TP11 : réalisation d'un algorithme génétique (AG) en C

*Le but du TP est de réaliser un AG pour un problème d'optimisation difficile, le problème du sac à dos.*

1. **le problème du sac à dos**

Un transporteur veut transporter objets indicés de 1 à dans un camion limité à kg. L'objet a une valeur (prix du transport payé par le propriétaire) et un poids . On ne peut pas prendre tous les objets car leur poids total dépasse . Il faut donc choisir les objets transportés, pour maximiser le revenu sans dépasser la capacité du camion. Ce problème est appelé problème du sac à dos.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exemple | |  | Objet | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  | 7 |  | Valeur | 70 | 20 | 39 | 37 | 7 | 5 | 10 |
|  | 50 |  | Poids | 31 | 10 | 20 | 19 | 4 | 3 | 6 |

On utilise dans la suite le petit exemple ci-dessus. La solution optimale prend les objets 1 et 4 : le revenu est 107 et le camion est plein. Sur des cas plus grands, le camion n'est pas toujours plein et il ne contient pas forcément les objets de plus grande valeur. Il y a sous-ensembles possibles d'objets : on peut les tester tous pour trouver le meilleur, mais cela peut prendre trop de temps : si = 50, on a environ chargements possibles (un million de milliards). Le problème est donc difficile à résoudre, mais on peut calculer une bonne solution, souvent optimale, avec un algorithme génétique (AG).

1. **RAPPELS SUR L'ALGORITHME génétique DE BASE**

L'AG de base code une solution d'un problème par un vecteur binaire appelé *chromosome*. Il part d'une matrice de chromosomes, la *population*, remplie avec des chromosomes générés au hasard (une ligne = un chromosome). à chaque chromosome est associée une mesure d'intérêt appelée *fitness*. Ensuite, l'AG fait des itérations, les *générations*. Chacune génération un opérateur de *croisement* à des chromosomes de choisis au hasard (les *parents*), mais en favorisant les meilleurs. Les chromosomes obtenus (les *enfants*) sont placés dans une nouvelle population de même taille. Quand est pleine, on copie ses chromosomes dans et on recommence pour la génération suivante.

Après un certain temps, on voit apparaitre des chromosomes identiques (*clones*), ce qui nuit à la diversité de la population. Après chaque croisement, avec une petite probabilité, on applique donc à l'enfant un autre opérateur, la *mutation*, qui modifie au hasard une petite partie du chromosome. On apporte ainsi du "sang neuf" à la population, ce qui réduit le nombre de clones.

Voici l'algorithme général. On doit détailler les étapes soulignées, qui dépendent du problème à traiter.

initialisation de la population avec chromosomes générés aléatoirement

déplacer le meilleur chromosome dans

pour de 1 à (nombre choisi de générations)

copier la meilleure solution actuelle dans pour ne pas la perdre

pour chaque enfant de 2 à

sélection de deux parents et au hasard mais en favorisant les meilleurs

croisement de et pour générer un enfant rangé dans

avec une petite probabilité donnée , mutation de

fin pour

ici la population est pleine, la copier dans

déplacer le meilleur chromosome de dans

afficher les statistiques de cette itération/génération (meilleure solution…)

fin pour

sortir la meilleure solution obtenue .

Si on fait assez de générations, l'AG améliore lentement les chromosomes : le fitness moyen augmente. Ce processus fonctionne car la sélection qui favorise les parents de fitness élevé donne souvent des enfants de meilleur fitness, voire une nouvelle meilleure solution. Mais les mauvais chromosomes ne doivent pas être exclus des croisements car ils peuvent contenir un morceau de solution optimale.

1. **ADAPTATION DE L'AG AU Problème du sac à dos**

Important. Comme en maths, on va utiliser des tableaux indicés à partir de 1. Ainsi, on déclare comme suit un vecteur de entiers en C : int V[n+1]; et on utilise les éléments V[1] à V[n] mais jamais V[0]. La population se déclare donc : int P[m+1][n+1]; pour chromosomes et objets.

Codage des chromosomes. Un chromosome (solution possible) est un vecteur de entiers binaires indicé de 1 à . L'élément vaut 1 si l'objet est pris. Dans l'exemple suivant, on prend les objets 2, 4, 5. Avec les tableaux et , on en déduit le fitness (ici le revenu, 64) et la charge du camion (33).

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Objet : | 1 | 2 | 3 | 4 | 5 | 6 | 7= |  | Fitness |  | Charge |
| Chromosome : | 0 | 1 | 0 | 1 | 1 | 0 | 0 |  | 64 |  | 33 |

Codage de la population. C'est simplement une matrice de lignes et colonnes, indicées à partir de 1. Chaque ligne contient un chromosome avec si l'objet est pris, 0 sinon. est flanquée par deux vecteurs (fitness) et (loading). Voici un exemple avec chromosomes :

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7= |  |  |  |  |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |  | 25 |  | 13 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |  | 80 |  | 37 |
| 3 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |  | 64 |  | 33 |
| 4 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |  | 51 |  | 42 |
| 5 = | 1 | 0 | 0 | 1 | 0 | 0 | 0 |  | 107 |  | 50 |

Initialisation. On génère aléatoirement chaque chromosome d'indice , et on calcule son fitness et sa charge . Voici l'algorithme théorique pour un chromosome :

initialisation : mettre à zéro la ligne de , et

faire

// Liste des objets-candidats (pas déjà pris et respectant la capacité)

pour variant de 1 à si et

fin pour

// Tirage au sort d'un objet-candidat dans

tant que .

*Commentaires*. L'idée est de prendre des objets au hasard, tant que c'est possible (), car ceci augmente le revenu et donne une solution assez bonne. Le "faire … tant qu'un test est vrai" se traduit en C par "do { … } while (test);". est une fonction fournie pour ce TP; elle renvoie un entier aléatoire dans . On répète ce processus pour remplir . Quand est pleine, on déplace le meilleur chromosome (le n° 5 de revenu 107 dans l'exemple) sur la ligne 1 : à la fin de chaque itération et à la fin du programme, la meilleure solution trouvée sera donc .

Sélection des parents. Le premier parent est tiré au sort par la méthode du tournoi binaire : on tire au sort 2 indices de chromosomes et dans selon une loi uniforme : est celui qui a le revenu le plus grand. C'est pourquoi il faut mémoriser le fitness (revenu) de chaque chromosome . Cette méthode sélectionne les meilleurs parents avec une probabilité plus grande. Mais les tests montrent qu'on favorise trop ces parents si on choisit aussi par tournoi. On choisit donc uniformément dans , ce qui conduit en moyenne une meilleure solution finale. Algorithme :

tirer au sort dans par tournoi binaire :

faire tant que

si alors sinon fin si

faire tant que

Noter que les deux boucles ont pour but de tirer au sort deux chromosomes distincts.

Croisement de et . On utilise un *croisement simple à un point de coupure*. Le point de coupure est un indice devant lequel on coupe les parents. L'enfant sur la ligne de est formé des objets du parent 1, avant , et des objets du parent 2 à partir de . Donc doit être supérieur à 1 sinon l'enfant est identique à ! Voici l'algorithme qui calcule en parallèle le fitness et la charge de l'enfant :

initialisations : mettre à zéro la ligne de , et

=

pour variant de 1 à si // On copie l'objet s'il est dans

=

ajouter à et à

fin pour

pour variant de à si // On copie l'objet s'il est dans

=

ajouter à et à

fin pour

Exemple avec la population donnée plus haut, deux parent et , et :

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | *cut*=3 | 4 | 5 | 6 | 7= |  |  |  |  |
|  | 1 | 0 | 0 | 0 | 0 | 0 | 1 |  |  |  |  |
|  | 0 | 0 | 1 | 0 | 1 | 1 | 0 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| Enfant | 1 | 0 | 1 | 0 | 1 | 1 | 0 |  | 121 |  | 58 |

On note un problème : *l'enfant viole la capacité du camion* (50)! On pourrait le "réparer" pour respecter la capacité mais il deviendrait trop différent des parents et l'AG fonctionnerait moins bien. On va donc utiliser une technique classique : on autorise des violations mais en les pénalisant. Pour cela, on soustrait une pénalité au fitness pour le rendre moins intéressant. Les solutions infaisables en terme de capacité seront ainsi progressivement éliminées par l'AG à cause de leur fitness trop faible.

En pratique, on définit un facteur de pénalité comme, exemple et on ajoute le terme à si la capacité est violée (). Plus la violation est importante, plus le fitness sera dégradé. Dans l'exemple ci-dessus, le fitness devient 121 - (58 - 50) x 1000 = -7879.

Mutation de l'enfant

On utilise la technique du "bit flip" : on choisit un indice dans et on inverse le bit . Pour le problème, ceci revient à enlever l'objet s'il est dans la camion ou à l'ajouter s'il n'y est pas. Attention! Il faut enlever la pénalite avant l'opération si et la recalculer après.

1. **CODE pré-rempli FOURNI**

Des fichiers sont déjà écrits pour vous faciliter le travail, ils sont sur le site E-Learning/SY15 :

* "randgen.h" est le générateur aléatoire avec la fonction . Le générateur est déjà initialisé.
* "base.cpp" est un squelette de programme principal. Il contient un #include "randgen.h".
* "test-10-269.txt" est un fichier de données pour tester votre AG, avec et .
* Il y a aussi deux tests plus gros "test-20-878.txt" et "test-100-1000.txt".
* Les fichiers contiennent et sur la 1ère ligne, puis lignes contenant chacune et .
* Créez un projet C avec CodeBlocks et télécharger les 5 fichiers dans le dossier du projet.
* Copiez le texte de "base.cpp" dans le "main" de votre projet C et ajoutez "randgen.h" au projet.

Important :

* Les variables fournies sont globales (avant le main), elles sont visibles par toutes les procédures.
* Tous les tableaux sont indicés à partir de 1 et dimensionnés en conséquence.

Le squelette définit d'abord des constantes et des noms de types pour les tableaux :

* "#define nmax 110" est le nombre maximum d'objets accepté dans les fichiers de données.
* "#define m 30" est la taille des populations (nombre de chromosomes).
* "ovect" est un nouveau nom de type pour des tableaux indicés par les objets, comme .
* "cvect" est un nouveau nom de type pour des tableaux indicés par les chromosomes, comme .
* "popu" est un nouveau nom de type pour les populations.

Il précise ensuite les données et les paramètres de l'algorithme (que vous pouvez modifier) :

* FN (file name) : nom de fichier, vous pouvez choisir un des 3 fichiers en enlevant les commentaires.
* Les données trouvées dans le fichier : , , et (valeurs et poids des objets, toujours des entiers). et sont déclarés avec les éléments 0 à mais il faut utiliser seulement les indices 1 à .
* pmuta = 0.05 : probabilité de mutation.
* penal = 1000 : pénalité en cas de dépassement de la capacité.
* niter = 2000 : nombre d'itérations ou générations (il en faut beaucoup pour avoir un bon résultat!).
* frequ = 50 : fréquence du message de statistiques (= un toutes les 50 générations).
* track = 0 ou 1 : 1 écrit des informations détaillées (population initiale, parents choisis…).
* check = 0 ou 1 : 1 appelle une procédure CheckChro qui teste si les chromosomes sont sans erreurs.

Puis il déclare les principales variables :

* La population avec les vecteurs et (fitness et charge de chaque chromosome).
* La population qui reçoit les enfants, avec ses vecteurs et .

Puis des procédures utilitaires. SI vous n'êtes pas spécialiste du C, n'essayez pas de les comprendre, contentez-vous de les utiliser. évitez les modifier sous peine d'erreurs graves à l'exécution :

* CheckChro (Pop, Fit, Load, c) : vérifie le chromosome c d'une population Pop avec des fitness dans Fit et des charges dans Load. Elle stoppe le programme en cas d'erreur. Exemple pour vérifier le chromosome 3 de P : "CheckChro (P, F, L, 3);". Pour le n° 2 de NP : "CheckChro (NP, NF, NL, 2);".
* CopyChro (P1, F1, L1, c1, P2, F2, L2, c2). Copie le chromosome d'index d'une population (fitness dans et charge dans à la ligne d'une population (fitness dans et charge dans ). Le fitness et la charge sont aussi copiés. Exemple, transférer le meilleur chromosome (indice 1) sur la ligne 1 de : CopyChro (P,F,L,1,NP,NF,NL,1).
* Error (Msg) : écrit un message (chaine de caractères) Msg puis stoppe le programme.
* LoadFile () : charge les données du fichier de nom spécifié par FN dans et .
* Pause (message) : écrit un message Msg et attend une touche pour continuer.
* WriteChro (Pop, Fit, Load, c, wait) : affiche le contenu, le fitness et la charge du chromosome n° c de la population Pop avec des fitness dans Fit et des charges dans Load. Wait = 0 ou 1, si 1 il y a une pause pour qu'on ait le temps de lire. Exemple, afficher le chromosome 4 de avec une pause : "WriteChro (P, F, L, 4, 1);".
* WritePopu (Pop, Fit, Load) : liste tous les chromosomes d'une population Pop avec des fitness dans Fit et des charges dans Load. Appelle WriteChro pour chaque chromosome et fait une pause à la fin. Exemple, afficher les chromosomes de : WritePopu (NP, NF, NL).
* WriteStats : affiche une ligne de stats sur la population toutes les "frequ" générations. Pour les chromosomes faisables (qui respectent la capacité), elle affiche le minimum des revenus (meilleur solution provisoire), le maximum, la moyenne (avg) et l'écart dev = moyenne/min qui constitue une mesure de la diversité de la population. Elle écrit aussi le nmbre de chromosomes infaisables.

Enfin, le code contient la structure générale de l'AG.

1. **TRAVAIL à FAIRE**

Compléter les procédures vides qui sont dans le code. Les calculs sont expliqués dans la section III. Les parties à remplir sont indiquées par un commentaire // Taper le code manquant ici. Les appels à la procédure de vérification CheckChro sont déjà insérés.

* FillPopu () : remplit la population initiale avec des chromosomes aléatoires, et aussi les tableaux (fitness de chaque chromosome) et (charge). Ces chromosomes respectent la capacité.
* PutBestFirst () : déplace le meilleur chromosome de à l'indice 1 (avec son fitness et sa charge).
* int Tournament () : renvoie un n° de parent de avec la méthode du tournoi binaire.
* Crossover (p1, p2, c) : effectue un croisement sur les chromosomes-parents d'indices et dans et range l'enfant obtenu sur la ligne de , en calculant aussi son fitness et sa charge Attention au calcul des pénalités si l'enfant viole la capacité du camion!
* Mutation (c) : applique une mutation aléatoire de type "bit flip" à l'enfant d'indice de . Mettre à jour le fitness et la charge. Attention à la pénalité qui peut changer suite à la mutation!

Puis déboguer l'algorithme sur le petit problème "test-10-269.txt". Paramètres pour ce test :

* Mettre track, check et frequ à 1 pour afficher le détail de chaque itération et vérifier les résultats.
* Utiliser une très petite population (#define m 5)
* Et faire seulement 2 ou 3 itérations (générations) : int niter = 2 ou 3.
* éventuellement, vous pouvez mettre la mutation en commentaires pour ce test.

Le résultat va être mauvais (pas assez d'itérations) mais vous allez pouvoir détecter des erreurs.

Si les 2-3 itérations de test se passent bien, testez ensuite avec les paramètres recommandés :

* niter = 2000.
* track = 0 pour éviter le déluge de détails et afficher seulement les statistiques des itérations..
* frequ = 50 pour éviter 2000 lignes de statistiques (vous en aurez une toutes les 50 itérations).
* Mais laissez check à 1 (peu coûteux) pour détecter d'éventuelles erreurs qui restent dans les calculs.