multi-objective path planning of an autonomous mobile robot in static and dynamic environments using a hybrid PSO-MFB optimisation algorithm. *CoRR*, abs/1805.00224, 2018.
- [12] Hanson robotics – une entreprise spécialisée dans les robots intelligents, March 22, 2017.
- [13] 5-les domaines d’application de la robotique - la robotique !, 2014.
- [14] Hongwei Tang, Wei Sun, Hongshan Yu, Anping Lin, Min Xue, and Yuxue Song. A novel hybrid algorithm based on PSO and FOA for target searching in unknown environments. *Applied Intelligence*, jan 2019.

- [15] Masoud Dadgar, Shahram Jafari, and Ali Hamzeh. A PSO-based multi-robot co-operation method for target searching in unknown environments. *Neurocomputing*, 177:62–74, February 2016.
- [16] Monica Anderson and Nikolaos Papanikolopoulos. Implicit cooperation strategies for multi-robot search of unknown areas. *Journal of Intelligent and Robotic Systems*, 53(4):381–397, June 2008.
- [17] Avinash Gautam and Sudeept Mohan. A review of research in multi-robot systems. In *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*. IEEE, aug 2012.
- [18] Aaron Staranowicz and Gian Luca Mariottini. A survey and comparison of commercial and open-source robotic simulator software. In *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments - PETRA 11*. ACM Press, 2011.
- [19] Lynne E. Parker. Distributed algorithms for multi-robot observation of multiple moving targets. *Autonomous Robots*, 12(3):231–255, 2002.
- [20] Simon Fong, Suash Deb, and Ankit Chaudhary. A review of metaheuristics in robotics. *Computers & Electrical Engineering*, 43:278–291, apr 2015.
- [21] Hannaneh Najd Ataei, Koorush Ziarati, and Mohammad Eghtesad. A BSO-based algorithm for multi-robot and multi-target search. In *Recent Trends in Applied Artificial Intelligence*, pages 312–321. Springer Berlin Heidelberg, 2013.
- [22] Habiba Drias. *Swarm Intelligence: Swarm Intelligence and Evolutionary Algorithms*. 2017.
- [23] Lyes Abada and Saliha Aouat. Tabu search to solve the shape from shading ambiguity. *International Journal on Artificial Intelligence Tools*, 24(5), 2015.
- [24] Drifa Hadjidj and Habiba Drias. Grasp and guided local search for the examination timetabling problem. *IJAISC*, 2(1/2):103–114, 2010.
- [25] Nassima Aleb and Samir Kechid. Automatic test data generation using a genetic algorithm. In *Computational Science and Its Applications - ICCSA 2013 - 13th International Conference, Ho Chi Minh City, Vietnam, June 24-27, 2013, Proceedings, Part II*, pages 574–586, 2013.
- [26] Habiba Drias. Genetic algorithm versus memetic algorithm for association rules mining. In *2014 Sixth World Congress on Nature and Biologically Inspired Computing, NaBIC 2014, Porto, Portugal, July 30 - August 1, 2014*, pages 208–213, 2014.
- [27] Dalila Boughaci, Belaid Benhamou, and Habiba Drias. Scatter search and genetic algorithms for MAX-SAT problems. *J. Math. Model. Algorithms*, 7(2):101–124, 2008.
- [28] Sadjia Benkhider and Habiba Drias. A new memetic approach for the classification rules extraction problem. *IJDMMM*, 5(4):318–332, 2013.
- [29] Wassila Guendouzi and Abdelmadjid Boukra. A manhattan distance-based binary bat algorithm vs. integer ant colony optimisation for intrusion detection in the audit trails. *IJCSE*, 18(4):424–437, 2019.

- [30] Kamel Eddine Heraguemi, Nadjet Kamel, and Habiba Drias. Association rule mining based on bat algorithm. In *Bio-Inspired Computing - Theories and Applications - 9th International Conference, BIC-TA 2014, Wuhan, China, October 16-19, 2014. Proceedings*, pages 182–186, 2014.
- [31] Ilyes Khennak and Habiba Drias. Bat-inspired algorithm based query expansion for medical web information retrieval. *J. Medical Systems*, 41(2):34:1–34:16, 2017.
- [32] Habiba Drias, Moufida Rahmani, and Manel Khodja. ACO approaches for large scale information retrieval. In *World Congress on Nature & Biologically Inspired Computing, NaBIC 2009, 9-11 December 2009, Coimbatore, India*, pages 713–718, 2009.
- [33] Ilyes Khennak and Habiba Drias. An accelerated PSO for query expansion in web information retrieval: application to medical dataset. *Appl. Intell.*, 47(3):793–808, 2017.
- [34] Ilyes Khennak and Habiba Drias. A firefly algorithm-based approach for pseudo-relevance feedback: Application to medical database. *J. Medical Systems*, 40(11):240:1–240:15, 2016.
- [35] Habiba Drias and Hadia Mosteghanemi. Bees swarm optimization based approach for web information retrieval. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2010, Toronto, Canada, August 31 - September 3, 2010, Main Conference Proceedings*, pages 6–13, 2010.
- [36] Riadh Belkebir and Ahmed Guessoum. A hybrid bso-chi2-svm approach to arabic text categorization. In *ACS International Conference on Computer Systems and Applications, AICCSA 2013, Ifrane, Morocco, May 27-30, 2013*, pages 1–7, 2013.
- [37] Youcef Djenouri, Habiba Drias, Zineb Habbas, and Hadia Mosteghanemi. Bees swarm optimization for web association rule mining. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, Macau, China, December 4-7, 2012*, pages 142–146, 2012.
- [38] Yassine Drias, Samir Kechid, and Gabriella Pasi. Bee swarm optimization for medical web information foraging. *J. Medical Systems*, 40(2):40:1–40:17, 2016.
- [39] Souhila Sadeg, Habiba Drias, Ouassim Ait El Hara, and Ania Kaci. ABSO: advanced bee swarm optimization metaheuristic and application to weighted MAX-SAT problem. In *Brain Informatics - International Conference, BI 2011, Lanzhou, China, September 7-9, 2011. Proceedings*, pages 226–237, 2011.
- [40] El-Amine Zemali and Abdelmadjid Boukra. CS-ABC: a cooperative system based on artificial bee colony to resolve the DNA fragment assembly problem. *IJDMB*, 21(2):145–168, 2018.
- [41] Thomas D. Seeley, Scott Camazine, and James Sneyd. Collective decision-making in honey bees: how colonies choose among nectar sources. *Behavioral Ecology and Sociobiology*, 28(4), April 1991.
- [42] Pinar Civicioglu and Erkan Besdok. A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. *Artificial Intelligence Review*, 39(4):315–346, jul 2011.
- [43] Xin-She Yang. A new metaheuristic bat-inspired algorithm. In *Nature Inspired Cooperative Strategies for Optimization (NISCO 2010)*, pages 65–74. Springer Berlin Heidelberg, 2010.

- [44] Xin-She Yang. Flower pollination algorithm for global optimization. In *Unconventional Computation and Natural Computation*, pages 240–249. Springer Berlin Heidelberg, 2012.
- [45] Dan Simon. Biogeography-based optimization. *IEEE Transactions on Evolutionary Computation*, 12(6):702–713, dec 2008.
- [46] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.
- [47] David Edward Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1998.
- [48] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 5, pages 90–98. Institute of Electrical and Electronics Engineers, 1985.
- [49] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, mar 1997.
- [50] Hansruedi Bühler. Réglage par logique floue. 1994. 204 pages, 16x24 cm.
- [51] Yifan Cai and Simon X. Yang. A POTENTIAL-PSO APPROACH TO COOPERATIVE TARGET SEARCHING OF MULTI-ROBOTS IN UNKNOWN ENVIRONMENTS. *International Journal of Robotics and Automation*, 28(4), 2013.
- [52] Alonzo Kelly and Anthony Stentz. Rough terrain autonomous mobility—part 1: A theoretical analysis of requirements. *Autonomous Robots*, 5(2):129–161, 1998.
- [53] Jack Elton Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.