FILIERE MP : ENS (PARIS) – ENS LYON – ENS CACHAN

PAGE DE GARDE DU RAPPORT DE TIPE 2013

NOM : BERENGER Prénoms : Marc, Stéphane, Vincent

Lycée : Thiers

Classe : MP*2

Ville : Marseille

Concours auxquels vous êtes admissible dans la banque inter-ENS :

(Mettre une croix très visible dans la ou les case(s) vous concernant)

ENS Cachan	MP - option MP		MP - option MPI	
	Informatique			
ENS Lyon	MP - option MP		MP - option MPI	X
	Informatique - option M		Informatique - option P	
ENS (Paris)	MP - option MP		MP - option MPI	
	Informatique			

Matière dominante du TIPE :
(mathématiques, informatique ou physique) Informatique

Titre du TIPE : Algorithmes génétiques et résolution approchée de problèmes NP-complets

Nombre de pages (à porter dans les cases ci-dessous) :

Texte	T	5	Illustrations	I	5	Bibliographie	B	1

Résumé imprimé (6 lignes) :

Le but de ce TIPE est d'étudier l'utilisation d'algorithmes génétiques dans la résolution de problèmes complexes. Après avoir présenté les différents opérateurs et mécanismes de ce type d'algorithme, nous nous intéresserons à la convergence de ce dernier vers la solution recherchée, et enfin nous étudierons le cas concret du problème du voyageur de commerce et de sa mise en œuvre en langage Caml.

A Marseille, le 06/06/13
Signature du (de la) candidat(e)

Signature du professeur responsable de la classe préparatoire dans la discipline

Cachet de l'établissement

LYCÉE THIERS
5, Place du Lycée Thiers
13232 MARSEILLE Cedex 1

Algorithmes génétiques et résolution approchée de problèmes NP-complets

Introduction :

Des problèmes d'optimisation célèbres pour leur simplicité d'expression et leur complexité de résolution sont souvent ramenés à des problèmes dits de classe NP-complets grâce à une reformulation du problème d'optimisation en problème de décision. De nos jours, aucune méthode déterministe « efficace » ne permet de résoudre ce genre de problème, si bien que des algorithmes de résolution approchée se sont développés et proposent de calculer en un temps polynomial une solution correspondant au mieux à l'optimal recherché.
Dans ce TIPE, je vais m'intéresser aux algorithmes génétiques, qui reposent sur la théorie de l'évolution darwiniste. L'utilisation de ce type d'algorithme a de nombreux avantages. Outre le fait de contrôler le temps de résolution, il permet de trouver des solutions en n'ayant à priori aucune connaissance de ces dernières, ce qui en fait un algorithme totalement polyvalent.

I) Le principe de l'algorithme génétique

1) Description

L'algorithme va reproduire fidèlement l'évolution des êtres vivants, en s'inspirant des modèles génétiques actuels et en faisant évoluer une population de solution. Ainsi, il faut définir la représentation des solutions sous forme de code, puis la donnée d'une fonction de fitness d'une solution, qui sera la grandeur à optimiser. Le but de l'algorithme sera d'améliorer la fitness de la population globale de génération en génération, afin de tendre vers un extremum de la fonction de fitness et donc vers l'optimum recherché. Un individu solution est alors totalement déterminé par son caryotype, et grâce aux mécanismes génétiques, ce caryotype va évoluer pour donner un individu plus performant.

Au cours d'une génération, plusieurs changements vont se succéder pour améliorer la fitness de la population :

- Sélection naturelle : grâce à un opérateur, on choisit, en fonction de la fitness d'une solution, si cette dernière sera éliminée ou pas de la population
- L'opérateur de croisement ou crossing-over : afin d'améliorer la vitesse de convergence de l'algorithme, on fait se « reproduire » deux individus solutions pour obtenir un individu fils qui vient remplacer les individus éliminés lors de la sélection naturelle
- L'opérateur de mutation : absolument indispensable, ce dernier garanti la convergence de l'algorithme vers l'optimum. Ayant une probabilité $p > 0$ d'intervenir, cet opérateur change aléatoirement le caryotype d'un individu, permettant ainsi de conserver la diversité de la population.

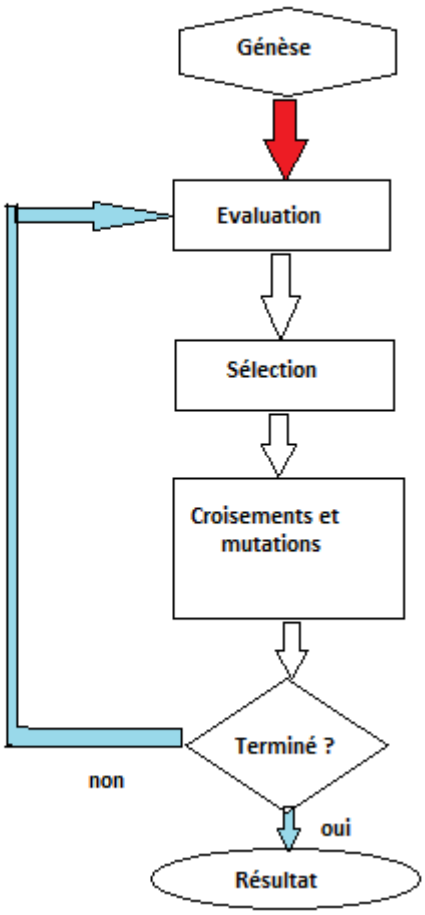


Schéma récapitulatif de l'algorithme génétique

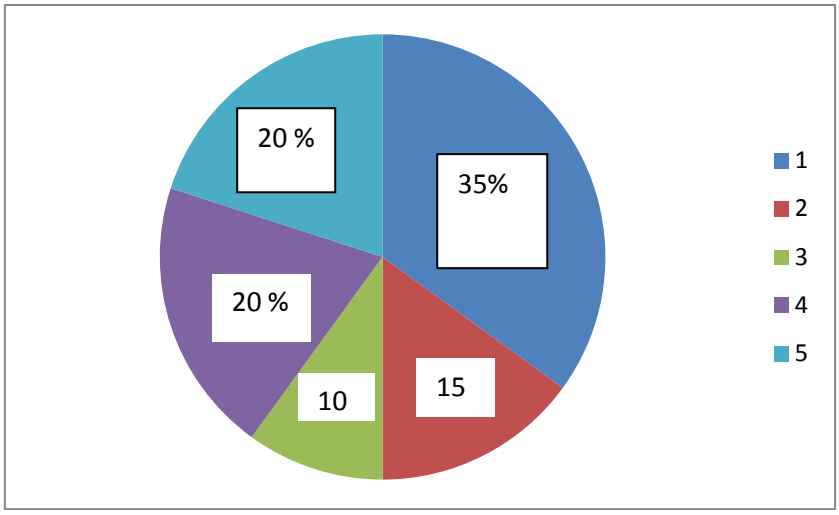
Dans la suite de cet exposé, nous noterons les populations d'individus-solutions P, et n le nombre d'individus composant cette population.

2) L'opérateur de sélection

Cet opérateur est chargé de définir quels seront les individus d'une population P qui survivront et seront présents dans la population suivante. Cet opérateur est très important puisqu'il va déterminer l'efficacité de l'algorithme en choisissant qui survit et qui meurt, en fonction de sa fitness. Plusieurs méthodes existent et présentent chacune avantages et inconvénients.

a) La loterie biaisée

Avec cette méthode, chaque individu a une chance de survivre proportionnelle à sa fitness, c'est-à-dire à son adaptation au problème. Ainsi, si l'on représente la « roue du forain », chaque individu se voit attribuer un secteur de celle-ci proportionnel à sa fitness, et en simulant un lancé aléatoire, on détermine le secteur sur lequel la roue s'arrête.



Lorsque l'on fait tourner la roue, c'est l'individu 1 qui a le plus de chance d'être sélectionné, mais il se peut que ce soit l'individu 3, avec la plus basse fitness, qui survive. Se pose alors l'inconvénient majeur de ce mode de sélection : la croissance de la fitness globale n'est pas assurée : ainsi, il se peut que nos meilleurs individus, voire l'optimum, soient éliminés d'une génération sur l'autre, ce qui va à l'encontre même du principe de l'algorithme. Le second inconvénient, mais qui est récurrent chez tous les opérateurs de sélection, est la convergence prématurée de l'algorithme vers un extremum local. En effet, le fait de conserver les meilleurs individus va porter atteinte à la diversité de la population, si bien que les élites d'une population vont devenir majoritaires et l'extremum ne sera pas atteint, sauf peut-être grâce aux mutations.

b) La méthode élitiste

Certainement la plus facile à mettre en place, cette méthode consiste à trier par ordre de fitness décroissante les n individus de P et de n'en garder qu'un pourcentage prédéfini, que l'on peut faire varier au fil des générations si on le souhaite. Ainsi, les individus les moins adaptés sont éliminés. Contrairement à la loterie biaisée, cette méthode a une variance nulle, mais le phénomène de convergence prématurée est accru du fait d'une diversité quasi-inexistante. Il faut donc jouer sur les paramètres de mutation et de croisement pour conserver cette diversité.

c) La sélection par tournoi

On effectue n tirages aléatoires avec remise de deux individus de P. On note P_1 celui avec la fitness la plus élevée, P_2 celui avec la fitness la plus basse. On fixe une probabilité p telle que $0.5 < p < 1$ que P_1 batte P_2 lors d'un duel. Ainsi, grâce à un lancer aléatoire, on détermine lequel des deux gagne le duel, et il est alors « cloné » dans une population P'.
Malgré la forte variance de cette méthode, elle permet de préserver une certaine part de diversité, et en modifiant la valeur de p on peut obtenir des solutions efficaces.

3) L'opérateur de croisement ou crossing-over

Suite à l'opérateur de sélection, on ne dispose plus que de $\frac{n}{2}$ individus. L'opérateur de croisement va régénérer la population en combinant les chromosomes de deux individus parents. Pour cela, on transpose le processus naturel génétique à l'informatique. On sélectionne deux parents P_1 et P_2, représentés chacun par un chromosome. On fixe le nombre de « points de croisements ou *loci* » et on décide de « coller » un morceau de P_1 à un autre morceau de P_2 pour obtenir un nouvel individu.

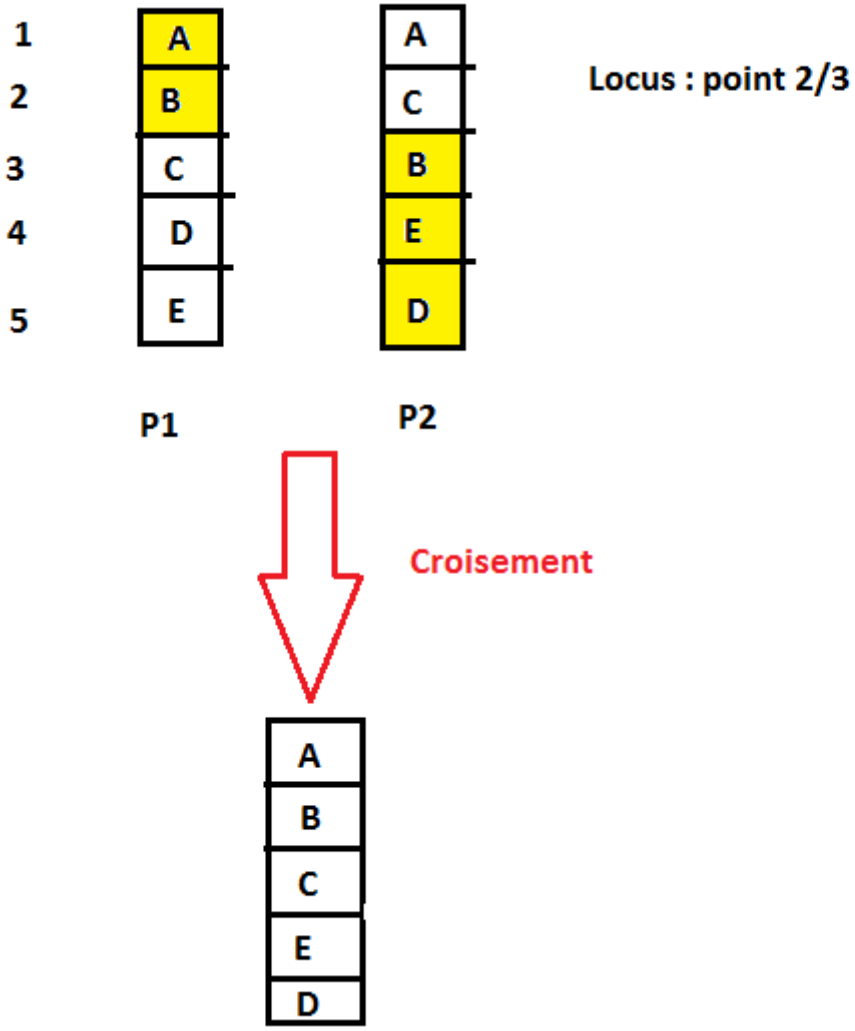


Schéma d'un croisement par crossing-over à un locus

On détermine aléatoirement le point de croisement : ici 2/3. On prend la première moitié du chromosome de P_1 que l'on colle à la deuxième moitié du chromosome de P_2. Notons qu'il y a un doublon sur les cases 2 et 3 : B revient deux fois. On décide de supprimer le deuxième B et de le remplacer par la lettre manquante : C. On obtient donc un fils. En effectuant $\frac{n}{2}$ fois cette opération, on retrouve une population P' de n individus.

L'utilisation des croisements a pour but de favoriser l'exploration de l'espace des solutions et d'apporter de la diversité dans la population après l'opérateur de sélection. Néanmoins, il n'est pas suffisant pour garantir la convergence de l'algorithme vers la solution optimale.

4) L'opérateur de mutation

Considérons une probabilité p > 0 qu'un individu de P subisse une mutation. L'opérateur de mutation va agir sur le chromosome de cet individu avec cette probabilité p, souvent très faible, pour modifier son caryotype de manière aléatoire. Typiquement, cet opérateur va échanger l'ordre de deux allèles sur le chromosome, en choisissant aléatoirement le locus, comme par exemple échanger A et B dans l'individu P_1.

Cet opérateur permet d'introduire aléatoirement de nouvelles solutions, préservant ainsi la diversité au sein de la population. Il permet également de contrer le phénomène de convergence prématurée. Mieux, la mutation garantit la propriété d'ergodicité, qui signifie que chaque point de l'espace de recherche peut être atteint.

II) Convergence de l'algorithme

1) Chaine de Markov : définitions et propriétés

On s'intéresse à l'évolution aléatoire d'un phénomène au cours du temps, que l'on va caractériser par un processus stochastique $\{X_t, t \in T\}$ où X_t est une variable aléatoire indexée par t qui représente un instant. T représente quant à lui un ensemble d'instant, que l'on va supposer infini dénombrable (pratiquement, on considèrera T = N) dans la suite de l'exposé. On appelle S l'espace des états, c'est-à-dire l'ensemble des valeurs que peut prendre X_t au cours du temps. On suppose ici cet ensemble fini : le processus stochastique s'appelle alors chaine.

Définition : chaine de Markov :
On dit qu'une chaine est de Markov si

$\forall\, n \in N,\ \forall\, s_0, s_1, .., s_n\ \in S$:

$$P(X_n = s_n | X_0 = s_0, X_1 = s_1, \dots, X_{n-1} = s_{n-1}) = P(X_n = s_n | X_{n-1} = s_{n-1})$$

Cette définition signifie que pour connaître l'évolution future du processus stochastique, il suffit uniquement de connaître son état présent.

Une chaine de Markov est dite **homogène** si

$$\forall\, n > 0, \forall k\ \geq 0, \forall\, s, \sigma\ \in S, \qquad P(X_n = s\ | X_{n-1} =\ \sigma) = P(X_{n+k} = s\ |\ X_{n+k-1} =\ \sigma)$$

Définition : Matrice de passage :

On suppose que S = {1,…, r}. On note $\forall\, i, j\ \in S, p_{ij} = P(X_n = j\ | X_{n-1} = i)$
Et on définit la matrice de passage $\boldsymbol{P} = (p_{ij})_{1 \leq i,j \leq r}$. ***P*** est une matrice stochastique.

Propriété :$\forall m \geq 0$, la probabilité de passage que i à j en m étapes est égale à l'élément (i ,j) de la matrice $\boldsymbol{P}^{m}$.

Distribution des états à l'instant m : Soit $p_i^{(m)} = P(X_n = i)$. On note distribution après m étapes le vecteur ligne : $\pi^{(m)} = (p_1^{(m)}, \dots, p_r^{(m)})$.

Propriété : $\pi^{(n+1)} = \pi^{(0)} P^{n+1} = \pi^{(n)} P$

Classification : On définit la relation d'équivalence « communiquer » entre les états de S : On dit que l'état j est accessible depuis l'état i si $\exists\, n \in N\,, p_{ij}^{(n)} > 0$. On dit que i et j communiquent si i est accessible depuis j et si j est accessible depuis i. On définit alors les classes d'équivalence formées des éléments qui communiquent entre eux, ce qui permet d'établir leur classification :

- Une classe est transitoire si on peut en sortir, sinon elle est persistante
- Une classe persistante constituée d'un seul élément est absorbante
- Une chaîne est irréductible si elle ne comporte qu'une seule classe
- Une chaîne est absorbante si toutes ses classes sont absorbantes.

Période : la période d d'un état i d'une chaine de Markov est égale au plus grand diviseur commun de tous les n tels que $p_{ii}^{(n)} > 0$. Si d >1 alors i est périodique, sinon il est apériodique.

Théorème : Si S est fini et si une chaîne de Markov est irréductible et apériodique, alors :
Pour toute distribution initiale $\pi^{(0)}$: $\lim_{n\to\infty} \pi^{(n)} = \lim_{n\to\infty} \pi^{(0)} P^n = \pi^*$
De plus, on sait que la distribution π^* est stationnaire.

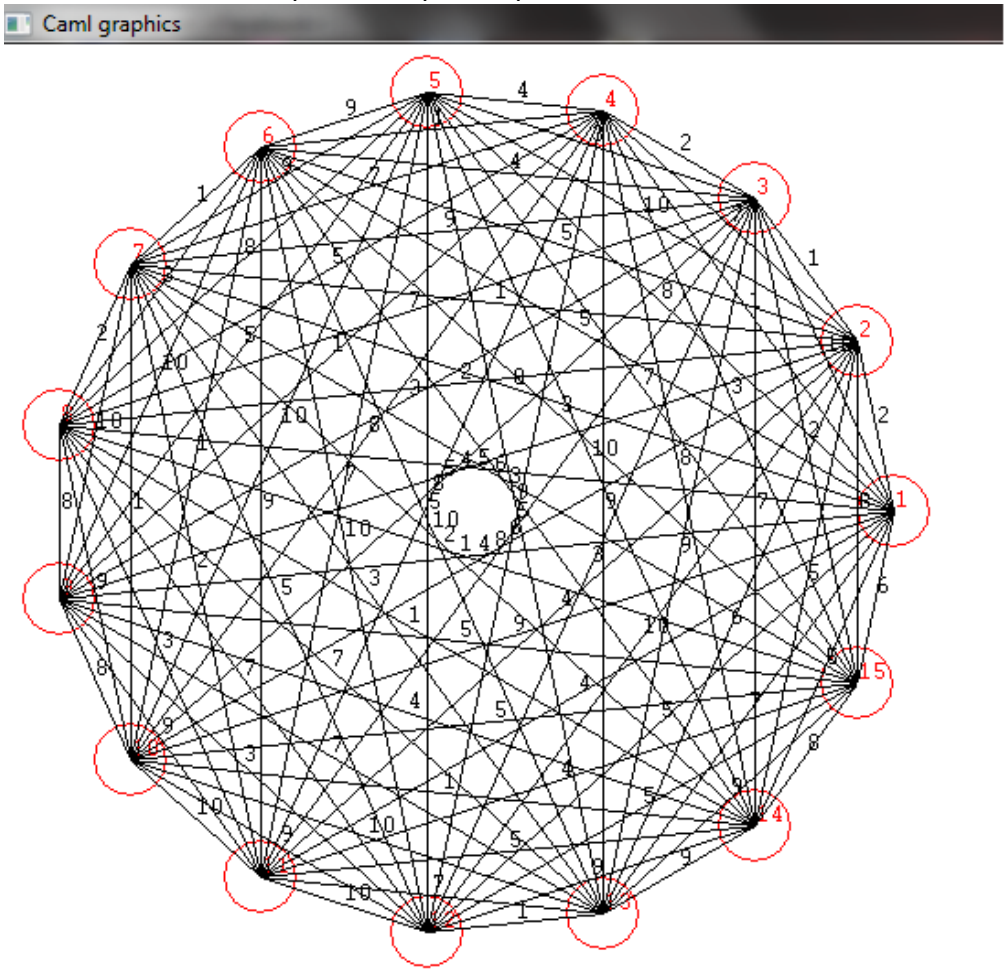
2) Application à l'algorithme génétique

On considère le processus stochastique $\{X_n\}_{n\in N}$ où X_n représente la population obtenue après n utilisations des opérateurs. L'espace des états S est fini et représente l'ensemble de toutes les solutions possibles. D'après la description de l'algorithme dans la section I), l'application des opérateurs ne dépend que de la population courante : il s'agit donc d'une chaîne de Markov. Dans la représentation de notre problème d'optimisation, tous les états sont coaccessibles grâce à l'opérateur de mutation : la chaîne est donc irréductible. Enfin, la chaîne est apériodique en conséquence de l'opérateur de mutation.
D'après le théorème du II)1), l'algorithme converge vers une distribution limite qui est stationnaire : il s'agit de la distribution pour laquelle l'extremum est atteint. Ainsi, la preuve de la convergence théorique de l'algorithme génétique est établie.

III) Etude pratique : le voyageur de commerce

1) Enoncé du problème et représentation

Le problème du voyageur de commerce est un problème d'optimisation typique mais qui illustre parfaitement l'efficacité de l'algorithme génétique. On considère un ensemble de n villes toutes reliées les unes aux autres par un chemin. On affecte à chaque chemin un « poids » représentant la durée du trajet d'une ville à l'autre. Le problème consiste à trouver le circuit de poids minimal passant par chacune des villes, en finissant par la ville d'où l'on est parti. Si on ne fixe pas la ville de départ, il y a (n-1) ! chemins différents possibles. On décide de représenter ce problème par un graphe connexe complet non orienté à n sommets, les chemins étant pondérés par les poids.



Représentation sous forme de graphe du problème avec 15 villes

Si on considère l'exemple précédent, l'exploration systématique de tous les chemins revient à calculer le poids de 14 ! = 87178291200 chemins possibles.
Si on suppose que l'ordinateur fait un calcul par µs, il lui faudrait plus d'un jour pour explorer tous les chemins. Une telle attente n'est pas concevable, ce qui justifie l'utilisation de l'algorithme génétique.

Pour représenter en Caml le problème, on crée le type graphe :

```
type graphe = {ville : int ; aretes :  (int *  int * int)
list};;
```

ville correspond au nombre de ville et aretes est le triplet donnant : la ville i, la ville j, et le poids du chemin de i à j. Pour faciliter l'accès aux données du problème, on le représente également par une matrice m telle que m.(i).(j) = poids du chemin de i+1 vers j+1.La fonction de fitness est alors le poids du chemin emprunté.

On choisit enfin la méthode de sélection élitiste ainsi qu'un croisement à un locus.

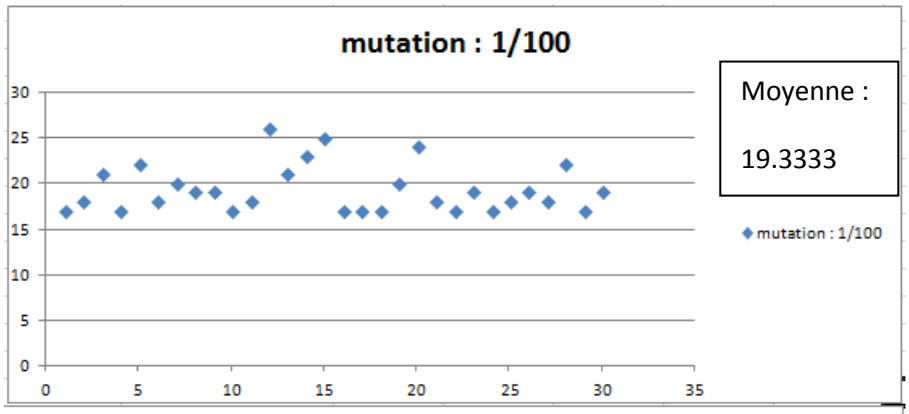
Calcul de la complexité :

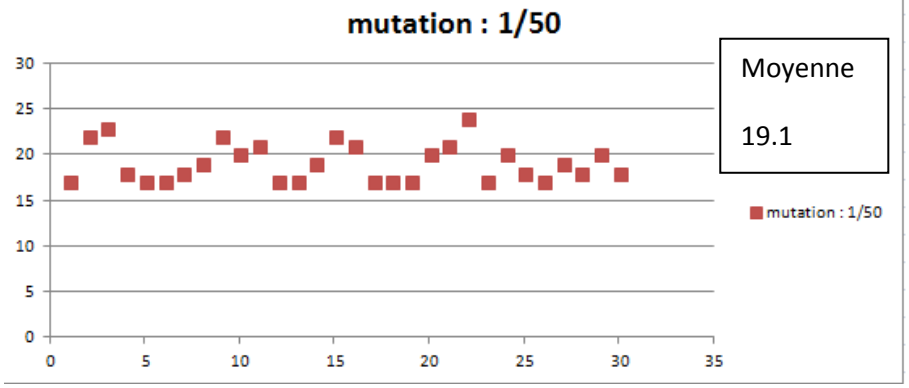
On note n le nombre de villes, p le nombre d'individus de la population, t le nombre d'étapes. La complexité totale du programme (voir annexe) est :

$$O(t(p\log(p)+n+pn)+n^2+pn)$$

2) Etude statistique de l'influence des paramètres

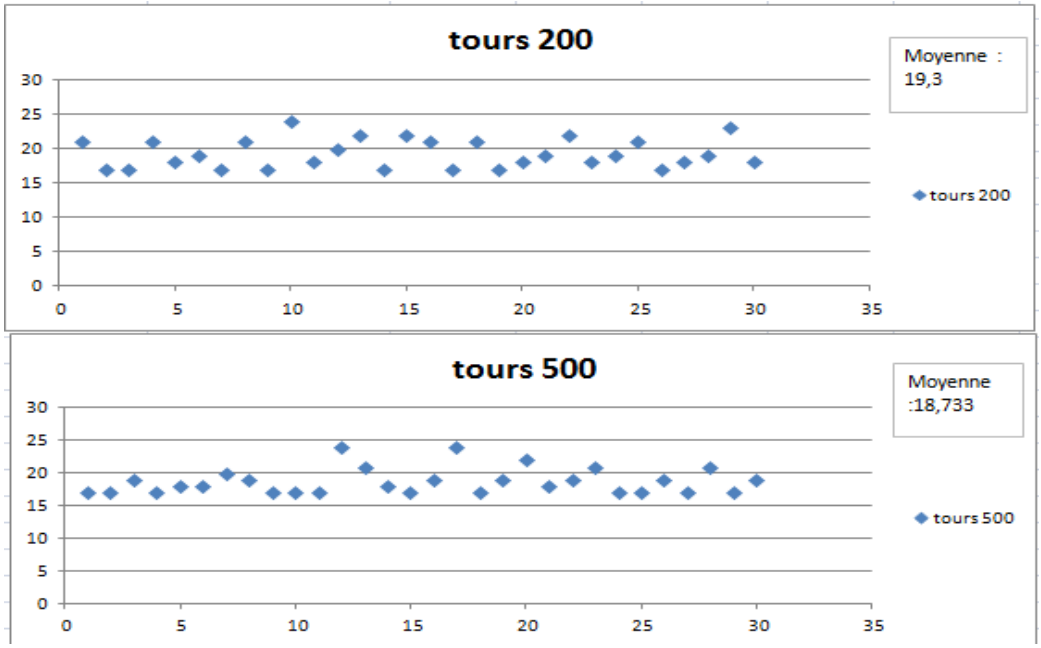
On étudie l'exemple de la figure précédente, problème à 15 villes, dont le chemin de taille minimale a pour longueur 17.
Influence de la mutation :





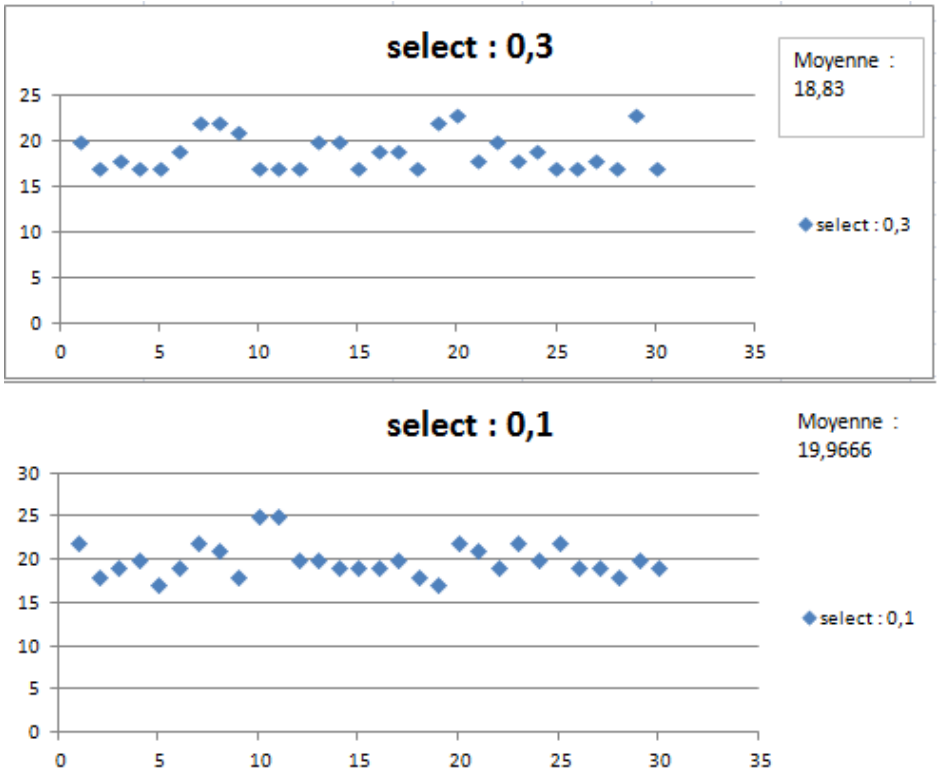
On remarque qu'une mutation ayant une probabilité de 1/50 permet de contrer un petit peu mieux l'effet de convergence prématurée de la solution vers un extremum local.

Influence du nombre de générations :



On voit très nettement que plus il y a de générations, plus la solution approchée est proche de la solution optimale. Néanmoins, cette augmentation du nombre de générations augmente assez significativement le temps de calcul, ce qui peut être regrettable.

Influence du taux de sélection naturelle :



Il s’agit de l’illustration la plus nette du phénomène de convergence prématurée : lorsque la sélection naturelle exerce une pression trop intense sur la population, l’algorithme reste bloqué dans un extremum local, réduisant considérablement son efficacité.

Conclusion de l’étude :

Les algorithmes génétiques transposent à l’informatique l’évolution des êtres vivants, afin de tendre vers la solution optimale du problème posé. Sa polyvalence en fait un moyen de résolution de choix de problèmes très complexes comme les problèmes de classe NP-complet. Néanmoins, un « tâtonnement » est nécessaire pour déterminer empiriquement les différentes variables de l’algorithme et trouver une adéquation entre le temps de calcul et la qualité de la solution approchée.

Annexe

Codage de l'algorithme

Représentation des solutions sous forme de graphe et de matrice :

```
type graphe = {ville : int ; aretes :  (int *  int * int) list};;

let matrix_of_graph graphe =
 let n = graphe.ville in
 let mat = make_matrix n n 0 in
 let rec aux = function
  | [] -> mat
  | (i,j,p)::t -> mat.(i-1).(j-1) <- p ; mat.(j-1).(i-1) <- p; aux t
 in
aux graphe.aretes ;;
```

Création de la population :

```
let creer_individu nb_pts =
 random__init (int_of_float (sys__time ()));
 let rand = ref (random__int (nb_pts) + 1) in
 let i = ref 0 in
 let l = ref [] in
 while !i < nb_pts do
  while mem !rand !l do
    rand := random__int (nb_pts) + 1
  done;
  l:= !rand :: !l ;
  rand := random__int (nb_pts) + 1;
  incr i
 done;
!l;;
let creer_population nbre nb_pts =
  random__init (int_of_float (sys__time ()));
  let rand = ref (random__int (nb_pts) + 1) in
    let i = ref 0 in
      let l = ref [] in
        let w = make_vect nbre [] in
          for k = 0 to nbre - 1 do
            (while !i < nb_pts do
                while mem !rand !l do
                  rand := random__int (nb_pts) + 1
                done;
                l := !rand :: !l;
                rand := random__int (nb_pts) + 1;
                incr i
              done;);
            w.(k) <- !l;

            l := []; i := 0
          done;
          w ;;
let poids_tot l mat =
 let rec aux accu fin = function
  | [] -> accu,fin
  | [x] -> failwith "erreur poids"
  | [x;y] -> (accu+ mat.(x-1).(y-1),y-1)
```

```
  | (x::y::z::t) ->  aux (accu + mat.(x-1).(y-1)) fin (y::z::t) in
 let p,fin = aux 0 0 l in
 p + mat.( hd l-1).(fin)
;;

let pondere pop mat nbre nb_pts =
 let w = make_vect nbre ([],0) in
  for i = 0 to nbre -1 do
  w.(i) <- (pop.(i), poids_tot (pop.(i)) mat)
  done;
 w;;
```

Opérateur de sélection :

```
let selection_naturelle pop pourcent =
  tri_fusion pop;
  let n = vect_length pop in
    let x = int_of_float (pourcent *. (float_of_int n) +. 0.5) in
      for i = x to n - 1 do
        pop.(i) <- ([], 500000000)
      done;
      x
;;
```

Opérateur de croisement :

```
let cross_over l l' =
  random__init (int_of_float (sys__time ()));
  let n = random__int (list_length l) in
    let m = list_length l in
      let aux = ref [] in
        for i = m downto 1 do
          aux := i :: !aux
        done;
        let l1 = ref l and l2 = ref l' and fils = ref [] in
          for i = 0 to n do
            fils := (hd !l1) :: !fils;
            l1 := tl !l1;
            l2 := tl !l2;
          done ;

          let k = ref (n + 1) in
            for i = n + 1 to m - 1 do
              if not (mem (hd !l2) !fils) then (fils := (hd !l2) :: !fils;
                  l2 := tl !l2; incr k)
              else l2 := tl !l2;

            done;
            aux := subtract !aux !fils;
            while !k < m do
              fils := (hd !aux) :: (!fils);
              aux := tl !aux;
              incr k;

            done;
            rev !fils
;;
```

Opérateur de mutation :

```
let mutation l proba =
  random__init (int_of_float (sys__time ()));
  let n = random__int (proba) + 1 in
    if n = proba then (
        let m = list_length l in
          let p = random__int (m - 1) in
            aux [] 0 l where
      rec aux accu acco = function
| [] -> rev (accu)
| [x] -> rev (x :: accu)
| x :: y :: t -> if acco = p then aux (y :: x :: accu) (acco + 1) t
else aux (x :: y :: accu) (acco + 1) t
)
else l
;;
```

Fonction principale :

```
let algo_gen graphe nbre_pop nbre_etape pourcent_select proba =
  let mat = matrix_of_graph graphe in
    let pop = population_pondere mat nbre_pop graphe.ville in
      for i = 1 to nbre_etape do
        tri_fusion pop;
        let x = selection_naturelle pop pourcent_select in
          for i = x to nbre_pop - 2 do
            pop.(i) <- let v = cross_over (fst (pop.(i - x))) (fst (pop.(i
+ 1 - x))) in
                (v, poids_tot v mat)
          done;
          for i = 0 to (seek_vide pop) - 1 do
            pop.(i) <- let v = mutation (fst (pop.(i))) proba in
              (v, poids_tot v mat)
          done;
      done;
      tri_fusion pop;
      pop.(0);;
```

Bibliographie :

Le problème du voyageur de commerce, de Ylfang Li, Yannick Kergosien, Jean-Charles Billaut, http://interstices.info/voyageur-commerce

Algorithmes génétiques, Souquet Amédée, Radet François-Gérard, http://deptinfo.unice.fr/twiki/pub/Linfo/PlanningDesSoutenances20032004/Radet-Souquet.pdf

Introduction aux chaines de Markov, Raymond Moché, http://gradus-ad-mathematicam.fr/documents/404_Markov.pdf

N'aimez-vous pas prévoir le futur parfois ? Epreuve commune de TIPE 2008 (partie D) , http://www.scei-concours.fr/tipe/TIPE_2008/sujets_2008/mathematiques_2008.pdf

Etude des Algorithmes génétiques et application aux données de Protéomique, Christelle Reynes, tel.archives-ouvertes.fr/docs/00/26/89/27/PDF/global.pdf

Chaine de Markov, http://fr.wikipedia.org/wiki/Cha%C3%AEne_de_Markov, date : 05/06/13

Algorithme génétique, http://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique, date : 05/06/13