UPC E.T.S. d'Enginyers de Telecomunicació
Barcelona

PROJECTE FI DE CARRERA

PROJECTE FINAL DE CARRERA

Convocatòria: **Any:** 1992

Alumne: Ramón Roca Mérida

Títol del Projecte: OBTENCIO DE BONS CODIS MITJANÇANT UN ALGORISME GENETIC

Director del projecte:
Francesc P. Comellas Padró

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Escola Tècnica
Superior d'Enginyers
de Telecomunicació

Jordi Girona Salgado s/n
Tels. (93) 204 65 51 204 [illegible]
08034 Barcelona
Apartat 30.002

FULL Nº . 1
CURS: .....

ALUMNE

COGNOMS: ROCA MÉRIDA | NOM: RAMON

PLA: 1964 | ESPECIALITAT:

INFORMACIO SOBRE EL PROJECTE

TITOL: OBTENCIÓ DE BONS CODIS MITJANÇANT UN ALGORISME GENÈTIC

INFORMACIO APROXIMADA DEL CONTINGUT DEL PROJECTE (ABAST I PARTS DEL PROJECTE):

REALITZAT EN COL.LABORACIO AMB ELS SRES.

.....

.....

DIRECTOR DEL PROJECTE SR: FRANCESC J. P. COMELLAS PADRÓ

ADMISSIO A EXAMEN

CONFORMITAT DELS MEMBRES DEL TRIBUNAL

EL PRESIDENT,

EL VOCAL,

EL SECRETARI,

F. Cabrelles

BARCELONA.....

REGISTRE SECRETARIA

(A OMPLIR PER SECRETARIA)

Obtenció de bons codis mitjançant un Algorisme Genètic

P.F.C: Ramón Roca Mérida.

Director: Francesc Comellas.

Departament de Matemàtica Aplicada i Telemàtica de l'E.T.S.E.T.B.

Esquema del Projecte

0. Pròleg. 1

1. Objectius i descripció del projecte. 4

2. Introducció als codis. 7

2.1 Nocions bàsiques de codis. 7
2.1.1 Codis de repetició i de paritat. 8
2.1.2 Codis vectors i distància de Hamming. 10
2.2 Codis de pes constant. 12
2.3 Codis de cobertura. 13
2.3.1 Cobertura utilitzant matrius. 15
2.3.2 Cobertura utilitzant un codi mixt. 16

3. Mètodes de recerca d'extrems. 18

3.1 Mètodes de càlcul. 18
3.2 Mètodes enumeratius. 20
3.3 Mètodes aleatoris. 21

4. Algorismes Genètics. (A.G.). 23

4.1 Algorisme genètic bàsic 26
4.1.1 Selecció. 28
4.1.2 Encreuament. 30
4.1.3 Mutació. 31
4.2 Exemple d'A.G. bàsic. 32

5. Fonaments matemàtics del A.G. 35

5.1 Patrons de semblances (Esquemes). 35
5.2 Qui ha de viure i qui ha de morir (Teorema fonamental). 38

5.3 El problema de la màquina escurabutxaques de k braços. 44
5.4 Quants esquemes són processats útilment. 48

6. Operadors i tècniques avançades. 51

6.1 Escalat de la funció. 51
6.1.1 Escalat lineal. 52
6.1.2 Truncament sigma. 54
6.1.3 Escalat de llei potencial. 54
6.1.4 Escalat de ranking. 55
6.2 Dominància, Diploidicitat i Abeyància. 55
6.2.1 Perspectiva històrica de la dominància i la diploidicitat en els A.G. 58
6.3. Inversions i altres operadors de reordenament. 62
6.4 Perspectiva històrica dels operadors de reordenament en A.G. 64
6.5 Altres microoperadors. 72
6.5.1 Segregació, translocació i estructures multicromosomàtiques. 72
6.5.2 Duplicació i deleció. 73
6.5.3 Determinació sexual i diferenciació. 74
6.6 Els nínxols i l'especiació. 75
6.6.1 Mètodes de nínxols per A.G. 77

7. Obtenció de bons codis utilitzant un A.G. 82

7.1. Aplicació d'un A.G. a l'obtenció d'un codi de pes constant (*Constant weight codes* o C.W.C). 83
7.1.1 Funció de cost en C.W.C. 85
7.1.2 Operants genètics pels C.W.C. 89
7.1.3 Encreuament de C.W.C. 90
7.1.3.1 Encreuament de paraules de pes constant. 95
7.1.4 Mutació de C.W.C. 97
7.1.5 Comentari de les prestacions dels operants emprats. 98

7.2. Aplicació de l'A.G. a l'obtenció de codis de cobertura (*Covering codes* o C.C) 99
7.2.1 Funcions de cost en C.C. 101
7.2.1.1 Generació de paraules a una distància R d'altra. 104
7.2.2 Operants genètics als C.C. 105
7.2.2.1 Encreuament als C.C. 105
7.2.2.2 Mutació als C.C. 106
7.3. Escalat de la funció de cost. 107
7.4 Variables d'execució. 112

8. Conclusions i resultats. 114

8.1 Algorismes implementats. 114
8.1.1 Nou.c 116
8.1.2 Fitxers de resultats. 119
8.1.3 Cover.c, CoverM.c, CoverF4.c 120
8.1.4 Fitxers de resultats 122
8.2 Elecció del tipus d'escalat idoni. 122
8.3 Elecció de les variables d'execució. 125
8.4 Resultats obtinguts. 135
8.4.1 Codis de pes constant 135
8.4.2 Codis de cobertura 136
8.5 Valoració de resultats. 138

Annex I Codis trobats. 140
A.1.1 Codis de pes constant. 146
Annex II. Algorismes implementats. 154
A.2.1 nou.c 154
A.2.2 cover.c 166
A.2.3 coverM.c 175
A.2.4 coverf4.c 181
Bibliografia. 186

Pròleg

Una gran part de les Telecomunicacions actuals està dedicada a l'estudi de la transmissió de dades. Aquestes dades poden ser resultats obtinguts per un ordinador o bé pot ser informació obtinguda de la digitalització d'algun tipus d'informació analògica com un senyal d'àudio o una imatge recollida per una càmara. Aquesta informació normalment tindrà dos possibles destins, per una part podrà ser emmagatzemada en una unitat d'emmagatzematge massiu com un disc dur o un CD ROM o bé podrà ser enviada per un canal d'informació.

Ens convindria que aquesta informació tingués un format el més reduït possible. Per una part estalviaríem en espai físic, per exemple en el disc dur, i per altra, per un mateix canal, podríem enviar més informació en el mateix temps.

Per altra banda, en la lectura i escriptura de dades o en la transmissió d'aquestes per un canal es produeixen errors. Aquests errors poden ser deguts, per exemple, a imperfeccions en el disc o a soroll existent en el canal.

Podem aconseguir millorar aquests dos factors, la representació eficient de la informació i la fiabilitat de la seva transmissió, mitjançant el que en diem codificació. La codificació consisteix en transformar la informació original en un altra utilitzant un conjunt de paraules que en diem codi. Els codis que permeten representar eficientment la informació s'anomenen codis de font i els que ens garanteixen una transmissió fiable, codis correctors d'errors. Els estudis dels codis és doncs, i per aquesta raó, una de les branques més tractades en comunicacions tant teòricament com pràcticament.

Una part important d'aquests estudis està dedicada a trobar fites teòriques de la mida dels codis. Per unes determinades condicions que ha d'acomplir, un bon codi serà el que tingui més paraules o bé el que en tingui menys segons el cas. Per exemple si volem un codi format per paraules binàries diferents, de longitud 4, sense cap més tipus de restricció, podem assegurar que el codi més gran estarà compost per 16 paraules, començarà per la paraula 0000 i acabarà per la paraula 1111.

Per determinar aquestes fites hi han dues línies de recerca que podem utilitzar. Per una part podem estudiar les propietats que

acompleix un determinat tipus de codi i tractar de deduir-ne la fita o bé, i això és el que farem, tractarem de trobar mitjançant mètodes d'optimització un codi no existent fins ara i demostrarem l'existència de la fita amb l'evidència del codi trobat.

Aquesta última via d'investigació serà la que nosaltres utilitzarem. A més, això ens permetrà no només obtenir fites teòriques sinó també obtenir un codi que podrem utilitzar directament.

Per poder trobar un bon codi, és a dir un codi d'una mida propera a la seva fita teòrica, hem de fer us d'eines adequades. En primer lloc necessitem una forma de valorar el codi, hem de trobar una funció que ens indiqui com de bo és, per tal d'anar millorant-lo. Per altra part necessitem un mètode de recerca d'extrems que sigui eficient. Encara que hi ha molts mètodes de recerca hi ha pocs que acompleixin els requisits que necessitem. En primer lloc necessitem un mètode que no necessiti conèixer informació auxiliar de la funció de valoració del codi com ara la seva derivada o alguna altra propietat. I en segon lloc necessitem, donat que els espais de recerca que utilitzarem seran molt grans, que sigui molt eficient.

Actualment, i des de fa poc temps, existeix un tipus de mètodes que acompleix les nostres exigències. D'aquests mètodes en diem mètodes de recerca probabilístics. Aquests mètodes van sorgir com a part del que anomenem intel.ligència artificial i en la seva majoria tenen com a model de funcionament comportaments de la natura. Dins d'aquests, els mètodes més importants són la Recuita Simulada o *Simulated Annealing*, basat en el mecanisme físic del refredament i formació dels cristalls, i els Algorismes Genètics basats en la genètica i l'evolució de les espècies.

La Recuita Simulada ja ha demostrat que pot ser aplicat i amb gran eficiència en l'obtenció de fites teòriques de codis. Nosaltres no aplicarem doncs aquest mètode sinó que estudiarem l'aplicació dels Algoritmes Genètics en problemes en els que la Recuita Simulada ha estat eficient. D'aquesta manera tindrem una forma de valorar perfectament l'eficàcia d'aquest mètode en aquest tipus de problemes.

Dos són els tipus o classes de codis en el que hem basat el nostre estudi. Per una part estudiarem la aplicació dels A.G. en trobar codis de pes constant i en segon lloc estudiarem l'aplicació en codis de cobertura.

Els dos casos plantegen problemes, com explicarem posteriorment, d'aplicació directe d'aquest mètode en els codis que volem trobar. Per aquesta raó haurem de modificar en certa part els algorismes originals, encara que conservant la filosofia original. Per altra banda també hem realitzat noves aportacions en els A.G. que milloren la seva eficiència.

En el primer capítol farem una descripció del projecte i dels objectius que volem assolir. Els capítols del 2 al 4 són una introducció dels conceptes que anirem tractant. Així en el capítol 2 introduirem els codis, en el 3 tractarem dels mètodes de recerca d'extrems i en el 4 presentarem els algorísmes genètics. En els capítols 5 i 6 estudiarem més a fons aquests algorismes. En el capítol 5 mostrem els fonaments matemàtics d'aquests algorismes i en el capítol 6 tractarem operants genètics més complexes.

En el capítol 7 parlarem de com hem aplicat el nostre algoritme en la recerca de bons codis i de les modificacions realitzades en alguns procediments bàsics de l'A.G. També tractarem les millores que hem aportat a aquesta técnica de recerca per augmentar la seva eficiència. Així parlarem no tan sols de nous tipus d'encreuament i mutació utilitzats sinó també d'un nou tipus d'escalat de la funció no existent fins ara. Per últim farem una valoració dels resultats trobats i dels mètodes emprats, en el capítol 8.

Tot projecte és un treball llarg i sentit. La persona que el realitza pateix i gaudeix en la seva realització però la gent que té al costat pateix més que gaudeix. És per això que vull agrair el suport que en tot moment m'ha donat el meu director de projecte Francesc Comellas. Ell ha aguantat, sempre amb el mateix bon posat, tant les bones idees que he pogut aportar al projecte, algunes, com les dolentes, unes quantes més. També vull agrair la paciència dels meus amics i companys que han suportat estoicament que en qualsevol lloc i en qualsevol moment jo els introduís en el no sempre fàcil camp dels A.G. Per últim vull agrair, com no, el suport de la meva família tant en el camp dels estudis com a la resta de la meva educació.

1 Objectius i Descripció del projecte.

El projecte es basa en l'aplicació d'un mètode de recerca d'extrems anomenat Algorisme Genètic en l'obtenció de dos tipus diferents de codis: els codis de pes constant i els codis de cobertura. Un mètode de recerca d'extrems és aquell que donada una funció, i un domini on la funció està definida, ens troba un extrem, això és un màxim o un mínim, segons desitgem.

Els Algorismes Genètics han estat aplicats amb èxit tant a problemes d'optimització teòrics com pràctics. Han estat aplicats a camps tant diversos com l'aviació (disseny òptim de les proporcions d'un avió i el perfil d'atac de l'ala), la química (distribució òptima de les presions dins d'una tuberia) o la matemàtica (solució del problema del viatjant ceg), sense oblidar el seu camp inicial d'aplicació: la biologia (modelització del procés d'evolució genètic de les espècies).

El nostre objectiu principal és estudiar la viabilitat i eficàcia dels Algorismes Genètics en la recerca de bons codis. Volem saber si és possible aplicar aquest algorisme en el nostre problema i una vegada aplicat valorarem si els resultats trobats són millors o pitjors que els dels altres mètodes existents.

Classifiquem a un codi com a bo si aquest s'apropa en quant a nombre de paraules a la seva fita teòrica. Els codis que considerarem estaran formats per un nombre finit de paraules. En el cas dels codis de pes constant cercarem codis amb el major nombre possible de paraules. En aquest codi les limitacions en quant a nombre de paraules les trobem en el extrem superior. Això succeeix perquè si augmentem el nombre de paraules, aquestes no compliran les propietats que desitgem pel nostre codi. El que voldrem trobar, en aquest cas, serà la fita superior del codi.

En el cas dels codis de cobertura ens trobem en l'extrem oposat. Ens interessa que el codi tingui les menys paraules possibles. Voldrem trobar, doncs, la fita inferior del codi.

Aquestes fites teòriques s'han intentat trobar de moltes formes diferents. Hi ha una línia de recerca que intenta trobar aquestes fites analitzant les propietats d'aquests codis. Generalment aquest sistema ens troba la fita però no necessàriament un codi efectiu que l'acompleixi.

L'altra línia de recerca, que és la que utilitzarem, tracta de trobar els codis directament mitjançant un mètode de recerca d'extrems. Evidentment si trobem el codi explicitament, tenim també la fita.

Un aspecte molt important pel bon funcionament de l'algorisme és la funció de cost. La funció de cost evalua la bonança del codi i de la seva elecció depèn el correcte funcionament del mètode de recerca. Com posteriorment explicarem, volem un mètode que només necessiti conèixer el valor puntual de la funció i cap altra informació auxiliar més, com pot ser la seva derivada. D'altra banda necessitem que el mètode sigui molt eficient ja que els espais de recerca que utilitzarem seran molt grans.

Dels mètodes existents actualment es coneixen un conjunt que poden acomplir els nostres requisits. D'aquests mètodes en diem mètodes probabilístics i es diuen així perquè utilitzen regles de transició entre estats no determinístiques. Podem dir que els mètodes probabilístics més importants són la Recuita Simulada i els Algorismes Genètics. Aquests mètodes formen part d'una branca del que en diem intel.ligència artificial. Són mètodes basats en l'observació i modelització de la natura. Per exemple, la Recuita Simulada és una modelació del refredament dels cristalls i la tendència d'aquests d'anar cap a estats de mínima energia. Els Algoritmes Genètics es basen en les mecanismes de la selecció natural i la genètica.

La Recuita Simulada ha estat aplicada amb èxit en la recerca de fites teòriques tant de codis de pes constant com de codis de cobertura. És per això que nosaltres utilitzarem aquests resultats com a referència de l'eficàcia del nostre mètode. Això comporta unes avantatges però també uns inconvenients. Per una part hem de tenir en compte que l'obtenció d'un codi amb aquests mètodes no és determinístic. Per tant només tenim certa probabilitat de trobar-lo. La probabilitat d'intentar trobar un codi determinat ja trobat anteriorment per altres autors és baixa.

El nostre projecte consistirà per una part en l'estudi de la viabilitat de l'aplicació d'aquest mètode, Algorisme Genètic, en aquest tipus de problemes. Una vegada aplicat farem una valoració dels resultats obtinguts amb el nostre algorisme mitjançant la comparació amb els resultats trobats per altres autors amb la Recuita Simulada.

L'aplicació de l'Algorisme Genètic en aquests problemes no només ha donat com a resultat l'obtenció de bons codis sinó que a més ens ha portat a millorar alguns aspectes del mètode de recerca. Aquestes millores no són d'aplicació exclusiva a aquest problema i poden ser utilitzades per altres investigadors d'A.G.

L'aplicació real d'aquest algorisme es realitzarà en el llenguatge de programació C en un entorn UNIX, i el programa es farà córrer en els ordinadors SUN del Departament de Matemàtica Aplicada i Telemètica de l'E.T.S.E.T.B.

2 Introducció als Codis

En els sistemes de comunicacions actuals, el disseny dels codis és d'una importància especial. Per poder representar eficientment informacions tals com els caràcters d'un text, informació d'àudio i de vídeo, o simplement dades informàtiques, s'utilitzen els codis de font (source code). El codi A.S.C.I.I. (American National Standard for Information Interchange), els codis de Huffman, els codis Morse i les diferents codificacions tant de veu com d'imatges utilitzades s'han d'incloure dins d'aquest grup. La informació original, una vegada codificada, és després enviada mitjançant un canal de transmissió, com pot ser una línia telefònica, un enllaç de microones o de fibra òptica. En aquests canals els senyals són corromputs amb soroll. Per assegurar una transmissió fiable, les dades han de ser codificades amb un codi de canal (codi corrector d'errors). Això permet la detecció dels errors i la seva eliminació. Els principals codis de canal utilitzats són els codis de Hamming, els codis de Reed-Solomon, els codis convolucionals i els codis de Trellis.

2.1 Nocions bàsiques de codis

El nostre estudi es basarà en cert tipus de codis correctors d'errors. És per això que deixarem de banda aprofondir en els codis de canal. Seguidament farem una introducció als codis correctors d'errors.

En les transmissions digitals els errors en la comunicació depenen de la relació entre el senyal i el soroll. En un sistema en particular i per a una certa relació senyal a soroll podem millorar la fiabilitat de la transmissió mitjançant una codificació de control d'errors.

El control d'errors implica automàticament un augment en el nombre de dígits a transmetre respecte del missatge original. Aquests dígits extra que no porten informació per si mateixos són els que permeten detectar o corregir errors en el missatge regenerat.

Introduirem el concepte de la correcció d'errors encara que la seva aplicació no serà el principal interès en el nostre projecte. De tota manera ens permetrà familiaritzar-nos en els conceptes bàsics dels codis.

Codificar per detectar errors, sense corregir-los, és més senzill que codificar per poder-los corregir. Quant existeix un canal de dos sentits de transmissió entre la font i el destí, el receptor pot requerir la retransmissió de la informació que ha contingut errors detectats. Aquesta estratègia de control d'errors s'anomena A.R.Q. (Automatic Repeat Request). En els casos en que això no és possible, el control d'errors que s'utilitza és l'anomenat F.E.C. (Forward Error Correction) que utilitza un codi corrector d'errors. La transmissió es realitza en un sol sentit i la regeneració del codi, quant un error és detectat, es realitza mitjançant la correcció d'aquest. Analitzem breument dos casos senzills de codis controladors d'errors: els codis de repetició i els codis de paritat.

2.1.1 Codis de repetició i de paritat

Quant tractem de parlar en un lloc sorollós, necessitem moltes vegades repetir les coses per que ens entenguin. Una aproximació grollera a la comunicació binària sobre un canal sorollós, utilitza d'alguna manera la repetició. D'aquesta forma cada bit del missatge es representa per una paraula formada per n bits idèntics. Qualsevol error en la transmissió farà canviar un bit de 1 a 0 o a l'inrevés.

Si els errors de transmissió tenen lloc aleatòriament i independentment amb probabilitat $P_e = \alpha$, llavors tenim que la probabilitat de tenir i errors en una paraula de n bits és:

$$P_e = \binom{n}{i} \alpha^i \cdot (1-\alpha)^{n-i} \approx \binom{n}{i} \alpha^i \qquad \alpha \ll 1$$

Els codis de repetició proporcionen fiabilitat quant α és suficientment petit per fer que P(i+1,n) << P(i,n), i per tant sigui molt menys probable tenir pocs errors per paraula, que molts per paraula.

Considerem, per exemple, un codi de tres repeticions amb paraules 000 i 111. La resta de paraules rebudes com per exemple, la 001 o la 101, indiquen clarament la presència d'errors. Depenent de l'esquema de decodificació aquest codi pot detectar o corregir paraules errònies. Per poder detectar errors sense corregir-los, diem que qualsevol paraula diferent de 000 o 111 és un error detectat. Els errors simples i els

dobles són detectats, però els errors triples són indetectables amb aquest tipus de codificació. Això té lloc amb una probabilitat:

$$P_{we} = P(3,3) = \alpha^3$$

Per corregir errors, utilitzem com a regla de decodificació que al menys dos o tres bits són correctes. Per tant 001 i 101 són decodificats com 000 i 111 respectivament. Aquesta regla corregeix errors simples però produeix errors en la decodificació amb probabilitat:

$$P_{we} = P(2,3) + P(3,3) = 3\alpha^2 - 2\alpha^3$$

Si la probabilitat d'error sense codificació és $P_e = \alpha$, veiem que aquest esquema de decodificació de triple repetició millora molt la fiabilitat de la transmissió per valors de α petits. De tota manera, aquesta millora té com a repercussió el reduir la velocitat de bit del missatge en un factor 1/3.

Existeixen altres codis més eficients que es basen en la noció de la paritat. Es diu que la paritat d'un nombre binari és parell quant la paraula conté un nombre d'uns parell, i es diu que és imparell quant té un nombre d'uns imparell. Les paraules per un codi de paritat es construeixen amb un missatge de n-1 bits i un bit de comprovació escollit perquè totes les paraules tinguin la mateixa paritat. Amb n=3 i paritat parell, les paraules valides són: 000, 011, 101, i 110. L'últim bit de cada paraula és l'anomenat bit de paritat.

Quant rebem una paraula amb paritat imparell, com 001, per exemple, sabem que hi ha hagut un nombre imparell d'errors. No és possible corregir l'error ja que no sabem on ha tingut lloc. A més si tenim un nombre parell d'errors no serà detectat.

Sota la condició de $\alpha << 1$, els errors dobles són molt més freqüents que els errors quàdruples o pitjors. Per tant la probabilitat de no detectar un error en una paraula d'un codi de paritat de n bits és:

$$P_{we} \approx P(2,n) \approx \frac{n(n-1)}{2}\alpha^2$$

Per propòsits comparatius, una transmissió de paraules de n-1 bits sense codificar tindria una probabilitat d'error:

$$P_{uwe} = 1 - P(0,n-1) \approx (n-1)\alpha$$

Per tant, si fem n=10 i $\alpha=10^{-3}$, llavors $P_{uwe} \approx 10^{-2}$ on, amb la codificació, arriba fins a $P_{uwe} \approx 5\times10^{-5}$ amb una reducció de la velocitat de bit de només 9/10.

2.1.2 Codis Vectors i distància de Hammming

En lloc de continuar amb una descripció detallada de codis en particular, introduirem una manera més general de tractar els codis en termes de codis vectors. Una paraula arbitrària de n bits pot ser visualitzada en un espai de n dimensions com un vector de coordenades igual als bits que composen la paraula. Podem representar el bit 101 en una notació vectorial com x=(101). Podem veure a la figura 2.1 una representació de totes les paraules possibles de 3 bits representats com a punts dins un espai tridimensional.

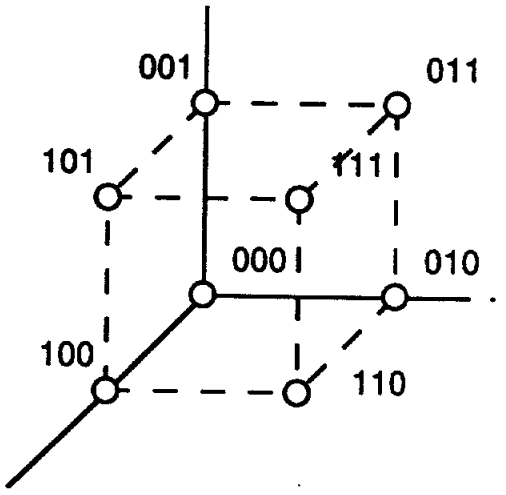
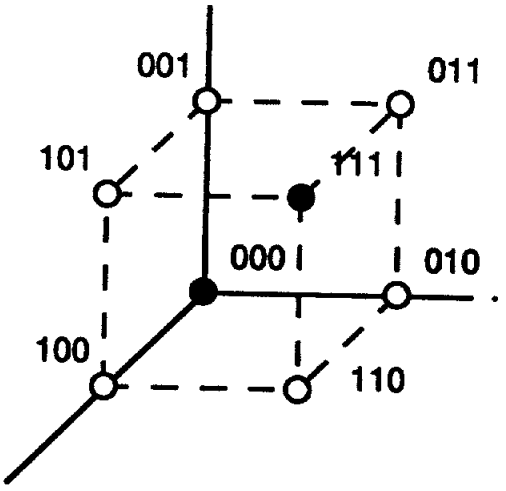


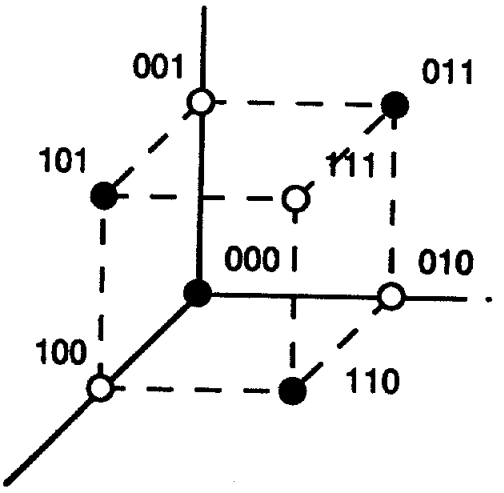
Figura 2.1

Podem representar igualment el codi de triple repetició i el codi de paritat de 3 bits.



Codi de triple repetició

Figura 2.2



Codi de paritat

Figura 2.3

Hem de fer notar que els vectors del codi de triple repetició tenen una separació entre les seves paraules major, que la separació existent entre les paraules que formen el codi de paritat. Aquesta separació, mesurada en termes de distància de Hamming, entre les paraules del codi té una relació directa amb la capacitat de control d'errors d'aquest codi.

La distància de Hamming d(x,y) entre dos vectors x i y es defineix com el nombre d'elements diferents que tenen aquests dos vectors. Per exemple si x=(101) i y=(110) llavors tenim que d(x,y)=2 perquè el segon i el tercer elements són diferents.

La distància mínima d_{min} d'un codi en particular és la distància de Hamming més petita entre els vectors-codis vàlids. Conseqüentment, la detecció d'errors és sempre possible quant el nombre d'errors de transmissió en una paraula és menor a d_{min} ja que la paraula errònia no serà un vector vàlid. També hem de dir que quant el nombre d'errors iguala o excedeix d_{min}, la paraula errònia pot correspondre a altre vector vàlid i l'error no podrà ser, en aquest cas, detectat.

Si raonem una mica més podem arribar als següents requeriments de distància mínima segons el grau de control d'errors que desitgem:

Detectar fins a L errors per paraula	$d_{min} \geq L+1$
Corregir fins a e errors per paraula	$d_{min} \geq 2e+1$
Corregir fins a e errors i detectar L>e errors	$d_{min} \geq e+L+1$

A la figura següent podem veure graficament el que acabem de dir. Si el nombre d'errors es menor a la meitat de la distància entre paraules del codi ho podrem corretgir sense cap equivocació possible.

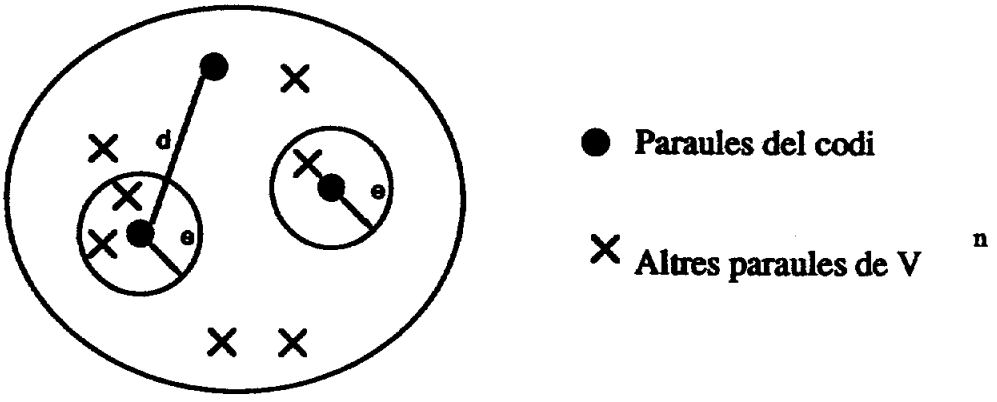


Figura 2.4

Podem veure com el codi de triple repetició té una distància mínima de 3. Per tant aquest codi pot ser utilitzat per detectar $L \leq 3-1=2$ errors per paraula o per corregir $t \leq (3-1)/2$ errors per paraula d'acord amb les anteriors observacions. Un codi més potent que tingués una $d_{min}=7$ podria corregir errors triples, o podria corregir errors dobles, i detectar errors quàdruples.

2.2 Codis de pes Constant

Podem incloure els codis de pes constant dins la categoria o classificació del codis de canal o correctors d'errors. Aquest codi es un conjunt de m paraules binàries de longitud n:

$$C=\{ x_1, x_2, , x_M \}$$

Amb una velocitat $R=\frac{1}{n}\log M$. Cada membre de $x \in C$ s'anomena paraula del codi. La distància mínima del codi C és:

$$d_{min}=\min_{x,y\in C;\ x\neq y} d_H(x,y)$$

On x i y són dos paraules del codi i d és la distància de Hamming entre dos paraules qualsevol, però diferents, del codi. Una propietat important d'aquest codi és que totes les paraules X_i de C tenen el mateix pes de Hammming. El pes de Hamming d'una paraula binària és el nombre d'uns de que està formada. Per exemple, la paraula binària 11001 té una longitud n=5 i un pes de Hamming de 3, ja que té 3 uns. Quant aquest codi de canal és utilitzat en un sistema de transmissió ens permet corregir i per tant eliminar fins a d_{min} errors de transmissió.

Un número important en la teoria dels codis de pes constant és A(n,d,w). Això és, la mida màxima d'un codi de pes constant format per paraules binàries de longitud n, distància mínima d, i pes constant de la paraula w.

Els valors A(n,d,w) són molt difícils de determinar. Per trobar aquestes constants han estat emprades sofisticades teories matemàtiques com, per exemple, la de Leech Lattice. Només unes quantes d'aquestes constants han pogut ser determinades exactament: A(24,8,8)=759 i A(24,8,12)=2576. Per altres valors només se n'ha pogut trobar un marge de possibles valors. Un dels valors més petits no determinats amb exactitud és: $17 \leq A(23,10,7) \leq 23$.

Aquest i altres valors han estat millorats primer per El Gamal *et al.* [1] i posteriorment per A. E. Brouwer *et al.* [4] mitjançant la utilització de tècniques combinatòries.

En aquest projecte arribem a trobar codis que igualen o superen els valors trobats per El Gamal *et al.* en el seu article. El Gamal *et al.* va utilitzar un mètode de recerca anomenat Recuita Simulada i nosaltres els trobem amb un altre anomenat Algorisme Genètic.

2.3 Codis de cobertura

Els codis de cobertura van ser introduits al 1948, quant Taussky i Todd [30] estudiaven teoremes de cobertura per grups. Després han aparegut moltes més contribucions a la recerca de codis de cobertura en diferents revistes dins de l'àrea de la codificació i de la combinatòria. El principal interès s'ha centrat en la recerca de codis binaris i ternaris de

radi de cobertura 1 (aquest problema també és anomenat com "The football pool problem"). Es pot dir que hi ha hagut un interès creixent en aquest àrea en els darrers anys.

El problema de trobar codis de cobertura pot ser formulat com un problema d'optimització combinatòria. El sistema de recerca anomenat Recuita Simulada ha demostrat funcionar molt bé en l'obtenció d'aquests codis. Nosaltres ho farem amb el nostre Algorisme Genètic.

Considerem l'espai F_q^n format per totes les paraules de longitud n i de coordenades pertenyents al conjunt {0 . . q-1}. Un codi $C \subseteq F_q^n$ cobreix F_q^n amb radi R, si cada paraula de F_q^n està dins de la distància de Hamming R d'alguna paraula de C. Definida la distancia de Hamming com en el cas anterior. Un codi q-ari, amb M paraules de longitud n i amb un radi de cobertura R es anomenat un codi (q,n,M)R.

A més es defineix:

$$K_q(n,R)=\min \{ M \mid \text{existeix un codi } (q,n,M)R \}$$

Per exemple si considerem l'espai següent : $F_2^3=\{000,001,010,011,100,101,110,111\}$ tenim que el codi $C=\{000,111\}$ és un codi (2,3,2)1, ja que les paraules dins d'una distància 1 de 000 són $C_1=\{000,001,010,100\}$ i de 111 són $C_2=\{111,110,101,111\}$ i $C_1 \cup C_2=F_2^3$. $K_2(3,1)=2$ ja que no existeix cap codi (2,3,1)1 com es pot veure fàcilment.

L'índex q de $K_q(n,R)$ es sol ometre si aquest es 2. A la literatura tenim que les funcions $\sigma(n,k) = K_k(n,1)$ i també $\sigma_n= K_3(n,1)$. El problema de determinar σ_n es anomenat normalment "The football pool problem".

Recentment molts investigadors han treballat en determinar els valors de $K_q(n,R)$, de tota manera, encara es coneixen pocs valors exactes d'aquestes constants. Els esforços s'han centrat principalment en trobar les fites superiors i inferiors d'aquests codis. Aquestes fites són trobades en molts casos trobant implícitament els codis (q,n,M)R. Per fer-ho s'utilitzen mètodes de recerca d'extrems com la Recuita Simulada o els Algorismes Genètics.

Aquestes fites poden ser trobades directament, sense cap més ajut, o bé es pot utilitzar algunes de les propietats conegudes dels codis per facilitar la recerca. D'aquí sorgeix les cobertures utilitzant matrius i la recerca de codis mixtes del tipus $F_4^a F_2^b$ que explicarem a continuació.

2.3.1 Cobertura utilitzant matrius

Aquest mètode va ser considerat per primer cop per Kamps i Van Lint [26] i després va ser generalitzat per Blokhuis i Lam [2]. Posteriorment va ser generalitzat per Van Lint Jr. [27] per permetre aconseguir radis de cobertura arbitraris.

Sigui $A = (I;M) = (a_1,\cdots,a_n)$ una matriu $r \times n$ on I és la matriu identitat $r \times r$ i M és la matriu $r \times (n - r)$ amb valors de F_q. Per $s \in F_q^r$ la R-cobertura de s utilitzant A, $S_{A,R}(s)$, es defineix com:

$$S_{A,R}(s) = \left\{ s + \sum_{j=1}^{n} \alpha_j a_j \mid \alpha_j \in F_q, \left|\{\alpha_j \neq 0\}\right| \leq R, 1 \leq j \leq n \right\}$$

Conseqüentment, si $A = I$ tenim una cobertura en el sentit tradicional. Un subconjunt S de F_q^r R-cubreix F_q^r utilitzant A si:

$$F_q^r = \bigcup_{s \in S} S_{A,R}(s)$$

Teorema 1: Si S R-cobreix F_q^r utilitzant una matriu $r \times n$ $A = (I;M)$, llavors $W = \{w \in F_q^n \mid Aw \in S\}$ cobreix F_q^n amb radi R. $|W| = |S| \cdot q^{n-r}$ es deu al fet que la matriu A té rang r.

Demostració: Agafem qualsevol $x \in F_q^n$. Llavors $Ax \in F_q^r$, per tant existeix s, $\alpha_1,\cdots,\alpha_R$ i $i_1,\cdots,i_R$ ja que $Ax = s + \sum_{j=1}^{R} \alpha_j a_{i_j}$, conseqüentment

$A(x - \sum_{j=1}^{R} \alpha_j c_{i_j}) = s \in S$ i $d(x,W) \leq R$. $|W| = |S| \cdot q^{n-r}$ és degut al fet que la matriu A té rang r.

El problema d'optimització consisteix ara en trobar les paraules de S i la part M de la matriu A. Les interconnexions entre les paraules i la matriu causa problemes i no és clar com realitzar la recerca d'aquests valors eficientment. Això serà discutit quant expliquem l'aplicació de l'A.G. per trobar aquests codis.

2.3.2 Cobertura mitjançant un codi mixt.

Considerem l'espai $F_4^1F_2^b$, on $F_4 = \{0,1,2,3\}$ i $F_2 = \{0,1\}$. La distància de Hamming d(x,y) entre dues paraules $x, y \in F_4^1F_2^b$ és el nombre de coordenades en que difereixen. El radi de cobertura d'un codi $C \in F_4^1F_2^b$ és R si cada paraula de $F_4^1F_2^b$ està dins de la distància de Hamming R de, al menys, una paraula de C, i R és l'enter més petit que compleix aquesta propietat.

Teorema 1: Sigui $C \in F_4^1F_2^b$ un codi de radi de cobertura R, compost de n paraules. Llavors existeix un codi binari $C' \subseteq F_2^{b+3}$ de radi de cobertura R que té 2n paraules. A més $K(b+3,R) \leq 2n$.

Demostració: Provarem que el codi C' obtingut de codificar la primera coordenada de C utilitzant les regles

$$0 \rightarrow 000 \text{ o } 111$$
$$1 \rightarrow 001 \text{ o } 110$$
$$2 \rightarrow 010 \text{ o } 101$$
$$3 \rightarrow 011 \text{ o } 100$$

de totes les formes possibles, compleix les condicions. Evidentment,

$C' \subseteq F_2^{b+3}$ i el nombre de paraules de C' és 2n. Queda per veure que C' té un radi de cobertura R. Agafem-ne arbitrariament $x = (x_1,x_2) \in F_2^{b+3}$, on $x_1 \in F_2^3$, $x_2 \in F_2^b$ .Sigui $y \in F_4^1$ la paraula que és codificada en x_1 d'acord amb l'esquema anterior. Ara sabem que existeix una paraula $c = (c_1,c_2) \in C \in F_4^1F_2^b$ ($c_1 \in F_4^1$, $c_2 \in F_2^b$) tal que $d(x',c) \leq R$, on $x' = (y,x_2)$. Siguin c' i c" les paraules de C' obtingudes de c tal com c_1 és codificat en c'_1 i $c"_1$. Si $c_1 = y$, llavors $c'_1 = x_1$ o $c"_1 = x_1$, i per tant $d(x,\{c',c"\}) = d(x_2,c_2) = d(x',c) \leq R$. Si $c_1 \neq y$, llavors $d(x_2,c_2) \leq R- 1$. Com tots els parells de l'esquema de codificació cobreixen F_2^3 amb radi de cobertura 1, $d(x_1,\{c'_1,c"_1\}) = 1$. Per tant en aquest cas, $d(x,\{c',c"\}) = d(x_1,\{c'_1,c"_1\}) + d(x_2,c_2) \leq 1 + (R - 1) = R$. Per tant el radi de cobertura és al menys R. De fet es pot veure fàcilment que és exàctament R.

Teorema 2: Sigui $C \in F_4^qF_2^b$ un codi de radi de cobertura R que té n paraules. Llavors $K(b+3q,R) \leq 2^qn$.

Demostració: Immediatament per la prova del teorema 1 i per les discussions prèvies.

3 Mètodes de recerca d'extrems

Un mètode de recerca d'extrems és aquell que donada una funció f(x), que en diem funció de cost, i un domini on aquesta està definida, ens troba l'extrem (un màxim o un mínim) dins del domini. El principal objectiu d'un mètode de recerca d'extrems és trobar un extrem absolut i no un extrem local. Un altre objectiu desitjable és que aquest sigui trobat en el menor temps possible. Això matemàticament vol dir que aquest sigui trobat efectuant el menor nombre d'avaluacions puntuals de la funció de cost.

Les aplicacions d'aquests mètodes són molt àmplies, ja que molts problemes, tant teòrics com pràctics poden, en últim terme, ser reduïts a la maximització o minimització d'una funció de cost, que en general dependrà de varis paràmetres.

Un concepte important per valorar els diferents mètodes de recerca d'extrems és la robustesa: La robustesa és el balanç entre l'eficiència i l'eficàcia necessària perquè un mètode pugui adaptar-se a diferents tipus de problemes. Això és, la capacitat d'un mètode de ser altament eficient sense dependre massa del tipus de problema al que és aplicat. Molts mètodes poden tenir una gran eficiència en cert tipus de problemes, però fracassar totalment en altres. Nosaltres cercarem per tant que el nostre mètode sigui robust en la nostre recerca.

La literatura actual identifica tres tipus principals de mètodes de recerca: mètodes basats en el càlcul, enumeratius i aleatoris. Examinem cadascun d'aquests mètodes per veure quines conclusions podem treure sense un test formal.

3.1 Mètodes de Càlcul

Els mètodes basats en el càlcul han estat estudiats bastament. Aquests es subdivideixen en dos classes principals: indirectes i directes. Els mètodes indirectes busquen un extrem local resolent el conjunt d'equacions, normalment no lineals, resultants d'igualar el gradient de la funció a zero. Això no és més que la generalització multidimensional de la noció de càlcul elemental de la recerca de punts extrems.

Donada una funció, per trobar un possible pic comencem restringint la recerca a aquells punts amb pendent zero en totes les direccions. Per altra part, els mètodes directes cerquen un extrem local de la funció, movent-se en la direcció indicada pel gradient en aquell punt. Encara que aquests mètodes han estat molt estudiats, millorats i reformats podem veure fàcilment la seva manca de robustesa.

Primer, els dos mètodes utilitzen una recerca local. Això vol dir que l'òptim que busquen és el millor en una proximitat del punt actual en el que es troben. Si utilitzem aquests mètode en l'exemple de la figura 3.1 podem intuir que el nostre mètode de càlcul funcionarà força bé però si la utilitzem en l'exemple de la figura 3.2 tenim moltes probabilitats de que el nostre mètode de càlcul quedi estancat en el màxim local i per tant no assoleixi l'objectiu principal de la nostre recerca. A més a més, una vegada assolit l'extrem local hem d'utilitzar altres tècniques com un recomençament de nou, aleatori, si volem aconseguir alguna millora.

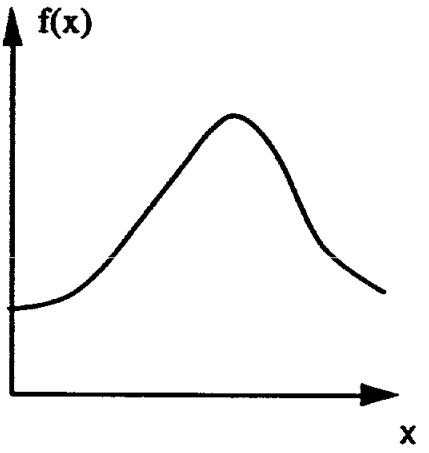


Figura 3.1

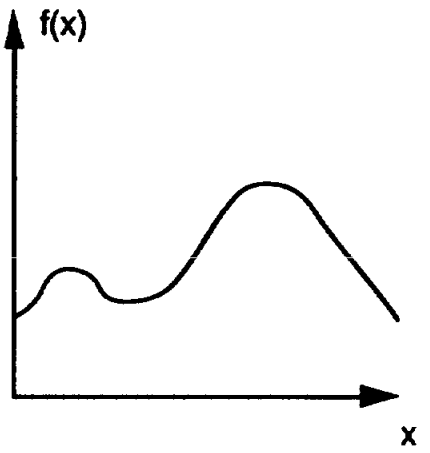


Figura 3.2

Segon, els mètodes basats en el càlcul depenen de l'existència de derivades. Inclús si permetem una aproximació numèrica de les derivades tenim severes restriccions en la seva utilització. A molts espais de paràmetres no es poden aplicar, a la pràctica, la noció de derivada i la bajanada que això ens pot aportar. El món real de recerca està ple de dicontinuitats, grans multimodalitats i espais de recerca amb soroll com es pot veure, per exemple, en la funció de la figura 3.3.

Figura 3.3

Per tant podem dir que els mètodes que depenen de l'existència de la continuitat i derivabilitat de la funció només poden ser utilitzats en un domini de problemes molt limitat. Per aquesta raó i pel fet de ser un mètode de recerca inherentment local hem de rebutjar els mètodes de recerca basats en el càlcul. No són suficientment robustos en dominis desconeguts.

3.2 Mètodes enumeratius

Els esquemes enumeratius han estat considerats de moltes maneres diferents. La idea general és molt senzilla: Dins d'un espai finit o d'un espai infinit discretitzat, l'algorisme de recerca comença a buscar solucions en cada punt de l'espai, un cada vegada. Encara que la simplicitat d'aquest algorisme és atractiva, i que l'enumeració és un tipus de recerca molt humana (quant el nombre de possibilitats són petites) aquests tipus d'esquemes no deuen ser tinguts en compte degut a la seva poca robustesa per una única raó: tenen molt poca eficiència.

La majoria de problemes pràctics tenen un espai de recerca suficientment gran com per no tenir possibilitats de trobar una bona solució en un temps raonable. Fins i tot esquemes enumeratius més complexes com l'esquema enumeratiu "dinàmic programming" falla en problemes de moderada mida i complexitat, sofrint del mal, melodramàticamant anomenat, "la cursa de la dimensionalitat" pel seu creador (Bellman [5]).

3.3 Mètodes Aleatoris

Els algorismes de recerca aleatòria han aconseguit una popularitat creixent un cop els investigadors han reconegut les limitacions dels mètodes basats en el càlcul i els esquemes enumeratius. De tota manera hem de descartar els esquemes simples de recerca aleatòria que només busquen punts a l'atzar i guarden el millor, ja que no podem esperar d'aquests més del que podíem esperar dels mètodes enumeratius.

Hem de tenir cura de distingir, el que són simples mètodes aleatoris dels que utilitzen tècniques aleatòries en la seva recerca. Els Algorismes Genètics són un exemple del tipus de mètodes que utilitzen tècniques aleatòries per fer una recerca altament eficient d'un espai de paràmetres del que depèn el valor de la funció. La utilització d'eines aleatòries en mètodes de recerca directes pot semblar estrany en un principi, però la natura està ple d'exemples que indiquen que això és vàlid i funciona.

Un altre mètode aleatori que utilitza eines aleatòries en la seva recerca es la Recuita Simulada. Aquest mètode creat com a modelització del sistema natural de la cristal.lització dels minerals, evoluciona utilitzant aquestes tècniques aleatòries cap a estats de mínima energia. Davis al 1987 [11] va publicar un llibre que explorava les connexions existents entre Simulated Annealing i els Algorismes Genètics. El més important que hem de tenir en compte és que una recerca aleatòria no ha d'implicar necessàriament una recerca a cegues.

Encara que això no ha estat un examen exhaustiu als milers de mètodes de recerca tradicionals existents, som capaços d'adonar-nos d'una conclusió important: Els mètodes de recerca tradicionals no són molt robustos. Això no implica que no siguin útils. Els esquemes que hem anomenat han estat utilitzats amb èxit en moltes aplicacions. Però quant s'han intentat atacar problemes més complexes s'ha vist la necessitat de crear nous mètodes.

Per centrar bé el problema, observem l'espectre de problemes de la figura 3.4. A la gràfica es mesura un índex d'efectivitat teòric per cada tipus d'esquema: Un esquema especialitzat que podria ser un mètode basat en el càlcul, un esquema enumeratiu i un esquema teòricament robust

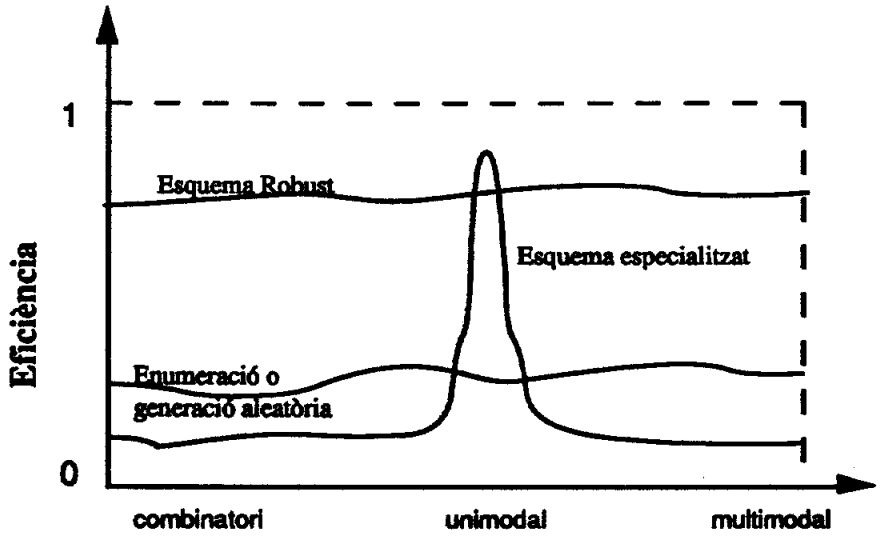


Figura 3.4

Les tècniques de gradient funcionen bé en una classe de problemes molt petita, però són molt ineficients en la resta. Per altra part els esquemes enumeratius funcionen amb una igualitària ineficiéncia sobre tot l'espectre de problemes com es pot veure per la seva baixa eficiència. global.

Seria molt més desitjable una característica com la que presenta l'esquema robust. Hi ha un sacrifici d'eficiència respecte als esquemes de gradient, en els problemes que aquests poden resoldre, però en canvi tenim un més ampli espectre de problemes amb un alt nivell d'eficiència. Ara veurem com els Algorismes Genètics aconsegueixen aquesta robustesa.

4 Algorismes Genètics

Els A.G. estan basats en els mecanismes de la selecció natural i de la genètica. Van ser tractats amb profunditat per John Holland al 1975 [24]. Holland, els seus col.legues, els seus estudiants i altres investigadors han treballat activament en aquest àrea des dels anys setanta.

No acceptem els mètodes dels algorismes genètics perquè els arguments de la bellesa de la natura ens semblin atractius, sinó perquè els algorismes han provat ser sistemes robustos, teòrica i pràcticament, en la recerca d'espais complexes. La primera monografia del tema és "Adaptation in natural and Artificial Systems." escrita per John Holland al 1975 [24]. Molts articles i dissertacions estableixen la validesa d'aquesta tècnica, en optimització de funcions i control de processos. Una vegada establerta la validesa del mètode, aquest ha trobat moltes aplicacions en camps com els negocis, la ciència i l'enginyeria.

La raó darrera d'aquest nombre creixent d'aplicacions hem de trobar-la en que aquests algorismes són computacionalment simples encara que són molt potents en la seva recerca. A més a més, no estan fonamentalment limitats per assumpcions restrictives de l'espai de recerca (continuitat, existència de derivades, unimodalitat, i altres temes).

En un principi els A.G. només eren utilitzats com a eines per estudiar i comprendre millor els mecanismes de la selecció natural i la genètica. Van començar com una modelació de la forma en que les espècies evolucionen. Posteriorment, veient les possibilitats que tenien, es van utilitzar com a algorisme de recerca d'extrems i va ser aplicat a resoldre altres problemes amb èxit.

La genètica ens diu que tots els éssers vius tenen codificada el que són, i el que poden arribar a ser, en els seus cromosomes. En els éssers humans aquests paràmetres determinen, per exemple, el color dels ulls, el potencial de creixement, la intel.ligència, etc. Aquests paràmetres venen determinats per seqüències de bases nitrogenades dins de l'A.D.N. Aquesta informació que caracteritza l'individu és modelada a l'algorisme genètic amb un "string" (un "string" és una seqüència o tira d'unitats d'informació). Aquest "string" és la parametrització del individu.

La genètica també ens indica com els individus es reprodueixen. Això és, com partint de dos individus es crea un nou individu. Aquest nou individu tindrà característiques semblants a la dels seus pares.

Els individus no evolucionen però les espècies sí. Això és degut a la selecció natural. La selecció natural és el mecanisme amb el qual els millors individus d'una població, els millor adaptats al medi, tindran més probabilitats de sobreviure i de tenir descendència. Als A.G. modelem aquesta adaptació al medi amb la funció de cost. Aquesta tindrà un valor per cada individu i ens indicarà la seva adaptació al medi. Quant més adaptat al medi estigui, millor funció de cost tindrà i per tant, i com veurem posteriorment, tindrà més possibilitats de tenir descendència.

Per modelar la natura com a sistema el que fem és utilitzar un conjunt finit d'individus, en diem població, que anirà evolucionant. En cada generació es crea una nova població d'individus artificials (representats pels "strings") utilitzant parts dels individus de la generació anterior. La nova població creada tindrà la mateixa mida que la generació anterior. Per poder trobar aquesta nova població el que fem es creuar un nombre de parelles igual a la meitat de la població total.

Cada parella d'individus, en diem pares, donarà un parell d'individus fills que pertanyeran a la nova generació. Així aconseguim que la mida de la població resti constant el que computacionalment és molt dessitjable. Per poder trobar bons individus, strings amb valors de la funció de cost alts, el que farem serà deixar que la nostre població evolucioni durant moltes generacions. Anirem creant nous individus generació rera generació amb l'esperança de que, com a la natura, aquests es vagin adaptant al medi per sobreviure i per tant vagin millorant la seva funció de cost.

La creació de nous individus no és determinista, però això no implica que els algorismes genètics siguin simples recerques aleatòries. Els Algorismes Genètics exploten eficientment la informació històrica per especular en nous punts de recerca, nous individus, que esperem estiguin millor adaptats al medi (i per tant tinguin, globalment, major funció de cost).

Els Algorismes genètics es diferencien de la resta de mètodes de recerca en quatre punts principals:

1. Els A.G. treballen amb una codificació del conjunt del paràmetres, no amb els mateixos paràmetres.

2. Treballen amb una població de punts en cada iteració, no amb només un.

3. Utilitzen la informació del valor puntual de la funció, i no necessiten conèixer les seves derivades o altra informació auxiliar.

4. Utilitzen regles de transició probabilístiques, no determinístiques.

Els A.G. requereixen que el conjunt de paràmetres del problema a optimitzar es codifiqui en un string de longitud finita d'un alfabet finit. Posem un exemple: la maximització de la funció de cost $f(x)=x^2$ en l'interval de 0 a 31. En una aplicació d'un A.G. el primer pas que hem de fer és codificar el paràmetre x com un string de longitud finita. N'hi ha moltes maneres de codificar aquest paràmetre. Una manera simple i directa és codificar x com a un número binari de 5 bits. Un possible individu d'aquesta població podria ser 01101. A cada un d'aquests individus li fem correspondre el valor de la funció en aquell punt.

Com a primera operació l'A.G. crea una població d'individus a l'atzar i li fa correspondre a cada individu la seva funció de cost. Partint d'aquesta població inicial el que fa és crear una altra de la mateixa mida utilitzant eficientment els valors de les funcions de cost dels individus. L'Algorisme Genètic progressarà fent evolucionar aquesta població generació rera generació.

En molts mètodes d'optimització ens movem d'un sol punt a un altra en l'espai de recerca utilitzant algun tipus de regla de transició per determinar quin serà el proper punt. Aquest mètode punt a punt és perillós, ja que és una prescripció perfecte per localitzar falsos pics en un espai de recerca multimodal (amb molts pics). Pel contrari, els Algorismes Genètics treballen amb una rica base de punts simultàniament (una població de strings) pujant molts pics en paral.lel. Per tant amb aquest mètode es redueix la probabilitat de trobar un fals pic.

Moltes tècniques de recerca requereixen pel seu funcionament informació auxiliar de la funció. Per exemple, les tècniques de gradient necessiten conèixer la derivada puntual de la funció per poder arribar a trobar un extrem. Altres mètodes de recerca locals com les tècniques de

greedy o l'optimització combinatòria requereixen la majoria, sinó tots, els paràmetres tabulats. En canvi, els Algorismes genètics no necessiten aquesta informació auxiliar: Els A.G. són cecs. Només necessiten per millorar conèixer el valor puntual de la funció associada amb els strings individuals. Aquest fet és el que determina que es pugui aplicar a un ampli nombre de problemes.

Els A.G. utilitzen regles de transició probabilístiques per trobar els seus resultats. Els nous individus de la població es creen amb certa probabilitat. Com en el cas real el fet de que dos individus es trobin és probabilístic, també ho és el que tinguin descendència, i també tenim el fet que pels mateixos pares no tindrem sempre els mateixos fills.

Pot semblar en un principi estrany però hem de tenir en compte que la utilització de regles probabilístiques per crear nous individus no implica que el mètode sigui un simple mètode aleatori. Els A.G. utilitzen les probabilitats com a eina per guiar la recerca cap a zones de l'espai de recerca on trobarem, amb més probabilitat, millors valors de la funció.

Aquestes quatre diferències: us directe de la codificació, recerca d'una població, no requerir informació auxiliar, i la utilització dels operadors aleatoris contribueixen a la robustesa dels A.G. i a l'avantatge respecte d'altres mètodes comunament utilitzats.

El nostre algorisme el que farà en resum, és crear una població d'individus a l'atzar, seleccionar un conjunt de parelles a creuar segons la seva funció de cost i creuar-les segons certes regles, per tal de donar dos nous fills. El que esperem d'aquesta nova població d'individus és que siguin millors que els seus pares. Matemàticament volem dir que les mitges de les seves funcions de cost siguin més altes.

A la següent secció introduirem un Algorisme genètic simple basat en l'us de tres operants aleatoris.

4.1 Algorisme Genètic bàsic

En aquest capítol estudiarem l'aplicació d'aquest mètode de recerca d'extrems a un problema molt senzill. Utilitzarem per resoldre'l un algorisme genètic bàsic de tres operadors, ja que com veurem posteriorment, poden haver vàries variacions respecte d'aquest.

Considerem el problema indicat anteriorment: L'optimització de la funció $f(x)=x^2$ on x pertany a l'interval de 0 a 31.

Per poder utilitzar un A.G. el primer que hem de fer és codificar els individus de la nostre població en un string de longitud finita. Per aquest problema, codificarem la variable x simplement com un enter binari sense signe de longitud 5. Amb un dígit binari de 5 bits podem obtenir números entre 0 (00000) i 31 (11111). Amb una funció objectiu ben definida i codificada nosaltres farem evolucionar una generació de l'algorisme genètic, amb els tres operants bàsics que són: reproducció, encreuament i mutació.

Per començar, generem una població inicial aleatòriament. La mida de la població (nombre d'individus del que està formada) és determinada prèviament per nosaltres. Es buscarà una mida de la població adequada al problema a resoldre. Aquesta població tindrà una mida, en aquest cas, de quatre individus. Per trobar una població inicial de 4 individus haurem de trobar a l'atzar vint valors binaris. Per poder-los trobar podríem llençar 20 vegades una moneda no esbiaixada i assignar un "zero" a la cara i un "u" a la creu. Això, és clar, a la realitat ho farem amb un ordinador i no amb una moneda. Suposem que trobem els següents valors pels individus de la nostre població inicial:

01101
11000
01000
10011

Una vegada creada la població a l'atzar el que fem és assignar-li a cadascun el valor de la funció de cost que li correspon.

Per calcular el valor de la funció de cost (l'adaptació de l'individu al medi, també anomenada "fitness") simplement passem a decimal el valor del string binari que representa a l'individu i elevem aquest valor al quadrat.

Per exemple, si l'individu té com a valor 01000 obtenim que x=8 i que f(x)=64. Hem de fer notar que l'algoritme genètic no necessita conèixer el valor de la funció f(x) com a tal. L'únic que necessita és conèixer el valor puntual d'aquesta funció als punts corresponents als strings codificats. Aquest és el concepte de caixa negra. Nosaltres només

coneixem quina es la sortida d'aquesta funció per una demanda concreta d'un valor puntual de la funció sense conèixer realment com ho troba.

Ara que ja coneixem la codificació de l'individu i com trobar la seva adaptació al medi li aplicarem el nostre algorisme genètic. Un algorisme genètic bàsic que aconsegueix bons resultats en molts problemes pràctics es composa de tres operadors:

1. Selecció
2. Encreuament
3. Mutació

4.1.1 Selecció

La Selecció és el procés on els strings individuals són seleccionats per reproduir-se depenent dels seus valors corresponents de la funció de cost. Els biòlegs diuen a aquesta funció la funció adaptació o "fitness" ja que el que realment determinarà si una codificació genètica s'ha adaptat bé al medi o no, serà en últim terme la seva capacitat per arribar a tenir descendència.

Matemàticament podem pensar que la funció f(x) és una mesura del profit, utilitat o bondat del que volem maximitzar. Seleccionar strings d'acord amb la seva adaptació vol dir que els strings amb valors més alts tindran més probabilitats de contribuir amb un o més descendents en la propera generació.

Aquest operador, per suposat, és una versió artificial de la selecció natural. La supervivència de Darwin amb criatures del tipus string. En les poblacions naturals l'adaptació ens ve determinada per l'habilitat dels individus de sobreviure als depredadors, a les malalties i als altres obstacles que li barren el pas cap a l'estat adult i cap a la reproducció.

Aquest operador matemàtic de selecció pot ser creat de formes molt diferents, però la més senzilla i també la més utilitzada és l'implementació algorísmica d'una ruleta no equiprobable de la sort.

Representem, dins la ruleta, a cada individu segons el valor relatiu de la seva funció de cost. Els individus amb major funció de cost tindran una porció de la ruleta més gran que els individus amb una funció de cost menor. Suposem que la nostre població inicial està composta per quatre individus i tenen els següents valors:

No	String	Adaptació f(x)	% del total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Suma		1170	100.0
Mitja		293	
Màxim		576	

Taula 4.1

Sumant els valors de la funció de cost obtenim un valor de 1170. El percentatge respecte del total de cada individu també està indicat a la taula 4.1. La ruleta corresponent es construirà fent correspondre a cada individu una proporció angular a la ruleta igual al seu valor relatiu de la funció de cost. Per aquest exemple tindríem la següent ruleta:

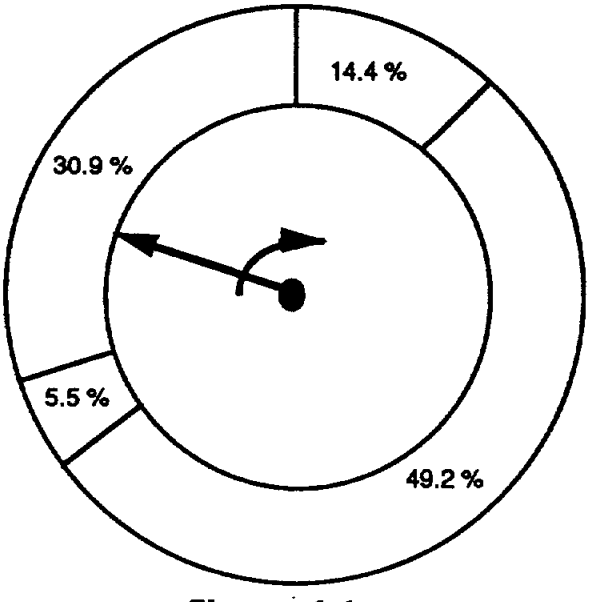


Figura 4.1

Podríem pensar en fer reproduir simplement als dos individus millor adaptats al medi, però això no és el que succeeix a la realitat. A la realitat un individu ben adaptat a l'entorn simplement tindrà més possibilitats de reproduir-se que un mal adaptat. Però tampoc és

determinístic, que l'individu millor adaptat tingui molta descendència i el mal adaptat cap.

Per reproduir aquest fet el que fem és implementar la ruleta tal com hem indicat anteriorment i seleccionar els individus a reproduir fent-la voltar. El lloc que assenyali la fletxa quant pari ens indicarà l'individu que serà seleccionat per reproduir-se. Els individus amb millor funció de cost tindran més possibilitats de reproduir-se ja que la seva porció angular a la ruleta serà més gran.

Hem de seleccionar els individus per reproduir-se en parelles. En el nostre cas, degut a que la mida de la nostre població es 4, hem de seleccionar dues parelles. Per fer-ho simplement fem girar quatre vegades la ruleta. Aquestes parelles ens donaran quatre fills que formaran la següent població després de passar pels restants operadors genètics.

La probabilitat de reproducció vindrà donada pel percentatge respecte del total que tingui el string. Per exemple, en aquest cas el que més possibilitats de reproduir-se té és el string 2 que té quasi el 50 per cent de probabilitats de reproduir-se en cada tirada. Després de quatre tirades haurem obtingut quatre strings que utilitzarem en el proper operador genètic.

4.1.2 Encreuament

Després de la selecció, els strings passen per l'operador de encreuament simple. A la reproducció sexual natural parts dels cromosomes provinents de l'individu pare s'uneixen amb parts dels de l'individu mare per donar un nou individu creat amb parts dels dos progenitors. Nosaltres simularem aquest mecanisme fent creuar els strings com si fossin cromosomes. Creuem els strings de dos en dos i obtenim dos nous strings que en direm fills, formats amb parts dels strings seleccionats de la generació anterior, que en diem pares.

Després de seleccionar dos individus pel seu encreuament procedirem de la següent manera: Seleccionem una posició per on tallar els strings (lloc d'encreuament). La posició d'encreuament es troba a l'atzar i variarà entre u i la longitud menys u del string,$(1 \leq K \leq L-1)$. Una vegada trobada aquesta posició K els creuem trencant els individus pares per aquesta posició, i intercanviem tots els caràcters entre les

posicions K+1 i la longitud màxima del string (L). Per exemple, considerem dos strings A_1 y A_2 de la nostre població inicial que volem creuar:

$$A_1 = 01101$$
$$A_2 = 11000$$

Primer trobem un número K a l'atzar entre 1 i 4. Obtenim K=4 (La posició de tall l'indicarem amb el símbol separador |). Per creuar-los deixem tots els bits sense canvi fins al separador "|" i intercanviem la resta a partir d'aquest. El resultat de creuar aconsegueix dos nous strings A_1' i A_2' on l'indicació " ' " indica que les strings formen part de la següent generació :

$A_1 = 0110\ |1$ $\qquad$ $A_1' = 0110\ |0$
$A_2 = 1100\ |0$ $\qquad$ $A_2' = 1100\ |1$

Els mecanismes de selecció i d'encreuament són sorprenentment simples, només impliquen generacions de números aleatoris, copies de strings i intercanvis de parts de strings. De tota manera la combinació de la selecció i l'intercanvi estructurat, encreuament, per mitjà de la randomització, dona a l'algorisme genètic gran part de la seva potència.

4.1.3 Mutació

L'últim operador que ens queda per conèixer és l'operador mutació. Les mutacions, a la natura, tenen lloc quant una part del cromosoma canvia. La majoria d'aquestes mutacions venen causades per radiacions ultraviolades provinents del sol. Poden causar des d'efectes pràcticament sense importància fins a canvis que poden provocar el no naixement de l'individu. Nosaltres també simularem aquest important efecte al nostre algorisme genètic. Aquest procediment consisteix en canviar un caracter del string amb certa probabilitat.

Quina és la funció d'aquest operant?. La funció d'aquest operant és d'incloure certa variació al nostre sistema. Això és necessari ja que les operacions de reproducció i d'encreuament només intercanvien i

manipulen informació ja coneguda. Per tant, si no existís aquest operant s'arribaria a un estat en el que no es crearia nova informació.

Com en l'operant anterior posarem un exemple de mutació. Suposem que volem mutar el string A_1 = 01101. Per mutar-lo hem de seleccionar en primer lloc quin bit canviarem. Per fer-ho trobarem a l'atzar un número "k" entre 1 i la longitud del string. En el nostre cas trobem un valor de 3. Per lo tant el que farem serà canviar el valor del bit 3. Obtindrem, doncs el següent string A_1'

$$A_1 = 01101 \qquad A_1' = 01001$$

L'últim aspecte en tenir en compte és que tant el mecanisme d'encreuament com el de mutació no es realitzen sempre. Només es realitzaran amb certa probabilitat. Al nostre algorisme tindrem definida una probabilitat d'encreuament i una de mutació. Per saber si realitzarem o no una mutació, o un encreuament trobarem a l'atzar un número entre 0 i 1. Si aquest número es més petit que la nostre probabilitat realitzarem l'operació i si és més gran no la farem.

Si no fem un encreuament el que farem serà copiar directament els strings pares sobre els fills. Intuïtivament el que ens diu és que sempre hi ha certa possibilitat que els pares sobrevisquin sense canvis a la següent generació. A la mutació succeeix igual. Quant no mutem el que fem és deixar íntegre el nostre string.

4.2 Exemple d'algorisme genètic bàsic

Apliquem ara el nostre A.G. al problema anterior, pas a pas. Com hem indicat anteriorment el primer que hem de fer és determinar la mida de la població i les probabilitats d'encreuament i de mutació. Aquestes variables es solen trobar experimentalment fent probes amb diferents valors i escollint-ne els millors. En el nostre cas la mida de la població és de quatre individus, la probabilitat d'encreuament és 0.8 i la de mutació és 0.01.

El següent que fem és generar la població inicial i li assignem la funció de cost que li correspon a cada individu. Per fer-ho transformem el nostre individu binari al seu valor decimal i una vegada obtingut el valor de x corresponent a l'individu li apliquem la funció de cost $f(x)=x^2$.

Per exemple si tenim l'individu 01000 obtenim un valor de x=8 al passar el seu valor a decimal el que correspon a un valor de f(x)=64.

Una generació de l'algorisme genètic comença amb la selecció. Seleccionem els individus a creuar fent girar la ruleta no equiprobable quatre vegades. La simulació actual ens dona com a resultat que el string 1 i el 4 tindran 1 copia cadascun en el grup que es creuarà. El string 2 rebrà 2 copies i el string 3 no en tindrà cap. Comparant això amb el nombre de copies esperades ($n \cdot P_{selec\_i}$) hem obtingut el que esperàvem. El millor obté més copies, els del mig una i el pitjor mor.

Ara tenim un grup de strings que esperen ser creuats. Per fer-ho el que hem de fer en primer lloc es decidir les parelles. Això ho podem fer de diferents maneres. Ho podem fer a posteriori d'haver-les elegit per reproduir-se, o bé podem decidir-ho mentre les anem seleccionant per creuar. Una vegada elegides les parelles a creuar, hem de decidir si les volem creuar, o no, mitjançant la probabilitat d'encreuament.

Una vegada ho hem decidit hem de trobar, a l'atzar, el lloc per on tallar als individus per poder-los creuar. Les parelles que han sorgit a la nostre simulació són la 2 i la 1 i es creuaran pel lloc 4. Els dos strings 01101 i 11000 es creuaran, i donaran dos nous strings 01100 i 11001. Els dos strings que queden en el grup a creuar es creuen pel lloc 2. Els strings que en resulten es poden veure a la taula 4.2.

L'última operació a realitzar, la mutació, es realitza amb una base, en aquest cas, de bit a bit. Assumim que la probabilitat de mutació de bit és en aquest exemple de 0.001. Amb 20 bits transferits podem esperar que mutin $20 \cdot 0.001=0.02$ bits en cada generació. Com a resultant d'això tenim que en el nostre cas no hi ha hagut cap mutació en la primera generació. Cap individu resultant de l'encreuament dels individus seleccionats ha canviat.

Després de la selecció, encreuament i mutació podem analitzar a la nova població. Per fer-ho només hem de trobar el valor de la funció de cost de la nova generació creada. Els resultats queden reflectits en la taula 4.1. Encara que pot resultar arriscat analitzar els resultats després d'una única generació podem començar a veure com els algorismes genètics combinen nocions d'altes prestacions per aconseguir bons resultats. A la taula podem veure que tant el màxim com la mitja de valors de la funció de cost han millorat en la nova generació. La mitja de

la població ha millorat de 293 a 439. El valor màxim s'ha incrementat de 576 a 729 en el mateix període.

No	Strings seleccionats	Parella	Lloc K de Encreuament	Nova valor generació	f(x) de x	x^2
1	01101|1	2	4	01100	12	144
2	11001|0	1	4	11001	25	625
3	11|000	4	2	11011	27	729
4	10|011	3	2	10000	16	256
Suma						1754
Mitja						439
Màxim						729

Taula 4.2

Encara que el procés aleatori ha causat aquestes felices circumstàncies, començarem veient que aquestes millores no han estat casuals. El millor string de la primera generació (11000) rep dues còpies a causa de que té un valor de la funció de cost per sobre de la mitja. Quant es combina amb el string més proper al màxim (10001) i es creua pel lloc 2 (trobat a l'atzar) un dels strings resultant (11011) demostra ser una molt bona elecció.

Aquest succés és una bona il.lustració de les idees i nocions que es demostraran posteriorment. En aquest cas la bona idea trobada és la combinació de dos conceptes amb uns valors corresponents per sobre de la mitja. Anotem aquestes bones idees amb el substrings 11--- i el --111. La combinació d'aquestes dues bones idees dona un bon resultat.

Encara que aquest argument és una mica heurístic comencem a veure com l'Algorisme genètic efectua una bona recerca. En la propera secció analitzarem el problema en termes d'esquemes de patrons.

5 Fonaments Matemàtics dels A.G.

No podem continuar sense respondre a una pregunta fonamental: en un procés de recerca donat només valors puntuals de la funció, quina informació és continguda en una població de strings i els seus corresponents valors de la funció de cost, que ens pot ajudar a guiar-nos millor a una recerca directa?.

Per respondre a aquesta pregunta més clarament, considerem els strings i els seus valors corresponents indicats a la taula 5.1. Quina informació és continguda en aquesta taula?.

Individu	Funció f(x)
01101	169
11000	576
01000	64
10011	361

Taula 5.1

A primera vista no hi ha molt: quatre mostres independents de diferents strings amb el seu valor de la funció de cost. Podem començar a mirar-nos'els de dalt a baix per veure algun patró comú. De seguida veiem, per exemple, que els strings que comencen per 1 tenen un valor més alt de la funció de cost. Pot ser això important per optimitzar la funció?. Certament amb la nostre funció $f(x)=x^2$ sabem que ho és. Però que és el que realment fem?. Primer, busquem semblances entre els strings de la població. Segon busquem relacions casuals entre aquestes semblances i els valors de la funció de cost. Al fer això admetem una fortalesa en l'informació per ajudar-nos en la nostre recerca. Per veure quanta i quina informació admetem hem de considerar l'important concepte dels esquemes i els patrons de semblances.

5.1 Patrons de Semblances (Esquemes)

En cert sentit no estem interessats en els strings com a strings únics. Ja que les semblances entre els millors pot ajudar-nos en la nostra

recerca ens preguntem com un string pot ser similar al seu company.

Específicament, ens preguntem en quin sentit és un string representatiu a altra classe de strings amb semblances en certes posicions al string. L'entorn dels esquemes ens dona l'eina necessària per respondre a aquesta pregunta.

Un esquema (schema) (Holland [22] i [24]) és un patró de semblances que descriu un subconjunt de strings amb semblances a certes posicions de l'string. Per aquesta discussió limitarem, sense pèrdua de generalitat, la nostre representació a l'alfabet binari {0,1}. Afegim al nostre alfabet un nou símbol especial, el "*" o jòquer. Amb aquest nou alfabet creem strings (esquemes) amb un alfabet ternari {0,1,*}. El significat de l'esquema és bastant clar si pensem en ell com un sistema d'adaptació de patrons: un esquema s'adapta a un string particular si a cada localització a l'esquema un 1 coincideix amb un 1, un 0 amb un 0 i un * amb qualsevol d'aquests valors. Com en l'exemple anterior considerem strings i esquemes de longitud 5. L'esquema *0000 adapta dos strings, concretament el subconjunt {10000, 00000}. Posem un altra exemple, l'esquema *111* descriu un subconjunt amb quatre components {01110, 01111, 11110, 11111}. Hem d'indicar que el símbol * és només un metasímbol utilitzat als nostres càlculs teòrics, però que no és emprat per l'A.G.

Per alfabets de cardinalitat (nombre de caracters de l'alfabet) K i longitud L tenim $(k+1)^L$ esquemes. A primera vista pot semblar que els esquemes fan la recerca més difícil ja que per un alfabet de cardinalitat k només (només?) tenim K^L strings. De fet, això ho fem per facilitar el raonament.

Reconeixem que si considerem els strings per separat només tenim quatre peces d'informació (amb una població d'aquesta mida). Però si considerem els strings, els seus valors de la funció de cost, i les semblances entre tots els strings de la població tenim un creixement en l'informació que ens ajuda en la recerca. Quanta informació admetem considerant les semblances?. Això està relacionat amb el nombre d'esquemes únics continguts a la població. Per trobar el nombre exacte necessitem conèixer els strings d'una població en particular. Per aconseguir una fita del nombre d'esquemes contem el nombre

d'esquemes continguts en un string individual i després donem un límit inferior del nombre d'esquemes a la població.

Per veure això considerem un string de longitud 5: 11111, per exemple. Aquest string pertany a 2^5 esquemes possibles perquè cada posició pot tenir el seu valor actual o bé un jòquer. Com a resultant, una població de n individus conté entre 2^L i $n \cdot 2^L$ esquemes depenent de la diversitat de la població. Podem veure com inclús una població de petita mida aporta gran quantitat d'informació. Ara veurem com l'A.G. explota eficientment aquesta informació.

On es porta tot això?. Més exactament, dels 2^L a $n \cdot 2^L$ esquemes continguts a una població, quants són processats d'una manera útil per l'A.G.?. Per obtenir la resposta a aquesta qüestió, considerem l'efecte de la selecció, l'encreuament i la mutació en l'augment o la disminució del nombre d'esquemes importants d'una generació a un altra. L'efecte de la selecció és fàcil de veure; ja que els strings amb millors valors tenen més probabilitats de ser seleccionats, en mitja tindrem un creixement en el nombre d'individus amb patrons observats com a millors.

De tota manera la selecció no troba nous punt a l'espai de recerca. Que li passa a un esquema quant l'introduïm a l'encreuament?. L'encreuament deixa a un esquema sense canvis si no el talla, però el pot trencar si ho fa. Per exemple considerem els esquemes 1***0 i **11*. El primer té més possibilitats de ser trencat que el segon. Com a resultat d'això veiem que esquemes de curta longitud tenen més probabilitats de ser reproduits.

La mutació, amb una probabilitat de mutació baixa, no sol trencar molts esquemes. Per tant tenim que els esquemes d'alts valors de funció de cost i de poca longitud (els anomenarem "blocs de construcció") es propaguen generació rera generació donant un increment exponencial d'elements. Tot això es fa en paral.lel sense cap ocupació física de memòria més que els n strings de la població.

En el proper capítol veurem quants d'aquets esquemes són processats útilment. Veurem que són de l'ordre de n^3. A causa de que aquest nivell de processament és tant important (i aparentment únic en els A.G.), li donem un nom especial "paralel.isme implícit".

5.2 Qui ha de viure i qui ha de morir: Teorema fonamental

Considerem que els strings, sense pèrdua de generalitat, estan constituïts per un alfabet binari V={0,1}. Per conveniències de notació, ens referirem als strings amb lletres majúscules i als caràcters individuals per lletres minúscules subscrites per la seva posició. Per exemple el string de 7 posicions A=0111000 es representarà simbòlicament com:

$$A=a_1a_2a_3a_4a_5a_6a_7$$

Aquí cadascun dels a_i representen una única característica binària o detector (d'acord amb les analogies naturals, a vegades s'anomena gen). Cada característica pot prendre un valor de 1 o de 0. (a vegades s'anomena als valors que pot prendre a_i al.lels). En aquest string en particular 0111000 a_1 és 0, a_2 és 1, etc. És també possible tenir strings on els detectors no estiguin ordenats seqüencialment com en el string A. Per exemple el string A' podria tenir el següent ordre:

$$A'=a_2a_6a_3a_1a_7a_4a_5$$

Però per ara assumirem que un detector pot ser determinat per la seva posició.

Els A.G. necessiten una població de strings, i nosaltres considerarem una població de strings individuals A_j, j=1,2, ,n. continguts a la població A(t) en el temps (o generació) t, on el tipus de lletra **negreta** indica població.

Ara necessitem una notació per descriure els esquemes continguts als strings individuals i a les poblacions. Considerem l'esquema H compost per l'alfabet de tres elements V^+={0,1,*}. Com hem indicat anteriorment el símbol * és un jòquer que pot significar tant un 1 com un 0 en una posició particular. Per exemple, considerem l'esquema de longitud 7, H=*11*0*. Fixem-nos com el string A=0111000 indicat anteriorment és un exemple de l'esquema H perquè els al.lels del string

a_i coincideixen amb les posicions de l'esquema h_i a les posicions fixes 2, 3 i 5.

Dels resultats del capítol anterior veiem que tenim 3^L esquemes de similitut diferents sobre un string binari de longitud L. En general per alfabets de cardinalitat k, tenim $(k+1)^L$ esquemes. A més recordem que en una població de n strings tenim com a màxim $n \cdot 2^L$ esquemes perquè cada string és per si mateix una representació de 2^L esquemes. Això ens dona una idea de la quantitat d'informació processada per l'A.G. De tota manera, per realment entendre els importants "blocs de construcció" de futures solucions, necessitem diferenciar els diferents tipus d'esquemes existents.

Tots els esquemes no es creen igual. Alguns són més específics que altres. Per exemple, l'esquema 011*1** està molt més definit sobre semblances importants que l'esquema 0******. A més, certs esquemes s'expandeixen més sobre la longitud del string que altres. Per exemple l'esquema 1*****1 s'expandeix més que l'esquema 1******. Per quantificar aquestes idees introduirem dos propietats dels esquemes importants: l'ordre de l'esquema i la longitud definida.

L'ordre de l'esquema H, ho denotem com o(H), és simplement el nombre de posicions fixes (en un alfabet binari el nombre de uns i de zeros) presents a l'esquema. A l'esquema anterior, l'ordre de l'esquema 011*1** és 4. (ho indiquem com o(011*1**)=4), on l'ordre de l'esquema 0****** és 1.

La longitud definida de l'esquema H, denotada per $\delta(H)$, és la distància entre la primera i l'última posició del string especificat. Per exemple, l'esquema 011*1** té una longitud definida $\delta=4$ perquè l'última posició especificada és la 5 i la primera posició especificada és la 1, i la distància entre elles és $\delta(H)=5-1=4$. En l'altre exemple (l'esquema 0******), la longitud definida és particularment fàcil de calcular. Ja que només hi ha una posició especificada, la primera i l'última posició especificada són la mateixa. I la longitud definida és per tant $\delta=0$.

Els esquemes i les seves propietats són interessant sistemes per discutir rigorosament i classificar semblances d'strings. Més que això, ens aporten els mitjans bàsics per analitzar l'efecte real de la reproducció i d'altres operadors genètics en els blocs de construcció continguts a la població. Considerem, doncs, l'efecte individual i

combinat de la selecció, encreuament i mutació en els esquemes continguts en una població de strings.

L'efecte de la selecció en el nombre esperat d'esquemes en la població és particularment fàcil de determinar. Suposem que en un moment donat i tenint m exemples d'un esquema particular H contingut a la població A(t), on nosaltres escrivim m=m(H,t) (tenim possiblement diferents quantitats d'esquemes H a diferents temps t).

Durant la selecció, un string és copiat d'acord amb la seva funció de cost, o més precisament un string A_i és seleccionat amb una probabilitat:

$$P_{sel\_i}= \frac{f(x_i)}{\sum_{j=1}^{n} f(x_j)}$$

Després de la selecció, nosaltres esperem tenir m(H,t+1) representants de l'esquema H en la població en un temps t+1 donats per l'equació:

$$m(H,t+1) = m(H,t)\cdot n\cdot\frac{f(H)}{\sum_{j=1}^{n} f(x_j)}$$

on f(H) és la mitja de la funció de cost dels strings representats a l'esquema H en el temps t. Si tenim en compte que la mitja de la població sencera pot ser escrita com $\bar{f} = \sum f(x_j)/ n$ llavors podem reescriure el creixement de l'esquema a la selecció com:

$$m(H,t+1)=m(H,t)\cdot\frac{f(H)}{\bar{f}}$$

En paraules, un esquema en particular creix amb la raó entre la mitja de la funció de cost de l'esquema a la mitja de la funció de cost de la població.

Dit d'un altra forma, els esquemes amb un valor de la funció de cost per sobre de la mitja de la població incrementaran el seu nombre de

representants de la següent generació, i els esquemes amb un valor per sota de la mitja tindran un nombre més baix de membres. Tots els esquemes d'una població augmenten o disminueixen el seu nombre de representants segons la mitja de l'esquema, només a l'operació de selecció. Podem veure alguna cosa més de la forma matemàtica d'aquest augment (o disminució) de l'equació dels esquemes?. Suposem que assumim que un esquema en particular H es manté per sobre de la mitja un percentatge $c\cdot\bar{f}$ sent c una constant. Sota aquesta assumpció podem reescriure l'equació de la següent manera:

$$m(H,t+1) = m(H,t)\cdot\frac{(\bar{f}+c\cdot\bar{f})}{\bar{f}} = m(H,t)\cdot(1+c)$$

Començant a t=0 i considerant estacionari el valor de c obtenim:

$$m(H,t) = m(H,0)\cdot(1+c)^t$$

Podem reconèixer aquesta equació com una progressió geomètrica o l'analogia discreta d'una exponencial. L'efecte de la selecció està quantitativament clar: la selecció permet un creixement (o decreixement) exponencial segons si està per sobre (o per sota) de la mitja de l'esquema. Connectarem aquesta conclusió amb el problema de la maquina escurabutxaques de K braços, però ara examinarem l'efecte de l'encreuament i de la mutació.

La selecció no fa res per explorar noves regions en l'espai de recerca, ja que no troba nous punts. L'encreuament és un intercanvi d'informació estructurada però aleatoritzada entre strings. L'encreuament crea noves estructures amb un mínim de disrupció a l'estratègia de col.locació dictada per la selecció.

Per veure quins dels esquemes es veuen afectats per l'encreuament i quins no, considerem un string en particular de longitud L=7 i dos esquemes representatius d'aquest string.

$$A = 0111000$$
$$H_1 = *1****0$$
$$H_2 = ****10*$$

Recordem que l'encreuament procedeix amb la selecció a l'atzar d'una parella, l'elecció aleatòria d'un punt d'encreuament, i l'intercanvi dels substrings des del començament del string fins al punt d'encreuament inclós, amb el corresponent substring de la parella escollida.

Suposem que volem creuar el string A amb un altre i trobem com a punt de tall el lloc 3, amb això volem dir que el punt de tall es farà entre les posicions 3 i 4. L'efecte del tall es pot veure fàcilment en els nostres esquemes H_1 i H_2 , on el punt de tall està marcat amb el signe | :

A= 011|1000
H_1=*1*|***0
H_2=***|*10*

Al menys que el string A' sigui idèntic a les posicions fixes de l'esquema (possibilitat que desestimarem), l'esquema H_1 serà destruït ja que la posició 2 de l'esquema H_1 serà reemplaçat amb un altra valor.

Pel mateix tall tenim en canvi que l'esquema H_2 té més possibilitats de no ser tallat que l'esquema H_1. Per quantificar aquesta observació, fixem-nos que l'esquema H_1 té una longitud definida de 5. Si el punt de tall és elegit entre L-1=7-1=6 possibles llocs, llavors tenim que l'esquema H_1 es destrueix amb una probabilitat $P_d=\frac{\delta(H_1)}{L-1}=\frac{5}{6}$ (i sobreviu amb una probabilitat $P_s = 1-P_d = \frac{1}{6}$). Igualment l'esquema H_2, té una longitud definida $\delta(H_2)=1$ i només pot ser destruit en un cas, quant la posició de tall cau entre els llocs 4 i 5. Per tant $P_d = \frac{\delta(H_2)}{L-1}=\frac{1}{6}$ i la probabilitat de supervivència $P_s = 1-P_d = \frac{5}{6}$.

Més generalment, podem veure que es pot calcular un límit inferior de la probabilitat d'encreuament P_s per cada esquema. Ja que un esquema sobreviu quant el punt de tall d'encreuament cau fora de la seva longitud definida, la probabilitat de sobreviure per un tall simple

és: $P_s = 1 - \frac{\delta(H)}{L-1}$, ja que l'esquema és molt probable que es trenqui quant el punt de tall cau dins dels valors fixes definits a l'esquema. Si l'encreuament és realitzat també amb una probabilitat P_c en un encreuament particular, hem de donar la següent expressió per la probabilitat de sobreviure:

$$P_s \geq 1 - P_c \cdot \frac{\delta(H)}{L-1}$$

que es redueix a l'anterior expressió per $P_c=1.0$.

Podem considerar també l'efecte combinat de l'encreuament i de la selecció. Com quant consideràvem la selecció únicament, estem interessats en calcular el nombre d'esquemes particulars H esperats en la propera generació. Considerant independència entre les operacions d'encreuament i de selecció obtenim la següent estimació:

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \left[1 - P_c \cdot \frac{\delta(H)}{L-1}\right]$$

Una vegada més el sentit de l'operació és clara. L'esquema H creix o decreix depenent d'un factor multiplicatiu. Amb l'encreuament i la selecció tenim que aquest factor depèn de dues coses: si l'esquema està per sobre o per sota de la mitja de la població i si la longitud definida de l'esquema és relativament curta o llarga. Clarament, podem veure que els esquemes amb valors per sobre de la mitja i longituds definides curtes seran representats amb una taxa d'increments exponencial.

L'últim operador a considerar és la mutació. Utilitzant la definició anterior, la mutació és l'alteració a l'atzar d'una única posició amb probabilitat P_m. Perquè un esquema H sobrevisqui han de sobreviure totes les seves possicions. Per lo tant, si un únic al.lel sobreviu amb probabilitat $(1-P_m)$ i cada una de les mutacions són estadísticament independents, un esquema en particular sobreviu quant cada una de les o(H) posicions fixes dins de l'esquema sobreviu. Multiplicant la probabilitat de sobreviure o(H) vegades obtenim la probabilitat de sobreviure a la mutació $(1-P_m)^{o(H)}$. Per valors petits de P_m ($P_m << 1$), la

probabilitat de l'esquema de sobreviure es pot aproximar per $1-o(H)\cdot P_m$.

Podem, per tant, concloure que un esquema H rep un nombre de còpies estimades en la següent generació després de la selecció, l'encreuament i la mutació donat per la següent equació (ignorant els termes creuats més petits):

$$m(H,t+1) \geq m(H,t)\cdot\frac{f(H)}{\bar{f}}\left[1-P_c\cdot\frac{\delta(H)}{L-1}-o(H)\cdot P_m\right]$$

Afegint la mutació podem canviar una mica la nostre conclusió prèvia: Els esquemes curts, de baix ordre i per sobre de la mitja reben un nombre de descendent exponencials en les següents generacions. Aquesta conclusió és important i per això li donem un nom especial: el teorema de l'esquema o teorema fonamental dels A.G.

Després estimarem quin és el nombre d'esquemes que és processat d'aquesta forma, però abans d'això hem de respondre a una pregunta important. Perquè hem de tenir un increment exponencial en el nombre de representants dels esquemes que són millors?. En altres paraules, és aquesta estratègia, una bona estratègia?. La resposta podem trobar-la on al principi pot semblar ser un lloc estrany, seguint a un jugador entre les màquines escurabutxaques a Las Vegas.

5.3 El problema de la màquina escurabutxaques de k braços

Perquè hem de tenir un increment exponencial en el nombre de representants dels esquemes observats com a bons blocs de construcció de solucions?. Aquesta pregunta enllaça amb un problema important de teoria estadística de decissió. Encara que això pot semblar una desviació del nostre problema principal, veurem de seguida com la solució òptima al problema de la màquina escurabutxaques és molt similar en forma a l'assignació d'un nombre de proves exponencial pel nostre A.G.

Suposem que tenim una màquina escurabutxaques de dos braços que anomenarem esquerre i dret. A més, coneixem que un dels braços dona un premi μ_1 amb una variança σ_1^2 i l'altra paga μ_2 amb una variança de σ_2^2 on $\mu_1 \geq \mu_2$. Amb quin braç hem de jugar?. Clarament hem de jugar

amb el braç que paga més freqüentment (el braç que paga μ_1). Però aquí està la gràcia. Ja que no sabem amb antelació quin braç està associat amb més grans guanys, ens trobem amb un dilema interesant. No només hem de prendre la decissió d'amb quin braç jugar, sinó que al mateix temps hem de recollir informació sobre quin és el millor braç. Aquest equilibri entre l'exploració del coneixement i l'explotació d'aquest coneixement és un tema recurrent, i fonamental en la teoria d'adaptació de sistemes.

Una manera d'aproximar-nos a aquest dilema és separar l'exploració de l'explotació, realitzant primer un únic experiment i després prenent una única decissió irreversible que dependrà del resultat de l'experiment. Això és una aproximació de la teoria de decisions tradicional que descriurem rigorosament.

Suposem que tenim un total de N tirades per distribuir entre els dos braços. Primer distribuìm un nombre igual de tirades n ($2n<N$) a cadascun dels dos braços durant la fase experimental. Després de l'experiment, distribuïm les N-2n tirades restants al braç que hem observat que paga més. Assumint que coneixem N, μ_1, μ_2 , σ_1^2 ,i σ_2^2 .Podem calcular les pèrdues esperades.

$$L(N,n)=(\mu_1-\mu_2)\cdot\left[(N-n)\cdot q(n) + n(1-q(n))\right]$$

On q(n) és la probabilitat que el pitjor braç observat sigui el millor després de n proves. Aquesta probabilitat és ben aproximada per la aproximació de la distribució normal:

$$q(n) \approx \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} \quad \text{on} \quad x = \frac{\mu_1-\mu_2}{\sqrt{\sigma_1^2+\sigma_2^2}} \sqrt{n}$$

D'aquestes equacions podem veure que hi han dos fonts de pèrdues associades a aquest procediment. La primera pèrdua està associada a utilitzar n vegades el braç equivocat durant la fase experimental. La segona és el resultat d'elegir el braç equivocat associat a un pagament més baix (μ_2) després d'haver realitzat l'experiment. No podem estar absolutament segurs, al final de l'experiment, d'haver elegit

el braç correcte. Per tant podem, ocasionalment, elegir el braç erroni i tenir pèrdues en les restants N-2n tirades.

Podem trobar el nombre òptim de tirades n^* prenent la derivada de l'equació de pèrdues i la igualant-la a zero.

Aquest procediment és fàcil d'analitzar, però no hi ha dubtes de que hi ha millors formes, potser formes òptimes, de distribuir les tirades del millor braç. Holland (1975) ha realitzat càlculs que mostren com hem de realitzar les tirades per minimitzar les pèrdues. Això dona com a resultat una distribució de n^* tirades en el braç pitjor i $N-n^*$ en el millor on n^* és donat per la següent equació:

$$n^* \sim b^2 \ln\left[\frac{N^2}{8\pi b^4 \ln N^2}\right] \quad \text{on} \quad b=\frac{\sigma_1^2}{\mu_1-\mu_2}$$

Operant l'equació tenim que:

$$N-n^* \equiv N \sim \sqrt{8\pi b^4 \ln N^2} \cdot e^{\frac{-n^*}{2b^2}}.$$

En altres paraules per distribuir les tirades òptimament, hem de fer una mica més que un increment exponencial de tirades al braç observat com a millor. Malauradament aquesta estratègia és inviable perquè requereix un coneixement del que passarà avanç de que tingui lloc. De totes maneres aquest pot ser un bon ideal que una estratègia realitzable hauria d'intentar aproximar. L'aproximació experimental indicada anteriorment mostra com unes poques tirades amb un increment exponencial és donen al pitjor braç quant el nombre de tirades s'incrementa.

Altra mètode que s'aproxima inclús més a l'ideal de distribució de tirades és l'A.G. de tres operadors discutit anteriorment. La teoria d'esquemes garanteix això donant al menys un nombre exponencial d'increments de tirades als blocs de construcció observats com a millors. En aquest sentit l'A.G. és un procediment proper a l'òptim (Holland [23] i [24]) per buscar entre solucions alternatives.

Amb un A.G. no estem simplement resolent un problema de màquines escurabutxaques de dos braços. En un A.G. normal considerem

simultàniament solucions de moltes màquines escurabutxaques de k braços. Per reforçar aquest punt, primer considerem la forma de la solució d'una única màquina escurabutxaques de k braços i després demostrarem que l'A.G. usual pot ser pensat com la composició de moltes màquines escurabutxaques de k braços.

La forma dels límits de la solució a la màquina escurabutxaques de k braços fou també descoberta per Holland [23]. La solució trobada per la mínima pèrdua en la distribució de tirades en k braços és similar a la de 2 braços e indica que hem de donar un nombre d'increments exponencials en el nombre de tirades en el braç que s'hagi observat com a millor. Aquest resultat no és sorprenent, però connecta amb les nocions dels esquemes si considerem un conjunt d'esquemes competitius com una màquina escurabutxaques de k braços. Per veure això, definim la noció de conjunt competitiu d'esquemes, i llavors contem el nombre i mida dels problemes de màquines escurabutxaques de k braços resolts en un A.G. donat una certa longitud de string.

Dos esquemes A i B amb posicions a_i i b_i són competitius si per totes les posicions i=1,2, ,L tenim que $a_i = b_i = *$ o $a_i \neq *$, $b_i \neq *$ $a_i \neq b_i$ per al menys un valor d'i. Per exemple considerem el conjunt de vuit esquemes que competeixen en les posicions 2, 3 i 5 en els següents strings de longitud 7:

*00*0**
*00*1**
*01*0**
*01*1**
*10*0**
*10*1**
*11*0**
*11*0**

Podem començar a veure les connexions entre el problema de la màquina escurabutxaques de k braços i la nostre llista de 8 esquemes competitius. Ja que aquests esquemes estan definits sobre les mateixes posicions, necessitem donar un nombre d'increments exponencial a l'esquema observat com a millor tal com donàvem un increment exponencial de tirades al millor braç observat en el problema de la

màquina escurabutxaques de k braços. Una de les principals diferències entre els dos problemes és que en els A.G. tenim una gran quantitat de problemes procedint en paral.lel. Per exemple, per tres posicions fixes sobre un string de longitud 7 tenim $\binom{7}{5}=35$ dels $2^3=8$ problemes de màquines escurabutxaques de k braços. En general per esquemes d'ordre j sobre strings de longitud L tenim $\binom{L}{j}$ diferents problemes de k_j braços, on $k_j=2^j$. No tots els $\sum\binom{L}{j}=2^j$ problemes són tractats igualment bé a causa de l'encreuament, que té una tendència a trencar a aquestes màquines amb longituds definides llargues, com hem discutit anteriorment. A la següent secció contarem el nombre d'esquemes que són processats útilment per l'A.G.

5.4. Quants esquemes són processats útilment?

Els nostres arguments anteriors ens indicaven que tenim entre 2^L i $n\cdot 2^L$ esquemes processats en una població de strings de longitud L i mida n. Com sabem no tots aquests esquemes seran processats amb alta probabilitat perquè l'encreuament destrueix aquells amb una longitud definida relativament alta. En aquesta secció calcularem un límit inferior d'aquells esquemes que són processats d'una manera útil: aquells que són reproduïts amb un desitjable increment exponencial.

El més ampli comptatge d'esquemes efectius processats d'una manera útil es ben conegut, però en general és poc entès d'on surt l'ordre de n^3 $O(n^3)$ (Goldberg, [17]).

Aquesta estimació significa que encara que només es processin n estructures en cada generació, un A.G. processa de l'ordre de n^3 esquemes. Aquest resultat és tant important que Holland li va donar el nom especial de paralel.lisme implícit.

Redescobrim aquesta estimació per entendre les seves assumpcions i examinem la font del seu nivell computacional.

Considerem una població de n strings binaris de longitud L. Considerem només els esquemes que sobreviuen amb una probabilitat

més gran que P_s, una constant. Com a resultat, acceptant l'operació d'encreuament simple i una taxa de mutació baixa, només admetem esquemes amb una taxa d'error $\varepsilon < 1-P_S$. Això condueix a considerar esquemes amb longituds $L_S < \varepsilon(L+1) + 1$.

Amb una longitud d'esquema particular, podem estimar un límit inferior en el nombre d'esquemes únics processats per una població de strings inicialment aleatòria. Per fer això primer contem el nombre d'esquemes de longitud L_S o menors. Llavors multipliquem això per un mida de la població adequada, escollint en mitja no més d'un de cada esquema de longitud $L_S/2$.

Suposem que volem contar el nombre d'esquemes de longitud L_S en el següent substring de longitud L=10.

1011100010

Per fer això calculem el nombre d'esquemes en la primera cel.la de 5 posicions :

-----00010

L'últim bit de la cel.la és fixat. Això vol dir que volem esquemes de la forma:

%%%%1****

On els símbols * són jòquers i els símbols % indiquen un 1, un 0 o un jòquer. Clarament tenim $2^{(L_S-1)} \cdot (L-L_S+1)$ esquemes d'aquest string en particular. Per sobreestimar el nombre d'esquemes de la població podríem simplement multiplicar això per la mida de la població i obtindríem $n \cdot 2^{(L_S-1)} \cdot (L-L_S+1)$. Això obviament sobreestima el nombre correcte perquè segurament tenim duplicats de baix ordre en gran poblacions. Per refinar l'estimació, prenem un mida de la població $n = 2^{L_S/2}$. Escollint-lo d'aquesta manera, esperem tenir un o menys d'un de tots els esquemes de longitud $L_S/2$ o majors. Reconeixent que el nombre d'esquemes està distribuït binomialment, concluïm que la meitat són d'ordre més grans a $L_S/2$ i l'altra meitat més petits. Si només

considerem els d'ordre més gran, estimarem un límit inferior en el nombre d'esquemes.

$$n_S \geq n \cdot (L - L_S + 1) \cdot 2^{L_S - 2}$$

Això difereix de la sobreestimació prèvia en un factor 1/2. A més la restricció de la mida de la població a un valor $n = 2^{L_S/2}$ dona la següent expressió:

$$n_S = \frac{(L - L_S + 1) \cdot n^3}{4}$$

Ja que $n_S = C \cdot n^3$, concluïm que el nombre d'esquemes és proporcional al cub de la mida de la població i que és de l'ordre de n^3, $o(n^3)$.

6 Operadors i tècniques avançades

En els capítols anteriors hem analitzat el funcionament d'un A.G. simple basat en els tres operadors bàsics: selecció, on els individus eren seleccionats a l'atzar segons el seu valor de la funció de cost per ser creuats utilitzant el mètode de la ruleta de la sort, l'encreuament on es creuaven els strings seleccionats en parelles amb certa probabilitat per donar dos nous punts en la recerca amb característiques comuns a la dels seus pares i per últim la mutació que ens aportava, mitjançant el canvi a l'atzar d'un dels bits a l'atzar del string, una variació per tal de crear noves estructures.

Com hem vist anteriorment aquest A.G. de tres operants, de forma tant simple pot donar bons resultats, però des de que Holland els va definir han hagut altres aportacions en aquest tema. Per una part, s'han creat nous operants i per altre s'han modificat alguns dels ja existents. Sens dubte un del nous operants creats té molta més importància que la resta. Aquest operant és l'escalat de la funció i consisteix en una transformació prèvia de la funció de cost abans de ser tractada pels demés operants de l'A.G. Aquest ha resultat ser un dels operants fonamentals per l'aplicació de l'A.G. a cert tipus de problemes de difícil resolució.

Recordem que els algorismes genètics van sorgir com una modelització de la selecció natural i de la genètica. Una aplicació no del tot exacta ja que tant la genètica com la selecció natural són molt més complexes que lo explicat en els capítols anteriors. De la transferència d'aquests mecanismes biològics més complexes han sorgit altres operants dels A.G. Així explicarem els operants següents: Dominància, Diploidicitat, Abeyància, Operadors d'inversió i de reordenació i d'altres microoperadors.

6.1 Escalat de la funció

Des de l'estudi de De Jong dels "baseline" [15], l'escalat de la funció objectiu s'ha tornat una pràctica àmpliament acceptada. Aquesta

transformació de la funció de cost es fa per mantenir nivells apropiats de competència en el curs de la simulació. Sense l'escalat, ben d'hora surgeix una tendència d'uns pocs superindividus de dominar el procès de selecció. En aquest cas els valors de la funció objectiu deuen ser escalats per prevenir una presa de la població per part d'aquests superstrings. Després, quant la població ha convergit, la competició entre els membres de la població és menys forta i la selecció tendeix a divagar. En aquest cas l'escalat ha d'accentuar les diferències entre els membres de la població per continuar recompensant als individus de millors característiques. Actualment els mecanismes d'escalat no són nous, daten ja dels primers estudis empírics dels A.G. realitzats per Bagley [6] i Rosenberg [36]. Una revisió dels procediments d'escalat es presenten en Forrest [13]:

1) Escalat lineal
2) Truncament sigma
3) Escalat de llei potencial
4) Procediment de ranking o normalització lineal

6.1.1 Escalat Lineal

Definida la funció de cost original com f i la funció escalada com f'. L'escalat lineal efectua una relació lineal entre f i f' de la següent manera:

$$f'=af + b$$

Els coeficients a i b poden trobar-se de diferents formes. En tots els casos volem que la mitja de la funció escalada f'_{avg} sigui igual a la mitja de la funció original f_{avg} perquè un us del procediment de la selecció ens assegurarà que cada membre de la població amb un valor de la funció de cost a prop de la mitja contribuirà estadísticament amb un descendent a la propera generació. Per controlar el nombre de descendents que tindrà el membre de la població amb més alt valor de la funció de cost, elegim l'altra relació de l'escalat, que ens determinarà exactament els valor de a i de b, per obtenir un valor màxim de la funció escalada $f'_{max}=c_{mult} \cdot f_{avg}$, on c_{mult} és el nombre de còpies desitjades pel

millor membre de la població. Per poblacions típiques petites (n=50 a n=100) ha estat utilitzat un valor de c = 1.2 a 2 amb èxit.

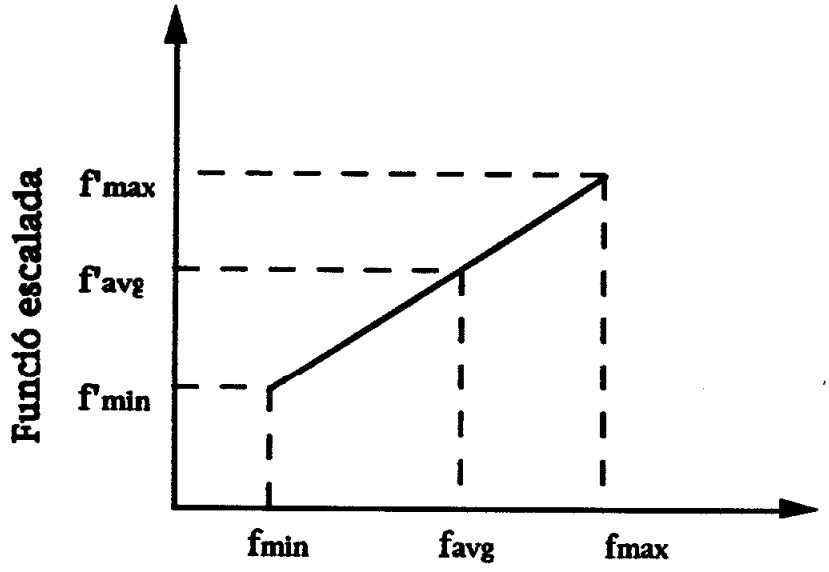


Figura 6.1

Al principi no tenim cap problema en aplicar la regla de l'escalat lineal, perquè els pocs individus extraordinaris són escalats cap a baix i els demés membres són escalats cap amunt, però, cap al final de l'execució, l'elecció de c_{mult} expandeix els valors de la funció original significativament. Això pot provocar dificultats en aplicar la regla de l'escalat lineal com es pot veure en la figura 6.2.

Aquest tipus de situacions són comuns en execucions madures, quant uns individus letals (amb funcions de cost molt més baixa que la resta) estan molt lluny de la mitja de la població i del màxim, que quant l'execució està a punt d'acabar estan molt junts. Si apliquem la regla de l'escalat en aquesta situació l'eixamplament requerit en els relativament a prop màxim i mitja causarà que els valors de la funció baixos arribin a ser negatius al ser escalats. Es disposa de diverses situacions per resoldre aquest problema; una d'elles és no escalar amb c_{mult} quant f'_{min} sigui més petit que zero i en aquest cas fer un nou escalat en el que $f'_{min}=0$. Això ens permet seguir amb aquest mètode.

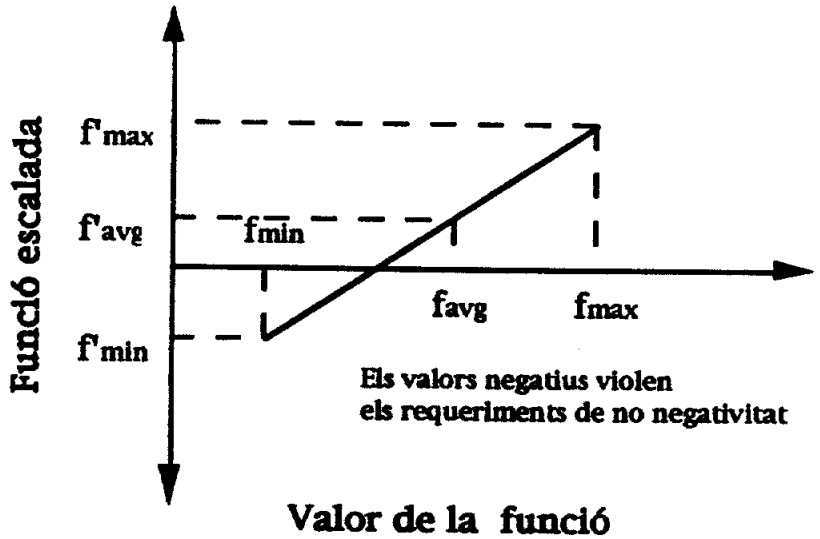


Figura 6.2

6.1.2 Truncament sigma

Per prevenir el problema anterior Forrest [13] va suggerir utilitzar la variança de la població per preprocesar el valor de la funció original abans d'escalar. En aquest procediment que s'anomena truncació sigma (σ) perquè utilitza informació de la desviació estandard de la població, restem una constant de la funció original de la següent manera.

$$f' = f - (\bar{f} - c \cdot \sigma)$$

En aquesta operació la constant c es troba com un múltiple raonable de la desviació estandard de la població (entre 1 i 3) i els valors negatius ($f'<0$) són arbitrariament igualats a zero. Després de la truncació sigma l'escalat pot procedir com s'ha descrit sense el perill dels resultats negatius.

6.1.3 Escalat de llei potencial

Guillies [15] va suggerir un escalat que utilitzava una llei de potències on la funció escalada es calcula elevant a una potència específica la funció original.

$$f'=f^K$$

En estudis limitats a una aplicació de visió artificial, Gillies va prendre una k=1.005. De tota manera, en general k és un valor que depèn del problema i podem necessitar canviar-lo durant l'execució per tal de eixamplar o estrènyer el rang de valors de la funció de cost.

6.1.4 Escalat de ranking

Els procediments, de natura una mica estrany de tots aquests mètodes d'escalat va portar a Baker [7] a considerar un procediment no paramètric per l'escalat. En aquest mètode la població és ordenada d'acord al seu valor de la funció de cost. Als individus se'ls assigna un nombre que només és funció de la seva posició dins d'aquesta ordenació. Baker va realitzar experiments comparant aquest escalat de ranking amb altres esquemes de selecció. Els resultats no estaven en general acabats però el procediment de ranking va mostrar la mateixa resistència a la sobre-selecció i a la infra-selecció demostrat pels esquemes de selecció normals utilitzats amb conjunció de procediments d'escalat.

Aquest mètode ha estat criticat perquè essencialment dissocia el valor de la funció objectiu de la funció de cost. De tota manera, la relació directa assumida entre el valor de la funció i el seu objectiu no està sustentada en teoria i el procediment de ranking ens proporciona un mètode consistent de controlar la descendència. El que es necessita són millors teories de la distribució de fills dins una població, que proporcioni un nombre apropiat de còpies sense tenir un detallat coneixement de la funció de cost o de la codificació.

6.2 Dominància, Diploidicitat i Abeyància

Pot semblar estrany que haguem ignorat el tema de la diploidicitat (parell de cromosomes) fins ara, ja que inclús els texts més elementals de genètica en parlen. Recordem que molts texts de genètica comencen amb una explicació de les plantes de pèsols de Mendel i alguna menció a la dominància. Si no ho hem tractat fins ara ha estat per recalcar la

importància fonamental de la selecció i l'encreuament. De tota manera l'existència de tants organismes diploides com poliploides ens porta a preguntar-nos si no ens seria útil utilitzar-lo en la recerca dels nostres A.G. Aquest capítol revisa el genotip diploide i els operadors de dominància per explicar el seu paper de protecció de solucions alternatives, d'una selecció excessiva.

Fins ara només hem considerat estructures haploides, això és estructures on un sol string portava tota la informació rellevant al problema a considerar. Sembla que quant la natura vol construir estructures complicades de vida ha de confiar en estructures cromosòmiques substancialment més complexes, cromosomes de dos branques o diploides. En la forma diploide un genotip té un o més parells de cromosomes (anomenats cromosomes homòlegs), cadascun d'aquests conté informació per la mateixa funció. Al principi aquesta redundància pot semblar innecessària i confusa. Perquè es necessita un parell de gens per decodificar la mateixa funció?. A més, quan el parell de gens decodifiquen diferents valors de la funció, com es fa a la natura per decidir quin al.lel es el que s'expressarà?.

Per respondre a aquestes qüestions considerem a un cromosoma diploide on les diferents lletres representen als diferents al.lels (diferents valors de la funció per cada gen).

AbcDe
aBCde

Cada posició (locus) de les lletres representen un al.lel. Les majúscules i les minúscules representen els al.lels alternatius a cada posició. En la natura cada al.lel pot representar una característica del fenotip diferent (el fenotip és la caràcterística del cromosoma que és expresada, és a dir que es realitza). Per exemple l'al.lel B pot ser el gen dels ulls marrons i l'al.lel b pot ser el dels ulls verds. Encara que això no és molt diferent al sistema anterior representat per una única estructura (haploide) tenim una diferència clara. Com ara tenim un parell de gens per cada posició, hem de tenir alguna cosa per decidir quin dels dos valors escollir, ja que no podem tenir els dos fenotips a l'hora (encara que la natura a vegades ho fa).

El mecanisme principal per eliminar aquest conflicte de redundància és per mitja d'un operador genètic que els genitistes anomenen dominància. Per una posició, s'observa que un al.lel (l'al.lel dominant) té prioritat (domina) als al.lels alternatius (els recessius) del seu lloc. Més específicament, un al.lel és dominant si és expressat (és mostra en el fenotip). Quant és aparellat hem assumit que totes les lletres majúscules són dominants i les minúscules recessives. El fenotip expressat pel cromosoma de l'exemple pot ser escrit així:

$$\begin{matrix} \text{AbCDe} \\ \cdots \\ \text{aBCde} \end{matrix} \quad \Rightarrow \quad \text{ABCDe}$$

A cada locus veiem que el gen dominant s'expressa sempre i el gen recessiu només ho fa quant s'acompanya d'altre recessiu. En el llenguatge dels genitistes diuen que el gen dominant s'expressa quant és heterozigot (la barreja de Aa $\Rightarrow$ A) o homozigot (pur, CC $\Rightarrow$ C), i l'al.lel recessiu només s'expressa quant és homozygot (ee $\Rightarrow$ e).

Els mecanismes de la diploidicitat i la dominància semblen prou clars. A un nivell més abstracte, podem pensar en la dominància com un mapeig de reducció de genotip a fenotip. Però, encara ens preguntem perquè aquesta despesa d'informació per després tallar-la i només utilitzar la meitat?. Deu haver bones raons per aquesta redundància del genotip diploide i l'enmascarament de l'operador dominant.

La diploidicitat i la dominància ha estat un dels temes d'estudi de la genètica i han estat expressades nombroses teories i explicacions del seu paper. La teoria que té més sentit en el context d'una recerca d'un A.G. diu que la diploidicitat ens proporciona un mecanisme per recordar al.lels i combinacions d'al.lels que van ser útils prèviament i la dominància proporciona un operador que protegeix aquests al.lels de record de la dura selecció en un entorn normalment hostil.

En un context natural, podem entendre la necessitat d'aquesta memòria de llarg terme i distribuïda, en termes de protegir aquesta memòria contra una ràpida destrucció. En el curs de l'evolució de la vida en la terra, el planeta ha sofert molts canvis en les seves condicions ambientals. Els organismes més efectius han estat aquells capaços d'adaptar-se més ràpidament a les condicions canviants. Els animals i les

plantes amb estructures diploides i poliploides han estat les més capacitades per sobreviure, perquè la seva constitució genètica no oblida fàcilment les lliçons apreses en els primers canvis ambientals.

La memòria redundant de la diploidicitat permet múltiples solucions (al mateix problema) per portar a terme, i només s'expressa una solució particular. En aquest sentit les velles lliçons no són oblidades per sempre, i la dominància permet que les velles lliçons siguin recordades i portades a terme ocasionalment.

6.2.1 Perspectiva històrica de la Dominància i la diploidicitat en els A.G.

Algunes de les aplicacions dels A.G. més noves incorporen genotips diploides i mecanismes de dominància.

En la dissertació de Bagley, un parell de cromosomes diploides són mapejats a un fenotip particular utilitzant una codificació de dominància variable com a part del propi cromosoma (Bagley [6]).

Cada locus actiu conté, darrera de l'informació del paràmetre al qual està associat i el valor del paràmetre en particular, un valor de dominància. A cada locus l'algorisme simplement selecciona l'al.lel que té un valor més alt de dominància. Al contrari del cas biològic on es permet una dominància parcial, la nostre interpretació demana que només un dels al.lels sigui expressat. El procés de decissió en el cas d'empat (valors de dominància igual) que inclou efectes de posició, és una mica complicat i per tant requereix una explicació amb més detall.

Un dels cromosomes és seleccionat arbitràriament per ser la "clau" cromosòmica i les seves posicions són examinades per torn d'esquerra a dreta. Cada vegada que es troba un lloc actiu, el contingut dels locus homòlegs és recuperat de l'altre cromosoma. Els valors de dominància són comparats i es selecciona l'al.lel amb un valor més gran de dominància. Si els valors de dominància són idèntics, la dominància segueix la clau cromosòmica. Això és, es prova el lloc actiu més proper en la clau cromosòmica a l'esquerra del lloc que estem examinant. Si el locus d'aquell lloc és dominant, el locus present en la clau cromosòmica es posa dominant, d'altra manera l'homòleg domina. Si el locus que examinem ocupa el lloc actiu inicial en la clau cromosòmica, el lloc de la clau domina.

L'introducció de valors de dominància per cada gen va permetre a aquest esquema adaptar-se a generacions successives. Malauradament, Bagley va trobar que els valors de dominància tendien a fixar-se molt d'hora en les simulacions, i per tant deixava la determinació de la dominància en mans del seu esquema per trencar l'empat (fet de forma tant complicada i arbitrària). Per empitjorar les coses, Bagley va prohibir que el seu operant mutació processés operadors de dominància. Per tant, això agreujava la prematura convergència dels valors de dominància. A més, Bagley no va comparar els esquemes haploides i els diploides, i en tots els casos l'entorn era estacionari. Al final la convergència dels valors de dominància a totes les posicions portaven a uns mecanisme arbitraris i aleatoris i els resultats resultaren incunclosos.

Els estudis orientats a la biologia de Rosenberg [36] contenen un model de cromosoma diploide però, de tota manera, encara que les modelacions químiques van ser modelades amb detall, la dominància no va ser considerada com a efecte separat. Al contrari, alguns efectes de dominància en aquest estudi eren el resultat de la presència o de l'absència d'un enzim en particular. L'enzim podia llavors inhibir o facilitar una reacció bioquímica, i així controlar que el fenotip s'expressés.

Els estudis de Hollstien [25] inclouen diploidicitat i tenen desenvolupats mecanismes de dominància. De fet Hollstien descriu mecanismes desenvolupats de dominància i llavors utilitza el més senzill en el seu estudi d'optimització de funcions. En el primer esquema, cada gen binari és descrit per dos gens, un gen modificador i un gen funcional. El gen funcional té un valor de 0 o de 1 i es decodifica en un paràmetre de la manera normal. El gen modificador pren valors de M o m. En aquest esquema l'al.lel 0 era dominant quant hi havia al menys un al.lel M present en un dels llocs dels modificadors homòlegs.

Això resultava en una expressió de la dominància mapejada com a la figura 6.3. Hollstien va reconèixer que aquest esquema de dominància de dos locus podia ser canviat per un esquema més simple d'un locus introduint un tercer al.lel en cada locus. En aquest esquema trièl.lic, Hollstien creava al.lels d'un alfabet de tres elements {0,1,2}. En el seu esquema el 2 prenia el paper de 1 dominant i el 1 prenia el paper de 1 recessiu. El mapa d'expressió dominant que va utilitzar es pot veure en la figura 6.4

	0M	0m	1M	1m
0M	0	0	0	0
0m	0	0	0	1
1M	0	0	1	1
1m	0	1	1	1

Figura 6.3. Mapa de dominància de dos locus.

L'acció d'aquest mapeig pot ser expressada dient que el 2 i el 1 es mapegen a 1, però el 2 domina al 0 i el 0 domina al 1. Holland (1975) va discutir i analitzar més tard les prestacions d'aquest mateix esquema triel.lic, però va introduir una simbologia més clara $\{0,1_0,1\}$ que la de Hollstien {0,1,2}.

	0	1	2
0	0	0	1
1	0	1	1
2	1	1	1

Figura 6.4. Mapa de dominància trièl.lic, amb un locus.

L'esquema triel.lic de Hollstien-Holland és el més clar, i a més és l'esquema més simple suggerit per als A.G. artificials, ja que combina un mapeig de la dominància i d'informació de l'al.lel en una única posició.

Amb aquest esquema l'al.lel més efectiu es torna dominant, per tant protegeix el recessiu. Es requereix un mínim increment d'amagatzamatge (mig bit extra per locus) i a més els canvis de

dominància es poden realitzar fàcilment amb un operador de mutació, mapejant un 1 a un 2 o viceversa. Fins i tot amb la claredat de l'esquema, els resultats de Hollstien amb la diploidicitat eren confusos.

Encara que les simulacions del seu "Breed Type III" mantenien majors diversitats de població (mesurada com a variança de la població) que la de les seves simulacions haploides, no hi havien millores significatives en les prestacions mitjes o finals. Això pot resultar sorprenent fins que ens adonem que el seu test només contenia funcions estacionàries. Si el propòsit de la diploidicitat-Dominància és protegir o rebutjar al.lels, només podem esperar diferències significatives entre els A.G. diploides i haploides quant l'entorn canvia amb el temps. És sorprenent que aquests canvis no fossin estudiats conjuntament amb aquest operador.

Brindle [9] va realitzar experiments amb un nombre d'esquemes de dominància en un conjunt d'optimització de funcions. Malauradament, les funcions que va utilitzar i la codificació que va emprar van ser qüestionades. A més, va ignorar el treball previ en la dominància artificial i la diploidicitat, i un nombre d'esquemes que va desenvolupar no tenien base teòrica o precedents biològics.

En la seva dissertació Brindle va simular i comparar sis esquemes, però degut a que la base dels seus test eren inapropiats i perquè ell només va considerar funcions estacionàries, la diploidicitat i la dominància no van ser ben provades en el seu estudi.

S'han portat a terme estudis més recents (Goldberg i Smith, [20], R.E. Smith [38] i [39]) en el paper de la dominància i la diploidicitat com a mecanismes i estructures desusades. Smith i Goldberg han comparat les prestacions d'un A.G. haploide, un A.G. amb un mapeig de dominància fixat (1 domina al 0) i un A.G. amb un mapeig de dominància trièl.lic de Hollstien_Holland en el problema de la motxilla, cec i no estacionari.

En un problema de motxilles normal, l'objectiu a maximitzar és el valor total d'objectes col.locats en la motxilla objecte amb una o més restriccions de pes. Matemàticament, el problema pot ser descrit com segueix:

$$\max \sum v_i \cdot x_i \qquad \text{on } x_i \in \{0,1\} \text{ amb } \sum w_i \cdot x_i \leq W$$

En un problema de motxilles cec, l'algorisme no té coneixements de l'estructura del problema, valors de v_i o valors de w_i. En aquest cas es va complicar el problema fent que les restriccions de pes fossin periòdiques amb el temps. $W(t) \in \{W_0, W_1\}$ canviava a cada període T de generacions a l'altre valor. Les figures comparen un A.G. haploide amb un A.G. diploide simple.

L'A.G. haploide es incapaç de seguir l'oscil.lació, mentre que l'esquema diploide simple és capaç, fins a cert punt, de canviar. També es va realitzar experiments amb l'esquema de Hollstien-Holland. Aquests resultats són una millora important sobre el mapa de dominància fix original, com esperàvem.

Ja que l'esquema trièl.lic preserva l'evolució de la dominància en cada locus, la població pot adaptar-se més ràpidament i més completament del que és possible en casos amb un mapeig de dominància fix o una estructura haploide.

6.3 Inversions i altres operadors de reordenament

En un problema arbitrari no podem estar segurs que un A.G. simple ens porti a la solució del problema. Al principi pot semblar desconcertant fins que reconeixem que la natura tampoc està segura de les seves codificacions i ha inventat operadors que busquen millors codificacions al mateix temps que cerquen millors valors pels seus al.lels.

En aquesta secció estudiarem aquests operadors de reordenament per veure si el mateix mecanisme pot ser utilitzat en la recerca d'A.G. artificials.

El mecanisme natural primari responsable de recodificar un problema és l'operador d'inversió. Sota l'inversió dos punt són escollits a la longitud del cromosoma, es talla el cromosoma per aquests punts, i els punts finals de la secció tallada canvien de lloc. Al principi aquest operador sembla una mica estrany si ho veiem en el context dels nostres cromosomes artificials simples. Per exemple considerem el següent string de 8 posicions on es troben 2 llocs a l'atzar (per exemple els llocs 2 i el 6 marcats amb un caràcter |)

10|1110|11

Si utilitzem l'operador d'inversió a les fosques, obtindríem el següent string:

100111111

No està del tot clar com l'operador d'inversió ens ajuda en la recerca d'una nova representació. De fet sembla que només reordena parcialment els al.lels dintre del string. El problema que tenim aquí es degut a que la nostre representació és molt simple. A la natura els gens poden ser reordenats a un string i encara ser responsables de la producció del mateix enzim. En altres paraules, a la natura el significat de l'al.lel és independent de la seva posició. Per tenir la mateixa flexibilitat en la nostre representació, anomenem els nostres al.lels utilitzant enters entre 1 i 8 i mirem que passa quant apliquem l'operador d'inversió sota la nostre representació ampliada.

12|3451678
10|1111011

Ara quan fem la mateixa inversió, també portem el nom de l'al.lel:

12654378
10011111

Utilitzant aquesta representació ampliada, els valors dels bits conserven el seu significat d'acord a la seva posició. En termes biològics, aquesta representació ampliada separa el gen del locus. Una conseqüència interessant d'això és que un operador d'inversió simple no té un efecte immediat en el valor de la seva funció de cost.

Encara que ara està clar que una representació ampliada permet a l'inversió actuar sense canviar el significat de l'al.lel, no està clar encara que contribució positiva té l'operador d'inversió i la representació ampliada al nostre A.G. artificial. Després de tot, hem raonat que una inversió simple no té cap efecte en el valor del string, llavors perquè es

molesta la natura amb aquest joc ocasional de canviar cadires?.

Considerarem la teoria darrera de l'operador d'inversió en aquest capítol. Per ara només considerarem que l'inversió pot ser útil al trobar bones distribucions de strings al mateix temps que altres operadors genètics busquen bons conjunts d'al.lels. Si la població actual conté una mala ordenació (on els al.lels amb altes interaccions epistàtiques o no lineals estan espaiades a grans distàncies en el cromosoma), l'encreuament destruirà importants paquets d'al.lels amb gran probabilitat.

Així, si l'esquema de reordenació pot solucionar el posicionament dels al.lels, llavors hi ha possibilitats d'obtenir bons ordenaments del al.lels que permetrà conseqüentment una propagació més eficient dels "blocs de construcció".

Ara examinarem aquest argument més rigorossament. En la propera secció, revisarem les investigacions anteriors dels operadors de reordenament a la recerca d'A.G.

6.4 Perspectiva històrica dels operadors de reordenament en A.G.

Les simulacions per ordinador realitzades per Bagley [6] contenien un operador d'inversió. Ell va implementar un operador d'inversió simple i la representació dels strings ampliada discutides anteriorment.

Una de les decissions a les que es va tenir que afrontar és com creuar dos parells de strings no homòlegs. Per veure que això és important, considerem un encreuament simple entre dos strings A i B:

A = 123|5678 B = 126|4378
1011|1011 1001|1111

Si ingènuament creuem els dos strings utilitzant un encreuament simple, potser al lloc d'encreuament quatre (marcat amb el caràcter |) obtindrem dos nous strings fills de la següent manera:

A = 1234|4378 B = 1265|5678
1011|1111 1001|1011

Podem adonar-nos immediatament que cap dels dos strings fills conté un gen sencer, i en en general aquest és l'argument primari contra la permissivitat d'encreuaments simples entre strings arbitràriament ordenats. Bagley va adoptar una manera directa per resoldre aquesta dificultat. Va prohibir l'encreuament simple entre strings no homòlegs. Malauradament, els resultats sota aquestes condicions van ser decepcionants.

L'efecte més obvi de l'efecte de l'inversió és un increment de la durada de l'execució de la simulació. Això era així perquè una de les conseqüències de l'inversió era un decreixement en el nombre d'encreuaments efectius ja que no es permetia l'encreuament entre strings no homòlegs. Com es pot veure una taxa d'encreuaments més baixa causa un efecte advers en la simulació, i aquest efecte és el que predomina en els resultats. La conseqüència desitjable de l'inversió, que és l'aparició a la població de gamets amb unions de gens geogràfics que reflectin combinacions, no es van observar.

Bagley va atribuir la manca d'èxit a les característiques del seu problema (una tasca de lloc). Va concloure que el seu problema era insuficientment difícil (epitàstic) per donar a l'inversió una bona oportunitat de mostrar les seves qualitats. Dit d'altra forma, l'A.G. simple treballava suficientment bé sense l'inversió en comparació amb l'A.G amb inversió com perquè es mostrés cap millora en la velocitat de convergència

De totes maneres, hi han altres raons perquè Bagley no pogués veure una avantatge en l'inversió. Recordem que Bagley no va permetre creuaments entre strings no homòlegs. Això essencialment evita l'encreuament entre molt parells de membres de la població. La natura ha adoptat una política més liberal. Altres polítiques menys restrictives han estat adoptades també per altres investigadors interesats en l'inversió.

L'estudi de Cavicchio [10] d'una recerca genètica en "pattern recognition" utilitzava un operador d'inversió sense restriccions d'inversions ni d'encreuament. Hem de tenir en compte que en el seu problema els gens consistien en nombres d'identificació de pixels

agrupats en clusters detectors. Amb aquesta representació, es pot fer qualsevol inversió, garantint sempre un cromosoma viable.

Per tant, l'encreuament subseqüent de parells de strings en qualsevol punt generen nous strings amb significat independent de l'ordenació del string. Encara que aquestes propietats de la codificació depenien del problema que ell utilitzava, els resultats de Cavicchio eren esperançadors. En un conjunt d'experiments, relativament alts valors d'creuaments i d'inversions produïen una bona taxa de convergència i un alt nivell de prestacions.

Els estudis de Frantz [12] de l'epístasis en la recerca genètica artificial no var tenir tant d'èxit en demostrar l'utilitat de l'inversió. Va intentar diverses variacions en els operadors d'inversió i en les regles d'aparellament en funcions amb diferents i controlades no linealitats. Va provar dos variants de l'inversió.

1 Inversió lineal
2 Inversió lineal i final

L'inversió lineal és simplement el nom que Franz va donar a l'inversor simple de 2 punts descrit anteriorment. L'inversió lineal i final realitza una inversió lineal amb una probabilitat especificada (0.75). Si no es realitza l'inversió lineal, es pot realitzar l'inversió final amb igual probabilitat (0.125) a l'esquerra i dreta del string. Sota l'inversió final, el final esquerre i dret del string s'agafa com un únic punt d'inversió i el segon punt d'inversió és elegit uniformement a l'atzar de punts no llunyans a la meitat de la longitud del string. L'inversió lineal i final fou un intent de Franz de minimitzar la tendència dels inversors lineals de trencar al.lels localitzats a prop del centre del string desproporcionadament a aquells que es trobaven a prop dels finals.

Franz va aplicar qualsevol dels seus operadors d'inversió d'un o dels dos modes:

1. Inversió contínua
2. Inversió en massa

En el seu mode d'inversió continua, l'inversió fou aplicada amb una probabilitat d'inversió especificada P_i, a cada nou individu tan bon punt és creuava. En el mode d'inversió en massa, no es realitza cap inversió fins que no es crea una nova població. Després la meitat de la població sofreix una inversió idèntica (utilitzant els mateixos punts d'inversió).

L'inversió en massa va ser dissenyada per eliminar la proliferació de subpoblacions no actives que acompanyaven a parelles estrictament homòlegs.

Frantz va intentar quatre regles d'aparellament per prevenir el problema usual dels creuaments ingenus entre parelles de strings no homòlegs.

1. Aparellament estrictament homòleg.
2. Aparellament viable.
3. Aparellament de qualsevol patró.
4. Aparellament del millor patró.

L'aparellament estrictament homòleg és la mateixa forma que l'adoptada per Bagley on només els strings homòlegs podien emparellar-se. L'aparellament viable permet un intent d'encreuament entre strings no homòlegs; però si els strings resultants no tenen una dotació de gens sencera no són inserits en la nova població. En els aparellaments de qualsevol patró, es seleccionen a l'atzar dos parelles (dos strings a aparellar-se) a l'atzar, i un dels dos s'elegeix perquè sigui el que marqui l'ordre. L'altre string es mapeja a l'ordenació principal i es realitza un encreuament simple. L'operació de mapeig garanteix la viabilitat del encreuament. L'aparellament del millor patró és la mateixa de la de qualsevol patró excepte que s'elegeix el millor dels dos strings per determinar quin serà l'ordenament principal.

Encara que va intentar un gran nombre de variacions i d'opcions, Franz no va poder demostrar clarament els efectes de posició. A més, no va poder demostrar clarament cap avantatge de l'inversió en cap combinació, mode o forma d'aparellament. El problema subjacent era l'entorn del problema adoptat. Frantz va utilitzar una combinació lineal de funcions lineals i no lineals de bits cadascun d'elles combinades entre sis o set al.lels en un string de L=25. Les funcions no lineals utilitzaven una presa de les 2^6 o 2^7 alternatives entre els sis o set al.lels del grup.

Malauradament aquestes funcions eren insuficientment difícils per requerir inversió. Com va puntualitzar Bethke [8], no només ha de ser una funció difícil i epitàstica per l'A.G., l'epítasi deu ser, a més, enganyosa. En altres paraules, els esquemes curts i d'altes prestacions deuen apuntar a àrees pobres de l'espai de recerca. Aquest no era el cas en l'estudi de Frantz, ja que un A.G. simple sense inversions podia trobar bones solucions fàcilment.

Després dels estudis de Frantz, han aparegut altres investigadors dels operadors de reordenament i d'inversió. Holland [24] fa una breu menció a la inversió, presentant modificacions al teorema dels esquemes per incloure l'efecte aproximat del encreuament simple.

Després no tenim moltes mencions als operadors d'inversió fins a la conferència internacional d'A.G. i la seva aplicació al 1985. A aquesta conferència, varis autors (Davis, [13], Goldberg i Lingle [18], Smith [37]) descriuen la construcció d'operadors de reordenament que combinen característiques de la inversió i del encreuament en un operador únic. Aquests operadors, encara que han estat trobats independentment, són similars: encreuament parcialment acceptat (PMX), encreuament ordenat (OX) i encreuament circular (CX).

PMX va sorgir per poder afrontar el problema del viatjant ceg. En el problema del viatjant, un viatjant, un hipotètic venedor deu fer una gira per un conjunt donat de ciutats en l'ordre que minimitza la distància total a recórrer. En el problema del viatjant cec, el venedor té el mateix objectiu amb la restricció que no sap quina distància viatja fins que ha recorregut totes les ciutats.

El problema del viatjant és un problema suficientment difícil (és un membre de la classe de problemes que es creu que són irresolubles en un temps polinòmic) sense la imposició de la restricció de la ceguesa.

Sembla natural codificar el problema d'ordenació com una permutació de vuit elements en la forma de presentació. Per exemple, en un problema de vuit ciutats on cada ciutat és visitada en ordre ascendent, la gira pot ser representada així:

12345678

La permutació de la representació de les ciutats visitades en un ordre invers podria representar-se per la següent permutació.

87654321

Encara que aquesta representació sembla prou natural, com encaixa en els esquemes representats a les nostres aplicacions d'A.G. simples?. En les discussions prèvies vam patir per separar el locus de l'al.lel del seu significat. Matemàticament diem que el valor de la funció de cost f només havia de dependre del valor de l'al.lel v, f=f(v), de tota manera en molts problemes pot ser útil permetre que la funció depengui d'alguna manera de l'ordenació dels strings: la funció de cost dependrà d'alguna combinació del valor de l'al.lel v i de l'ordenació o, f=f(v,o)

En el problema del viatjant la representació de permutació va a l'extrem oposat. Tenim un problema que només depèn de l'ordenació de la informació, f=f(o). En general hem de reconèixer que hi ha un espectre de possibles mètodes de codificació que més o menys depenen de l'ordenació i del valor de l'al.lel.

Quant permetem aquesta possibilitat, necessitem buscar una operador anàleg a l'encreuament, que permeti intercanviar semblances d'ordenament entre un parell de strings pares per donar descendents. Recordem que la potència de la recerca genètica és l'efecte combinat de la selecció i de la recombinació randomitzada. La mutació és simplement una política d'assegurances contra la pèrdua irreversible de material genètic.

Si hem de crear operadors amb la potència de l'encreuament, deuen ser operadors binaris i deuen combinar l'ordenament de "blocs de construcció" dels parents amb valors per sobre de la mitja d'una manera sensible. Goldberg i Lingle [18] van suggerir l'operador anomenat "encreuament parcialment adaptat" (PMX).

Sota el PMX, dos strings (les permutacions i els seus al.lels associats) són alineats, i s'agafen dos llocs de encreuament uniformement a l'atzar sobre la longitud del string. Aquests dos punts defineixen una secció d'adaptació que és emprada per efectuar un encreuament mitjançant operacions d'intercanvi posició a posició. Per veure això considerem dos strings:

A = 984|567|1320
B = 871|230|9646

PMX procedeix mitjançant un tipus d'intercanvi de posicions. Primer mapeja el string B cap al string A, el 5 i el 2, el 3 i el 6, i el 0 i el 7 intercanvien els seus llocs. Similarment es mapeja el string A cap al string B, el 5 i el 2, el 6 i el 3 i el 7 i el 0 canvien de lloc. Seguint el PMX ens trobem amb dos descendents A' i B' :

A' = 984|230|1657
B' = 801|567|9243

On cada string conté informació d'ordenació parcialment determinada per cadascun dels seus parents.

Després de fer varies simulacions podem veure com els resultats obtinguts mitjançant PMX són molt millors que els aconseguit mitjançant l'inversió simple.

S'han obtingut altres operadors similars a PMX (Davis [12] i [13]; Davis i Smith, [14]; Oliver, Smith i Holland, [29]; Smith [37]) aplicats a problemes de representació de permutacions. Aquets dos operadors són el encreuament per ordre i el encreuament cíclic.

L'operador de encreuament d'ordre comença d'una manera similar a com ho fa el PMX. Començant amb els strings de l'exemple A i B utilitzats per mostrar el funcionament del PMX, seleccionem una secció d'adaptació (per comparar, escollim la mateixa que en l'exemple de l'aplicació de PMX)

A = 984|567|1320
B = 871|230|9646

Com al PMX, cada string mapeja els formants de la secció d'adaptació de la seva parella. En lloc d'utilitzar un intercanvi punt a punt per realitzar el mapeig tal com ho fa el PMX, l'encreuament ordenat utilitza un moviment lliscant per transferir els forats deixats al transferir les posicions mapejades. Per exemple quant el string B mapeja el string A, les ciutats 5, 6 i 7 deixen un forat (marcat amb una H) en el string.

B = 8H1|231|0|9H4H

Els forats són llavors emplenats amb els noms de les ciutats de la secció adaptada preses de la parella. Realitzant aquesta operació i completant l'encreuament complementari obtenim com a descendència A' i B' el següent:

A' = 567|23|01984
B' = 230|567|9481

Encara que PMX i OX són molt similars, treballen amb diferents tipus de semblances. PMX tendeix a conservar la posició absoluta de les ciutats, mentre que OX tendeix a respectar la posició relativa entre elles. Ara examinarem l'últim d'aquests operadors de permutació, l'operador cíclic.

L'encreuament cíclic és un operador d'encreuament diferent. El encreuament cíclic realitza recombinacions sota la restricció de que cada ciutat ve d'un parent o d'un altre. Per veure com es fa això, comencem amb els recorreguts C i D:

C = 9821745063
D = 1234567890

En lloc de trobar llocs d'encreuament, comencem per l'esquerra i escollim una ciutat del primer parent:

C' = 9---------

Ja que volem una ciutat presa de cadascun dels dos parents, l'elecció de la ciutat 9 del string C significa que hem d'agafar la ciutat 1 del string C a causa de la posició 1 del 1 en el string D

C' = 9--1------

Aquesta selecció per torn continua fins que hem acabat amb el següent patró.

C' = 9--1-4--6-

La selecció del 6 significa que deuríem ara escollir un 9 del string C, però això no és possible; un 9 ja ha estat seleccionat a la primera ciutat. Al tornar, eventualment, a la mateixa ciutat completem un "cicle", que dona a l'operador el seu nom. Després d'haver completat el cicle. les ciutats que queden són emplenades amb l'altre string. Completant l'exemple i realitzant l'encreuament complementari trobem els següents itineraris fills:

C' = 9231547860
D' = 1824765093

Holland, Smith i Oliver [29] han trobat més resultats tan teòrics com pràctics comparant els operadors PMX, OX i CX.

6.5 Altres Microoperadors

S'han suggerit un gran nombre d'operadors de baix nivell per utilitzar-los en la recerca genètica adaptativa. Encara que aquests només afegeixen una potència marginal en comparació a l'adició de dominància i als operadors de reordenament, revisarem breument alguns d'aquests operadors suggerits per utilitzar-los en l'A.G.: segregació, duplicació intracromosomal i diferenciació sexual.

6.5.1 Segregació, translocació i estructures multicromosòmiques

A la natura molts organismes tenen genotips amb molts cromosomes. Per exemple l'ésser humà té 23 parells de cromosomes diploides. Per adaptar una estructura similar en la recerca genètica artificial hauriem d'extendre la nostre representació i permetre que el nostre genotip fos una llista de k parells de strings (assumint diploidicitat). Però perquè ho volem tot això? Holland [24] va suggerir que els genotips multicromosòmics podien ser útils a l'extendre la

potència de l'A.G. quant s'utilitzaven amb conjunció de dos operadors: la segregació i la translocació.

Per veure com treballa la segregació, simplement imaginem el procès de formació dels gamets quant tenim més d'un parell de cromosomes en cada genotip. L'encreuament succeeix com sempre; però, quant anem a formar un gamet seleccionem aleatòriament un dels cromosomes haploides. Aquesta elecció a l'atzar conegut com a segregació, trenca efectivament la relació que pot existir entre els gens dels diferents cromosomes.

Per suposat els gens localitzats al mateix cromosoma encara estan relacionats més o menys, depenent de la distància que els separi. La segregació és un operador útil si els gens relativament independents han d'estar localitzats a diferents cromosomes. En aquest sentit, en la relació trencada, els al.lels dèbils no poden trobar el camí per sobreviure amb els al.lels relacionats forts. La translocació podria veure's com un operador de encreuament intercromosòmic. Aquest organitza el cromosoma d'una manera apropiada i permet que la segregació ho aprofiti.

No hi ha hagut molta experimentació amb aquests operadors. En A.G. Hollstien [25] va utilitzar un operador de segregació al estudiar genotips diploides. Ell entenia la segregació com una ordenació a l'atzar dels al.lels parents durant la meiosi. Els experiments que va realitzar eren limitats i no van generar cap conclusió sobre la forma de l'operador. Estudis posteriors d'A.G. en "machine Learning" han requerit genotips ampliats i operadors similars a la segregació i a la translocació.

6.5.2 Duplicació i deleció

La duplicació intracromòsomica actua duplicant un gen particular i col.locant-lo amb el seu progenitor en el cromosoma. La deleció actua treient un gen duplicat del cromosoma. Holland va suggerir que aquests operadors eren mètodes efectius per controlar la taxa de mutació.

Si la taxa de mutació es manté constant i la duplicació causa k còpies d'un gen particular, la probabilitat de mutació efectiva (la probabilitat que al menys una de les k còpies sigui mutada) d'aquest gen es multiplica per k. Conseqüentment quant tenim una deleció la taxa de mutació efectiva disminueix. Hem de tenir en compte, que hem de

decidir quin dels gens alternatius s'expressarà. Ens afrontem a la mateixa situació que quant parlàvem de dominància i diploidicitat. De fet podem veure les múltiples còpies introduint una dominància intracromosòmica habitual que es dona a la diploidicitat. Holland va suggerir un esquema d'arbitració per escollir entre les diferents opcions, però no hi han publicats estudis d'aquests mecanismes fins ara.

6.5.3 Determinació sexual i diferenciació

En aquesta secció examinarem els mecanismes de la diferenciació sexual i examinarem com poden ser útils a la recerca de l'A.G.

A la natura molts organismes deuen ser de dos (o més) sexes diferents, i d'alguna manera els dos deuen anar junts per propagar l'espècie. Els detalls de la diferenciació sexual són diferents en les diferents espècies; de tota manera, l'exemple humà és suficientment representatiu per nosaltres, com per utilitzar-lo com a model.

Un dels 23 parells de cromosomes humans determina el sexe. Les femelles tenen dos cromosomes sexuals homòlegs (cromosomes X) i els mascles tenen dos cromosomes sexuals diferents (un cromosoma X i un cromosoma Y). Durant la gametogènesi els mascles formen esperma, que porta tant l'esquema X com el Y (en idèntiques proporcions); les femelles produeixen ous, que porten només cromosomes X. Quant la fertilització té lloc la segura producció de cromosomes X produïda per les femelles amb la incerta combinació amb un cromosoma X o Y produïda pel mascle ens porta a l'esperada i observada relació sexual 1:1 de mascles i femelles.

Un gran nombre de factors no relacionats amb el gènere estan relacionats amb els cromosomes sexuals. Aquests són els factors relacionats amb el sexe i s'identifiquen més freqüentment amb el cromosoma X.

Encara que tot això és interessant ens hem de preguntar que pot fer la diferenciació i la determinació per la recerca genètica artificial?. Malauradament no hi ha publicat cap estudi teòric o empíric de la determinació sexual i la diferenciació en la literatura dels A.G. De tota manera un raonament directe ens pot portar a una explicació possible sobre la seva utilitat. Clarament l'establiment de la diferenciació sexual divideix a l'espècie en dos (o més) grups cooperatius. Aquesta bifurcació

permet als mascles i a les femelles especialitzar-se una mica, i per tant cobrir el marge de comportament necessari més àmpliament que seria possible amb una única població competitiva.

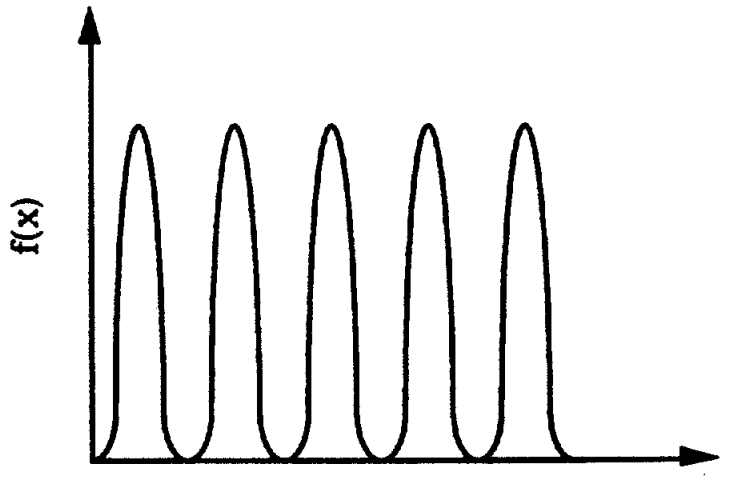
És molt probable que proves futures de sexe en la recerca d'A.G ens mostrin l'avantatge dels operadors sexual en problemes en que es requeixi una combinació de cooperació i especialització.

6.6 Els Nínxols i l'especiació

L'especialització permesa per la diferenciació sexual és portada més enllà a la natura per mitjà de l'especiació i de l'explotació del nínxol.

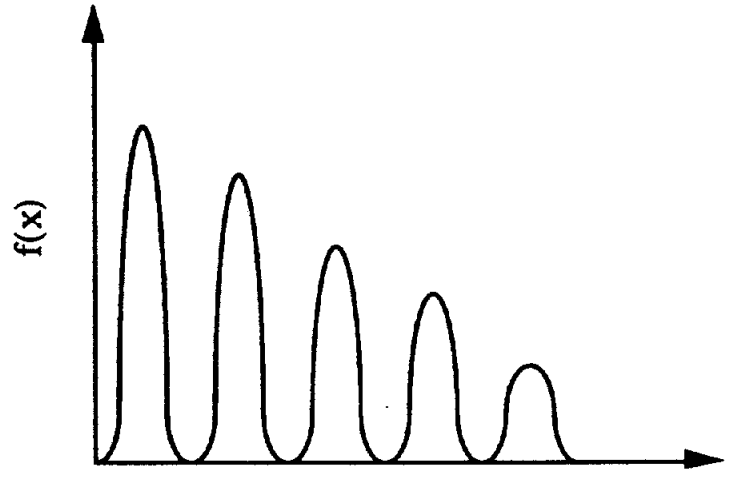
Intuïtivament podem veure un nínxol com un treball d'un organisme en un entorn, i podem pensar en les espècies com una classe d'organismes amb característiques comuns. Aquesta separació de l'entorn i dels organismes explotant aquest entorn en subconjunts diferents és tant comú a la natura que es mereix donar-li una segona ullada. Veient això, és estrany que no hagem observat cap subpoblació estable de strings (espècies) utilitzant diferents dominis de la funció (nínxols) en cap dels nostres exemples. Aquesta secció ens mostrarà com l'inducció d'espècies i nínxols pot ajudar en la recerca de l'A.G.

Per entendre perquè hem de promoure la formació de nínxols i espècies en l'A.G. considerem l'acció d'un A.G. simple en la funció simple mostrada en la figura 6.5. Si comencem amb una població inicial escollida uniformement a l'atzar, obtindrem un nombre de punts relativament estesos sobre el domini de la funció. Quant la selecció, l'encreuament i la mutació actuïn, la població pujarà els pics, i l'última distribució de la majoria dels strings estaran a prop d'un dels 5 pics. Aquesta convergència última en un dels pics o d'un altra sense diferència és deguda a errors estocàstics al mostrejar degut a que la mida de les poblacions són petites. D'alguna forma el que volem és reduir aquest efecte i permetre que subpoblacions estables es formin en cada pic.



Pics Iguals
Figura 6.5

També vodríem modificar les característiques d'un algorisme simple en problemes multimodals on els pics no són tots de la mateixa magnitud. Per exemple considerem la gràfica de la figura 6.6. En aquest problema tenim 5 pics com en l'exemple previ però els pics decreixen en magnitud amb valors de x creixents. Les prestacions d'un A.G. simple són fàcils de preveure. Un A.G. simple amb suficients generacions, distribuirà tots els seus punts sobre el pic més alt. En molts problemes voldríem localitzar subpoblacions de valors en proporció al seu valor.



Pics diferents
Figura 6.6

6.6.1 Mètodes de nínxols per A.G.

Un gran nombre de mètodes han estat implementats per induir formacions de nínxols en un A.G.

La disertació de Cavicchio [10] fou un dels primers estudis en tractar d'induir un comportament de nichol en un A.G. Va introduir un mecanisme anomenat preselecció. En aquest esquema un descendent substitueix al seu parent si el valor de la funció de cost excedeix la del seu parent. En aquest sentit es manté la diversitat de la població perquè els strings tendeixen a reemplaçar strings similar a si mateixos (o als seus parents). Cavicchio reclamava mantenir més poblacions diverses en un nombre de simulacions amb una mida de la població petita (n=20).

De Jong [15] va generalitzar la preselecció de Cavicchio en un esquema que va anomenar "crowding". En el crowding de De Jong s'utilitza una població solapada per sobre, on els individus substitueixen als strings existents d'acord a les seves semblances. Un individu es compara a una subpoblació aleatòria de factors d'encreuament (CF). L'individu amb més alta semblança (en la base de similitut bit a bit) és reemplaçat per un nou string.

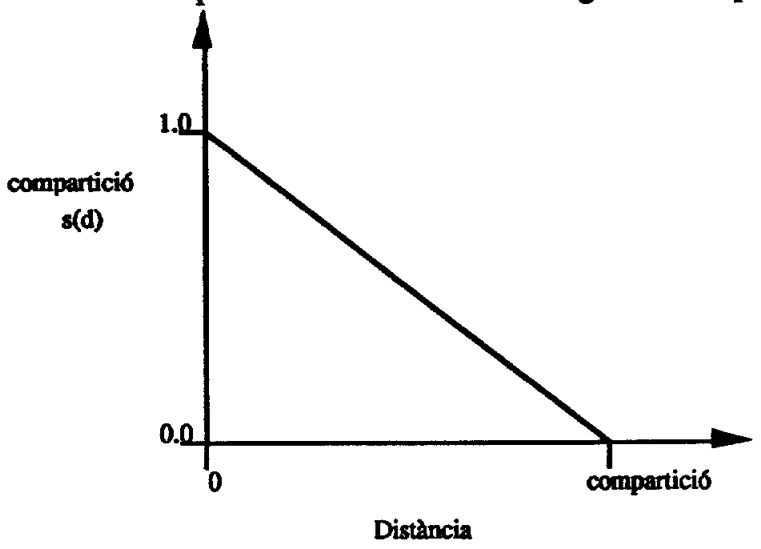
Al principi de la simulació, això porta a una selecció al.leatòria de reemplaçaments perquè és molt probable que tots els individus siguin diferents. Quant la simulació progressa, i més i més individus a la població són similars a d'altres (una o més espècies han aconseguit una emprenta substancial a la població) el reemplaçament d'individus per individus similars tendeix a mantenir la diversitat dins de la població i tenim lloc per a dos o més espècies. De Jong ha tingut èxit amb l'esquema d'encreuament amb funcions multimodals quant ha utilitzat factors d'encreuament de CF=2 i CF=3.

L'exploració més directe de la teoria biològica dels nínxols en el context dels A.G. té lloc a la dissertació de Perry [35]. En aquest treball defineix un mapeix de genotips a fenotips, un entorn de múltiples fonts, i una entitat especial anomenada esquema extern. Els esquemes externs són esquemes de semblances per caracteritzar als membres de les espècies. Malauradament la intervenció requerida d'un agent estrany limita l'us pràctic d'aquesta tècnica en l'A.G.

Grosso [21] també va mantenir una orientació biològica al seu estudi de formacions de subpoblacions explícites i operadors de

migració. Ja que en aquest estudi es van utilitzar funcions objectiu multiplicatives i heteròtiques (heterozigòtiques millor que homozigòtiques), aquest estudi suggereix que l'imposició d'una geografia en una recerca genètica pot ser una forma útil de permetre la formació de subpoblacions diverses. Es necessiten més estudis per determinar com fer això en els problemes de recerca artificials.

Un esquema pràctic que utilitza directament la metàfora de compartir recursos per induir nínxols i espècies és detallat per Goldberg i Richardson [19]. En aquest esquema, es defineix una funció de compartició de cada string a la població. Per veure això en paraules, considerem les funcions de prova d'una dimensió de les figures 6.5 i 6.6 i la funció de compartició mostrada a la figura 6.7. Per un individu donat, el grau de compartició es determina sumant els valors de compartició contribuïts per tots els demés strings de la població.



$d_{ij} = \| x_i - x_j \|$

Figura 6.7

Els strings propers a un individu necessiten un alt valor de compartició (proper a u), i els strings llunyans a l'individu necessiten un grau molt petit de compartició (proper a zero). Ja que l'individu està molt proper a si mateix, la seva funció de compartició és u. Després d'acumular el nombre total de comparticions d'aquesta manera, el valor de la funció de cost es calcula prenent el valor de la funció potencial (el valor sense compartir) i el dividim pel nombre acumulat de comparticions:

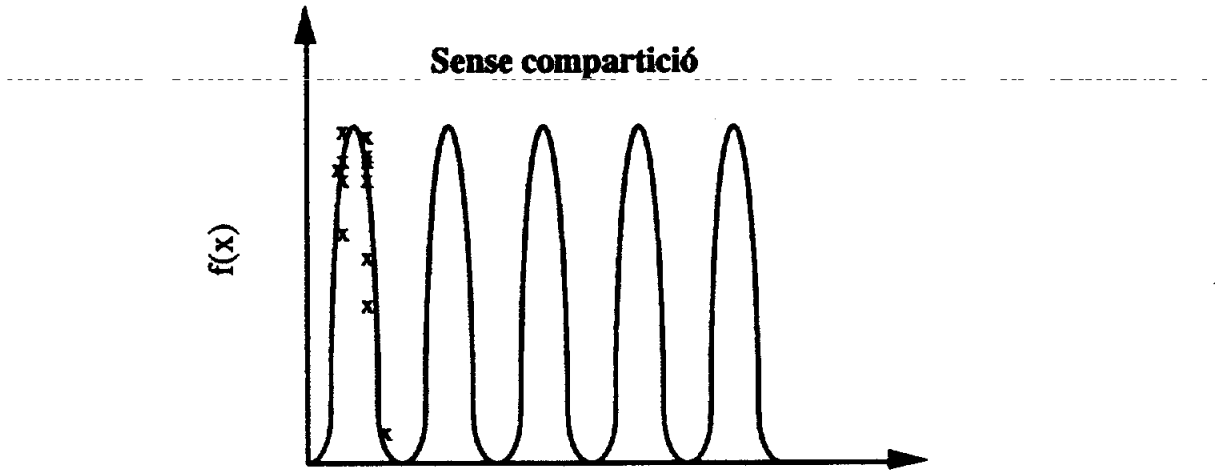
$$f_S(x_i)=\frac{f(x_i)}{\sum_{j=1}^{n} S(d(x_i,x_j))}$$

D'aquesta forma, quant molts individus estan en el mateix veïnatge contribueixen l'un en contra de l'altre. Com a resultat aquest mecanisme limita el creixement incontrolat d'espècies particulars dins d'una població.

Posteriorment Richardson i Goldberg [19] han suggerit una extensió multidimensional de la funció de compartició per permetre utilitzar-la en problemes d'optimització multidimensionals.

Els resultats de l'aplicació d'aquesta funció de compartició podem veure-ho a les figures següents:

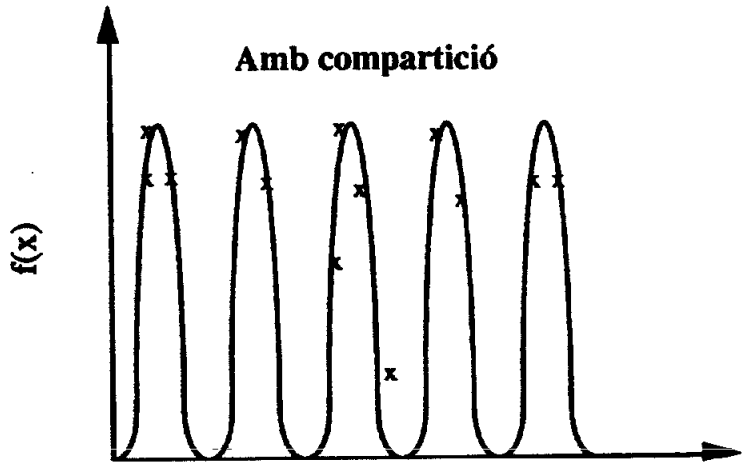
En el cas sense compartició, per una funció de cinc pics de la mateixa alçada, els individus es dirigueixen a uns dels pics a l'atzar.



Pics Iguals

Figura 6.8

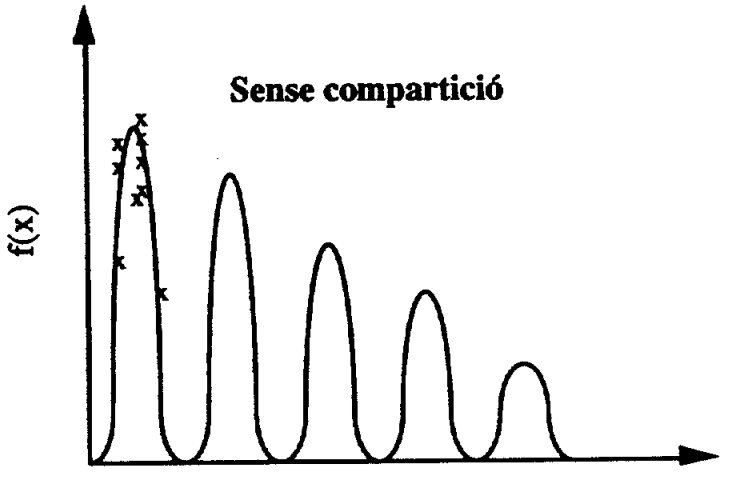
Quant s'utilitza una funció de compartició els individus de la població es distribueixen per igual a tots els pics, que és el que esperàvem.



Pics Iguals

Figura 6.9

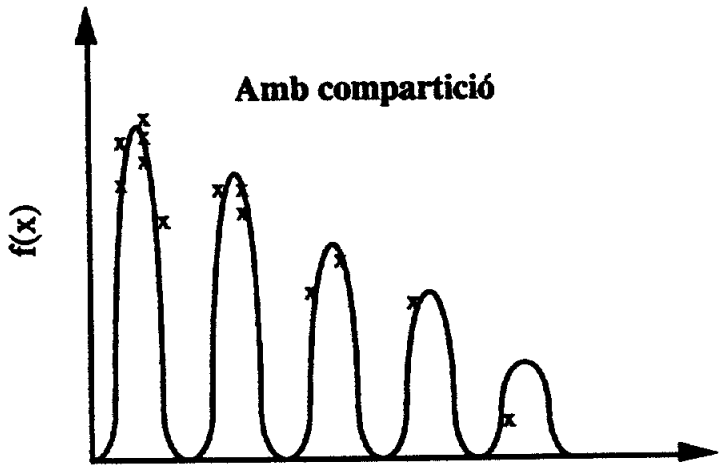
Si apliquem el nostre A.G. original, sense compartició a la funció de la figura 6.6 es distribueixen tots els valors a prop del pic més alt com es pot véure en la figura 6.10.



Pics diferents

Figura 6.10

En canvi quant s'utilitza una funció de compartició els individus es distribuéixen a prop de tots els pics i en un nombre proporcional a la alçada d'aquest pic.



Pics diferents

Figura 6.11

Com a conclusió podem dir que la compartició i la formació de nínxols ens serà útil sempre que volguem conéixer diferents valors de solucions dins d'un problema i no només un de sol.

7 Obtenció de bons codis utilitzant un A.G.

Després d'haver explicat el funcionament bàsic d'un A.G. simple de tres operants: selecció, encreuament i mutació i d'haver justificat matemàticament el seu bon funcionament, hem tractat altres operants genètics que en general no seran aplicables a tots els casos.

Ara tractarem l'aplicació d'un A.G. simple a l'obtenció de codis. Posteriorment veurem les millores, en quant a operants genètics o en quant a la funció de cost que podem realitzar.

Un codi, tal com hem indicat anteriorment, és un conjunt de M paraules formades per n elements agafats d'un alfabet de q valors (els elements de les paraules varien entre 0 i q-1):

$$C = (X_1, X_2, , X_M)$$

Treballarem amb dos classes principals de codis. Per una part tractarem de trobar bons codis de pes constant, amb un alfabet binari però amb paraules de pes constant, i per altre tractarem de trobar codis de cobertura.

Aquests últims podriem classificar-los com una família de codis, ja que no tractarem un únic tipus de codis sinó varis: codis de cobertura binaris, codis de cobertura q-aris i codis de cobertura mixtes del tipus $F_4^1F_2^b$

Aquests, com hem explicat en capítols anteriors, són uns codis formats per paraules en el que el seu primer element està agafat d'un alfabet 4-ari i la resta dels elements de la paraula ho estan d'un alfabet binari.

Aquesta varietat de codificacions provocarà, com veurem, una major complexitat en la realització dels operants genètics que els tractaran.

La simulació de l'A.G. s'ha realitzat mitjançant un programa en C en un entorn UNIX en els ordenadors SUN del departament de Matemàtica Aplicada i Telemàtica de l'E.T.S.E.T.B. No explicarem el funcionament detallat del programa (ja que la font ja és comentada). Si que explicarem

els algorismes dels operants realitzats especialment per a aquesta aplicació.

Encara que les aplicacions de l'A.G. a l'obtenció de codis de pes constant i a codis de cobertura tenen molts punts en comú, sobretot en la filosofia d'obtenció de bons codis, tractarem cada una d'elles per separat, ja que tenen elements que les diferencien clarament.

7.1 Aplicació d'un A.G. a l'obtenció de C.W.C.

El nostre objectiu en aquest problema es trobar codis, que per a una certa distància mínima d_{min} i una longitud de la paraula donada, tinguin el major nombre possible de paraules. Es tracta de trobar les ja anomenades constants A(n,d,w) que ens indiquen el nombre màxim de paraules que per una longitud de la paraula n, una d_{min} de d i un pes de la paraula constant w pot tenir aquest codi.

Aquesta, de tota manera, no serà una recerca a cegues en busca de qualsevol codi ja que partim de resultats trobats per altres investigadors. En concret nosaltres hem tractat d'igualar i, quant ha sigut possible, superar els resultats trobats pel que fa a C.W.C per El Gamal *et al.* [1]. En aquest article va trobar noves cotes teòriques de C.W.C. mitjançant un altre mètode de recerca d'extrems, la Recuita Simulada. Mitjançant aquest mètode va trobar nous codis que superaven, quant al nombre de paraules, els límits trobats anteriorment establint nous récords.

Abans d'explicar l'aplicació efectiva de l'A.G. hem d'indicar que aquest intent de trobar bons resultats és un arma de doble tall. Per una part tenim un bon sistema per comparar els nostres resultats i per veure si estem realitzant bé les coses. Però per altra tenim que, degut a que utilitzem mètodes probabilístics, pot ser més difícil tornar a trobar un resultat concret resultant de moltes simulacions. Podem dir que la probabilitat de tornar a trobar un codi en concret és molt més petita que el de buscar-ne diversos de diferents i trobar-ne un.

Per reduir aquest efecte hem intentat, si el temps de simulació ho ha permès, trobar tots els nous codis donats en el seu article.

Bé, afrontem-nos al problema cara a cara. Com podem trobar un codi amb un A.G. El primer que hem de determinar són les característiques del codi a trobar. O sigui en el nostre cas la longitud de la paraula, el nombre de paraules del que està format i el pes de la paraula.

Per suposat, qualsevol conjunt de paraules formades a l'atzar amb aquestes característiques no estarà a la distància mínima que nosaltres desitgem pel nostre problema. És aquí on entra en joc la funció de cost. Si codifiquem cada string del nostre A.G. com a un possible codi hem de fer-lo evolucionar (ja veurem com) fins arribar a trobar un codi amb les característiques definides i amb la distància mínima desitjada. En el nostre cas la simulació acabarà quant trobem un codi, un string de la població, que assoleixi la distància mínima que volíem.

Una vegada trobat aquest codi amb un nombre M de paraules tornarem a provar de trobar un nou codi igual a l'anterior però amb un nombre de paraules superior. Normalment aquest nombre serà de M+1 paraules excepte en els casos en que veiem que el codi ha estat trobat amb molta facilitat. En aquest cas triarem un nombre més alt de paraules per no perdre el temps. La recerca del codi acabarà quan el nostre A.G. sigui incapaç de trobar un codi a la distància mínima dessitjada amb el nombre de paraules indicat.

Hem de solucionar dos problemes principalment per poder arribar a trobar una solució. El primer problema al que hem de trobar solució és el de la funció de cost. Quina funció de cost hem d'utilitzar de forma que ens porti directament cap a codis de distància mínima grans?. L'elecció d'una bona funció de cost sol ser la majoria de vegades l'aspecte més crític perquè l'A.G. funcioni correctament, ja que la taxa de selecció depèn directament d'aquest valor. També hem vist que l'escalat de la funció pot arreglar, en part, una funció de cost dolenta o no massa bona.

El segon aspecte que hem de solucionar és fer que l'A.G. funcioni bé. Això és que els nostres operants genètics treballin adequadament amb aquest tipus de dades, (codis formats per paraules de pes constant). Com veurem després un encreuament entre paraules de pes constant no dona, en general, dos paraules del mateix pes que les dels seus pares. Tampoc la mutació clàssica, de bit, podrà ser utilitzada en el nostre cas, ja que el canvi d'un bit d'una paraula binària provoca l'augment o la disminució del pes d'aquesta paraula.

7.1.1 Funció de cost en C.W.C.

En un principi podríem pensar en utilitzar com a funció de cost la pròpia distància mínima del codi. Ja que aquest valor ha de ser creixent, si l'A.G. evolucionés, ens portaria cap a bons valors de distància mínima. El problema és que amb aquest valor de la funció de cost no seria possible que l'A.G. "trobés el camí" cap a valors de d_{min} alts, ja que aquesta funció només pren uns pocs valors per la totalitat de l'ampli espai a recórrer.

Aquest problema ja va ser apreciat per El Gamal *et al.* en la seva recerca de Codis de pes constant amb la Recuita Simulada. Ell va utilitzar, no aquesta distància mínima, sinó una valoració de totes les distàncies existents entre les paraules del codi. En concret va utilitzar la fòrmula:

$$f_{cost} = \sum d^{-2}$$

On d és la distància de Hamming entre dues paraules qualsevol del codi.

Hem d'esperar que per valors de distància entre paraules altres arribem a valors de d_{min} alts. Això és lògic ja que si totes les paraules es van distanciant una de les altres degut a aquesta valoració global de la funció de cost arribarà un moment en el que la distància més petita entre qualsevol d'elles arribi a igualar la nostre desitjada d_{min}. Moment en el qual aturarem la simulació.

Podrem utilitzar, doncs, directament aquesta funció amb el que sabem van trobar bons resultats El Gamal *et al.*?. Doncs no. La causa és que S.A. progressa de forma diferent al nostre A.G. S.A. progressa cap a estat de mínima energia, i per tant minimitza el valor de la funció de cost. Per contra l'A.G. implementat per nosaltres tracta de maximitzar-la.

Com no podíem treballar amb la funció directament vam pensar en dues alternatives lògiques. La primera és fer que la nostre funció sigui:

$$f_{cost}(A.G) = \frac{1}{f_{cost}(S.A)}$$

I la segona va ser :

$$f_{cost}(A.G) = K - f_{cost}(S.A)$$

Finalment vam adoptar la segona per una causa principalment, es conservava la forma de la funció. Es conserva la diferència relativa entre els valors de la funció de cost que prenen dos punts qualsevol, mentre que en el primer cas no. Si ens trobem en el final de la simulació podem trobar un valor de f(S.A) baix i al fer l'inversa se'ns disparia el seu valor molt amunt. Això provocaria l'aparició d'un superindividu que empitjoraria el comportament del nostre A.G.

El segon aspecte a determinar, una vegada decidit la forma de la funció, és el valor de K. Aquest valor és important ja que per valors de K alts tindrem una funció quasi plana pel nostre A.G. i aquest no progressarà adequadament, al no tenir una competència suficient entre els individus. Per altra banda per valors de K baixos podríem trobar-nos amb valors negatius d'aquesta funció. Els valors negatius no estan permesos en un A.G. amb una selecció del tipus de la ruleta de la sort. Perquè no ho podem permetre?. Ho podem veure fàcilment si tenim en compte que la probabilitat de que un individu sigui seleccionat és:

$$P_{sel} = \frac{f(x_j)}{\sum_{j=1}^{n} f(x_j)}$$

Si tenim algun valor de $f(x_i)$ negatiu la probabilitat de selecció no tindria cap sentit i el nostre A.G. funcionaria malament, o no funcionaria de cap forma segons la implementació algorísmica que haguéssim utilitzat de la ruleta de la sort.

Per tant l'ideal seria considerar un valor de K que ens garantitzés un valor de f() positius per qualsevol cas. Considerem com a exemple un codi de 18 paraules de longitud 23, de pes contant 7 i de d_{min} desitjada de 10.

Calculem en primer lloc el nombre de valoracions que haurem de realitzar de la distància entre paraules. Si tenim 23 paraules començarem veient la distància de la paraula 1 a la 2, de la 1 a la 3 i així

continuarem fins arribar a calcular la distància entre les paraules 1 i 23, a partir de la qual continuarem veient la distància entre les paraules 2 i 3, entre les paraules 2 i 4 etc. L'última distància a considerar lògicament serà la distància entre les paraules 22 i 23. El nombre total de distàncies calculades serà doncs: 22+21+ . . . +1. En aquest cas tindrem 253 valoracions de distàncies entre paraules. En un cas general tindrem:

$$(n-1) + (n-2) + \ldots + 1 = \frac{n\cdot(n-1)}{2}$$

Per tant hem de fer que:

$$f = K - \frac{n\cdot(n-1)}{2}\frac{1}{d^2} \geq 0$$

En el cas més estricte, el cas en que d=0 en tots els casos, o el que és el mateix si ens trobem amb un codi format per un conjunt de paraules iguals, hauriem de fer K= ∞.

Per solucionar aquest problema ens dirigim a la pràctica. Hem vist empíricament que si generem un codi a l'atzar amb aquesta d_{min} desitjada de 10 trobem normalment codis amb una d_{min} de 4 quant aquests són generats a l'atzar. Si prenem el cas més restrictiu en el que totes les distàncies entre paraules són igual a la distància mínima trobem un valor de k de la següent forma:

$$f = K - \frac{n\cdot(n-1)}{2}\frac{1}{d^2} \geq 0 \Rightarrow K \geq \frac{n\cdot(n-1)}{2\cdot 4^2}$$

Aquest és un valor suficientment segur per dos raons principalment. Per una part tenim que si la fita mínima real és de 4 el sumatori de distàncies serà en realitat molt més baix, i per altra part tenim que aquesta és la distància mínima d'un codi generat a l'atzar, però en quant l'A.G. generi un parell de generacions tindrem valors molt més alts.

De tota manera i tenint en compte la llei de Murphy ens hem previngut contra qualsevol cas estrany i igualem el valor de la funció de

cost a zero si pren un valor negetiu. Això de tota manera no succeirà pràcticament mai, tal com hem vist.

En general, i tenint en compte que tots els codis de pes constant que calcularem tindran una d_{min} desitjable de 10, hem considerat que en tots els casos la d_{min} del pitjor codi tindrà un valor de 4 amb el qual hem utilitzat un valor de k estimat per la següent fórmula:

$$K \geq \frac{n \cdot (n-1)}{3\,2}$$

Encara que hem tingut molta cura en l'elecció d'aquesta funció, haurem d'utilitzar un escalat de la funció perquè l'A.G. evolucioni adequadament i amb rapidesa. L'estudi dels possibles escalats utilitzats es veurà posteriorment.

Una vegada utilitzada aquesta funció de cost amb un escalat adequat vam observar que l'A.G. progressava adequadament i amb rapidesa, però no arribava cap a valors de d_{min} desitjats.

Fent abans algunes comprovacions prèvies de que l'A.G. funcionés correctament vam trobar una dada interessant. Comparant els nostres resultats amb els resultats trobats per El Gamal *et al.* vam veure que el nostre A.G. trobava codis de funció de cost amb un valor superior al dels codis trobats per El Gamal *et al.*, però el nostre codi tenia una distància mínima inferior a la del seu codi. Quina podia ser la causa d'això?. En primer lloc no podem culpar el nostre A.G. de trobar codis amb millor funció de cost, ja que aquesta és la seva feina. La causa d'això és que utilitzàvem la funció de cost:

$$f = K - \sum d^{-2}$$

Amb aquesta fórmula podem arribar a codis que tinguin moltes paraules a una d=14, per exemple, i unes poques a una distància d=8 (recordem que la d_{min} desitjada era de 10). Aquest codi pot tenir un valor de la funció de cost major que un codi en el que totes les distàncies entre les paraules que el formen sigui de d=10 i per tant tingui una d_{min} =10. És un problema de distribució de distàncies. Aquesta funció afavoreix l'aparició de d altes més que de distàncies més grans a d_{min}.

Per resoldre aquest problema hem fet una modificació a aquesta funció:

$$f = K - \sum g(d)^{-2} \quad \text{on } g(d) = \begin{cases} d \; ; \; d < 10 \\ 10 \; ; \; d \geq 10 \end{cases}$$

Amb el qual assegurem que el nostre màxim de la funció es correspongui amb un valor de d_{min} desitjat. Si probabilísticament és difícil trobar un valor màxim alt, més difícil és si aquest valor l'hem de trobar pel camí cap al màxim.

Llavors, com van aconseguir El Gamal *et al.* els seus bons resultats?. Hi han dues explicacions possibles. La primera és que fes una modificació a la seva funció de cost o bé que el seu èxit es degui a l'alt nombre d'execucions del seu programa que va realitzar.

Una vegada explicat com hem elegit la funció de cost passem a descriure la realització dels operants de l'A.G. capaços de portar al nostre problema cap a bones solucions.

7.1.2 Operants genètics pels C.W.C.

Dels tres operants bàsics en un A.G.: selecció, encreuament i mutació només el primer pot ser utilitzat sens cap limitació, una vegada és coneguda la funció de cost. L'encreuament i la mutació no poden treballar adequadament amb el nostre particular tipus de dades: les paraules de pes constant.

Nosaltres volem un encreuament i una mutació que ens conservi el pes de les paraules del codi. Posem un exemple per veure que això no s'acompleix. Considerem les paraules A i B de longitud 10 i pes 4. Escollim a l'atzar el lloc 4 per creuar-les. Indiquem el lloc de tall pel símbol " | " :

A = 0111|000100
B = 1001|010010

Al creuar-les ens trobarem amb els següents descendents A' i B':

A' = 01111010010
B' = 10011000100

Aquestes paraules tenen pesos de 5 i 3 respectivament. Igualment succeeix amb la mutació, si mutem el setè bit, per exemple, del individu A obtindrem el següent individu:

A' = 0111001100

que té un pes de 5. Podríem dir que l'encreuament té com a propietat el redistribuïr els pesos entre els individus creuats i la mutació el d'augmentar o disminuir en una unitat el pes de la paraula mutada a cada mutació.

Aquest problema ens obliga a trobar nous sistemes d'encreuament i de mutació que conservin el pes dels individus.

7.1.3 Encreuament de C.W.C.

Primer recordem una dada important. Nosaltres utilitzem com a individus codis formats per paraules de pes constant. Els nostres codis tindran un nombre de paraules grans, quinze com a mínim. Aquest alt nombre de paraules ens ha ajudat en part a la confecció de fins a tres tipus diferents d'encreuament.

1. Encreuament paraula a paraula.
2. Encreuament individu a individu.
3. Encreuament codi a codi.

En l'encreuament paraula a paraula considerem al nostre individu, que en realitat és una llista de paraules de pes constant, com un conjunt de paraules i creuem les paraules una a una. La primera paraula del primer individu és creuada amb la primera paraula del segon individu per donar les primeres paraules dels codis fills creuats. D'aquesta manera es creuen totes les paraules que formen els codi en parelles. Aquest encreuament entre paraules no pot ser un encreuament normal

ja que això provocaria, tal com ja hem vist, una variació en el pes de les paraules del codi. Aquest encreuament entre paraules serà explicat posteriorment.

Ja que hem de fer un total de n encreuaments de paraules individuals per creuar un codi (sent n el nombre de paraules que el formen). Podem realitzar dos variants d'aquest encreuament.

En una primera variant que en podríem anomenar de "punt de tall constant", el mateix lloc de encreuament és seleccionat per totes les paraules del codi. En una segona variant, anomenada de "punt de tall variable" aquest punt de tall és independent, i en general diferent, per cada parell de paraules a creuar. Per aclarir aquest concepte observem els dos codis simbòlics A i B formats per cinc paraules de pes constant. L'encreuament és realitzaria de la següent forma en el cas de l'encreuament de "punt de tall constant":

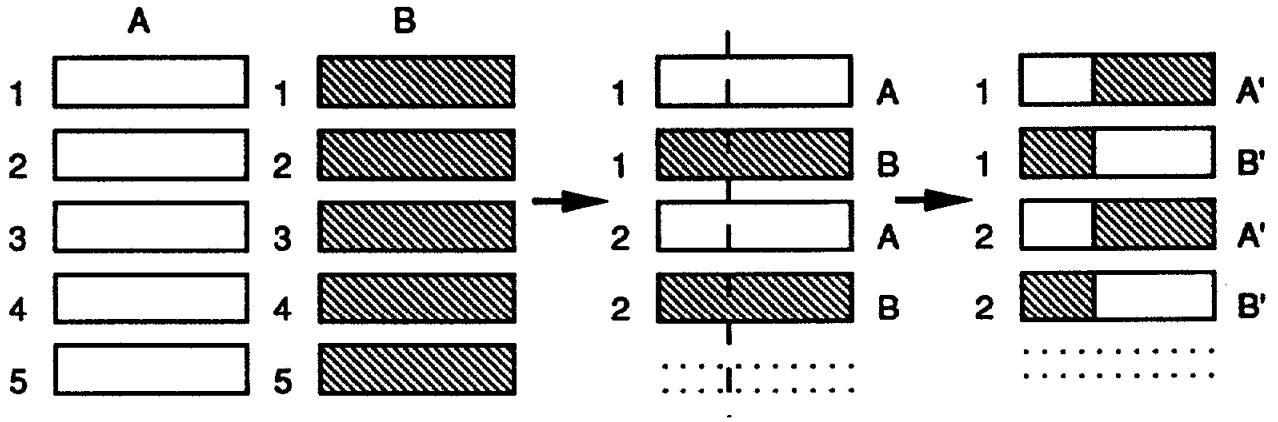


Figura 7.1

En el cas de l'encreuament amb punt de tall variable podríem simbolitzar l'encreuament de la següent forma:

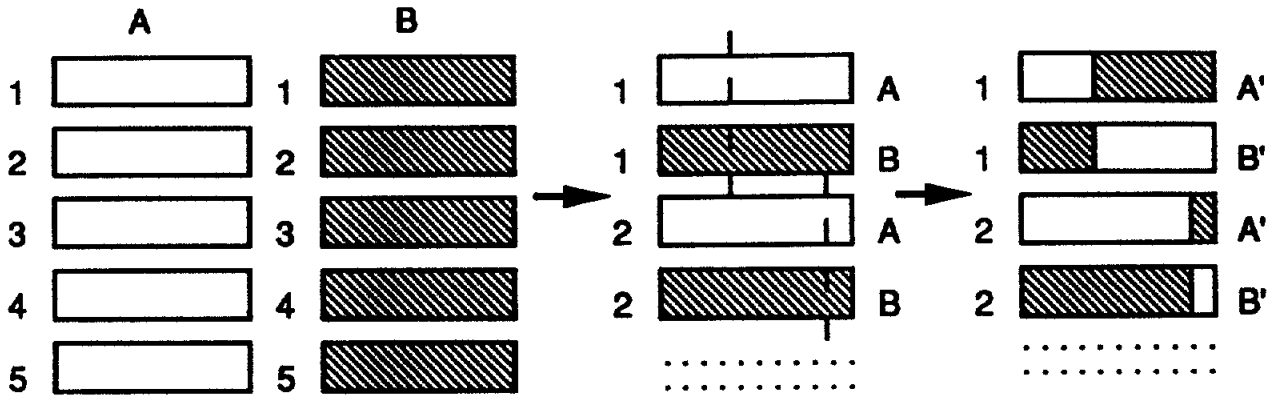


Figura 7.2

Com es pot apreciar l'encreuament només s'ha indicat simbòlicament ja que no hem indicat encara com fer-ho. Algorísmicament l'única diferència entre els dos tipus d'encreuament és que al segon s'han de realitzar més operacions de trobar aleatòriament posicions de tall.

La independència entre les paraules ens fa veure que en principi no hi hauria d'haver moltes diferències en quant a funcionament efectiu de les dues variants.

El següent encreuament que hem creat es l'anomenat "individu a individu". En aquest encreuament el codi és creat, en principi, com si fos un tot sense parts. Si considerem l'individu com una llista única de nombres binaris, l'únic que hem de fer per creuar-lo és trobar un punt de tall en el codi i creuar-lo de la forma clàssica. Si ho fem així ens trobarem que és molt probable que una paraula del codi canviï de pes. Quina és la raó d'això?. Per veure quina pot ser la causa posem un exemple. Creuem els codis simbòlics A i B formats per cinc paraules de pes constant. Com hem indicat seleccionem un únic punt de tall d'entre les $n \cdot L$ possibles (sent n el nombre de paraules i L la longitud de la paraula).

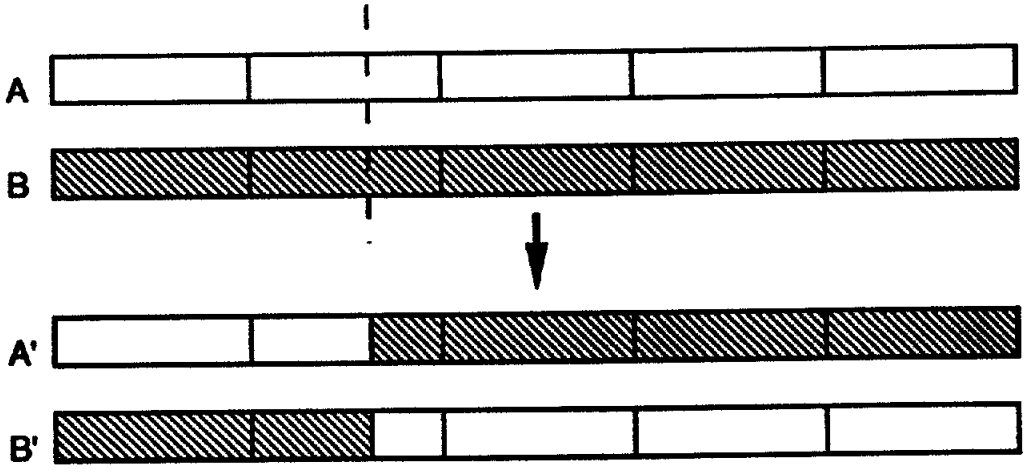


Figura 7.3

En aquest exemple hem pogut veure com la segona paraula d'aquest codi hipotètic és tallada pel mig i creuada amb altra paraula de pes constant. Com hem pogut veure, això provoca que hi hagin moltes possibilitats de fer un mal encreuament. Per solucionar aquest problema el que fem és trobar la paraula on tindrà lloc el tall i el lloc on aquest caurà dins de la paraula. una vegada ho sabem el que fem es intercanviar les paraules posteriors a aquesta paraula en el codi i creuar la paraula on cau el tall d'una manera adequada que conservi el pes. Simbòlicament ho podríem representar pel següent esquema.

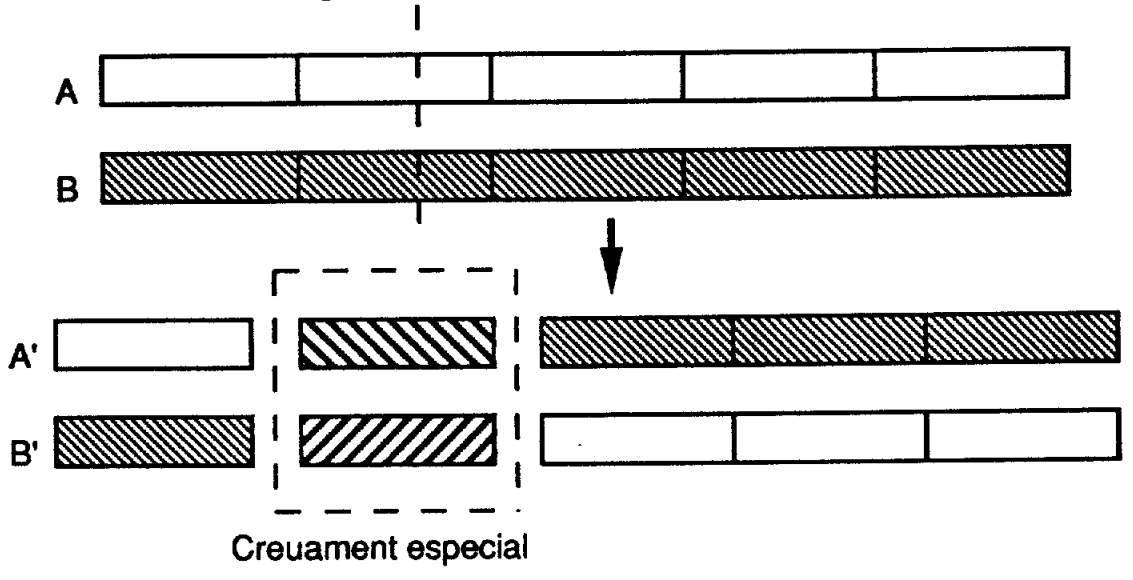


Figura 7.4

Aquest encreuament seria el més proper al clàssic, ja que tot l'individu és considerat quasi com a una única entitat sense importar-nos com està fet. L'única diferència està en l'encreuament paraula a paraula no clàssic efectuat a les paraules on cau el tall.

Observant aquest tipus de encreuament veiem que per codis amb un gran nombre de paraules es podria realitzar un encreuament molt més senzill i que no es diferenciaria massa de l'anterior encreuament. És el que anomenem encreuament "codi a codi" i consisteix en creuar individus, no com si estiguessin format per bits sinó com si estiguessin formats per paraules indivisibles. Això a la pràctica limita el nombre de punts de tall als existents entre les paraules. En aquest encreuament no es pot trencar cap paraula amb el que s'impedeix cap encreuament incorrecte. Simbòlicament ho podem expressar amb el següent esquema.

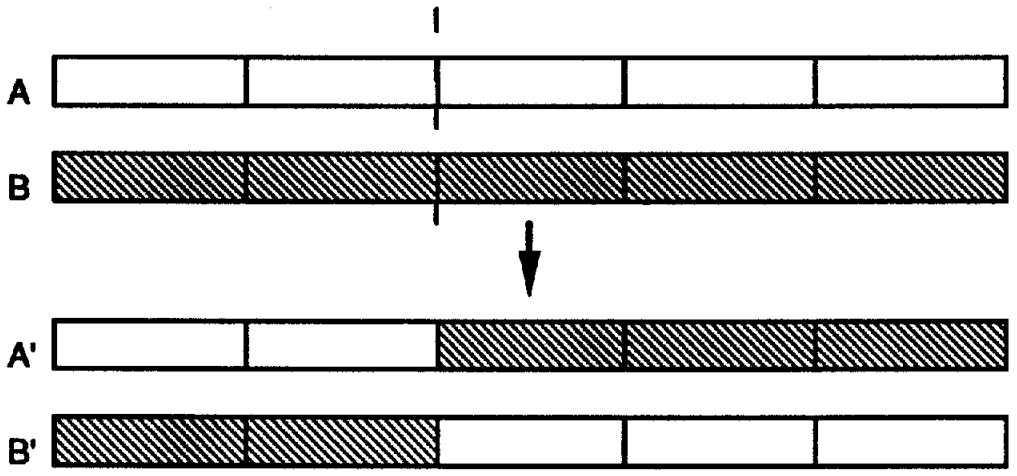


Figura 7.5

Ho podrem veure més clarament quant expliqui com es pot realitzar un encreuament entre paraules de pes constant, de forma que aquesta propietat es conservi, però ja es pot apreciar que la diferència entre els individus resultants d'aquest operant i de l'anterior serà d'uns pocs bits. Diferència insignificant quant, com en el nostre cas, el nombre de paraules és elevat.

Abans de comentar les diferències observades a la simulació pels tres mètodes emprats explicarem com es pot realitzar aquest encreuament paraula a paraula entre paraules de pes constant.

7.1.4 Encreuament de paraules de pes constant

Podem simbolitzar l'encreuament entre paraules de pes constant amb el següent algorisme. Suposem que volem creuar dues paraules A i B de longitud l i de pes p.

a. Trobem un nombre a l'atzar entre 1 i p-1 que ens indicarà el nombre d'uns a creuar. una vegada fet això trobem quina és la posició que ocupa aquest 1 al string A.

b. Des de el primer 1 del string A fins a aquest últim.

b.1 Mirem si la posició que ocupa aquest 1 està ocupat per un 0 al string B.

b.1.1 Si està ocupat per un 0 busquem el primer 1 del string B amb una posició tal que aquesta al string A estigui ocupada per un 0 (per exemple si a la posició 3 del string B tenim un 1 mirem si a la posició 3 del string A tenim un 0).

b.1.2 una vegada trobats tant el 1 del string A acompanyat del 0 del string B en la seva mateixa posició, com el 1 del string B acompanyat del 0 al string A en igual posició sobre el string els intercanviem.

b.2 Si la posició està ocupada per un 1, els deixem igual ja que un intercanvi en aquestes posicions no representaria cap canvi.

Aquest procediment pot semblar una mica resseguit però l'aclarirem amb un exemple. Creuem els dos individus següents de longitud 10 i pes 5.

A = 0011011010
B = 0101001101

Primer trobem el nombre d'uns a creuar entre 1 i 4. En aquest cas trobem que el valor generat aleatòriament és 3.

El primer 1 que trobem al string A per creuar és el de la posició 3. A la posició 3 del string B trobem un 0 per el que hem de buscar el primer 1 del string B amb un 0 homòleg al string A. Veiem que es troba a la posició 2.

Per tant els bits 2 i 3 dels strings creuats A' i B' estaran canviats de valor. El bit 2 del string A' serà un 1 i el bit 3 del mateix string serà un 0. El contrari passa pel string B'.

Continuem amb el següent 1. El trobem a la posició 4. Com veiem que té un 1 a la mateixa posició del string B no fem res i pasem al següent i últim 1 a creuar.

El trobem a la posició 6 i té un 0 homòleg al string B. busquem el següent 1 al string B amb un 0 homòleg al string A. El trobem a la posició vuitena i veiem que els bits que canviaran seran el sisè i el vuitè dels strings A' i B' respecte dels strings pares A i B. Podem veure aquesta operació gràficament de la següent forma:

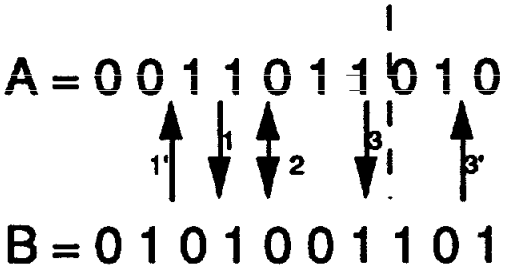


Figura 7.6

Amb el que obtenim:

A' = 0101001110
B' = 0011011001

Com es pot apreciar aquest encreuament si conserva el pes de la paraula. Un fet indicatiu de que les coses funcionen bé és que quant dos bits són iguals no canvien amb el que es conserva les semblances entre els strings. Es pot demostrar que aquest mètode funcionarà sempre bé, ja que com es pot veure fàcilment ha d'haver igual nombre d'uns al string A amb un 0 homòleg al string B com uns al string B amb un 1 homòleg al string A.

Aquest és un nou tipus d'encreuament no existent fins ara i que s'ha creat específicament per aquest problema. De tota manera podria ser utilitzat en altres problemes juntament amb una mutació que

conservi el pes, per resoldre problemes que puguin ser representats per paraules binàries de pes constant.

Aquest podria ser el cas de la col.locació de n dames en un taulell d'escacs de PxP posicions. Podríem representar aquest problema amb un string de longitud PxP i pes n. Cada 1 indicaria la posició d'una dama. La funció de cost hauria de ser funció del nombre d'escacs que es fessin les n dames per cada posició particular d'aquestes.

Ara passarem a descriure el funcionament de l'operant mutació, que és molt senzill en comparació amb l'operant d'encreuament.

7.1.4 Mutació de C.W.C.

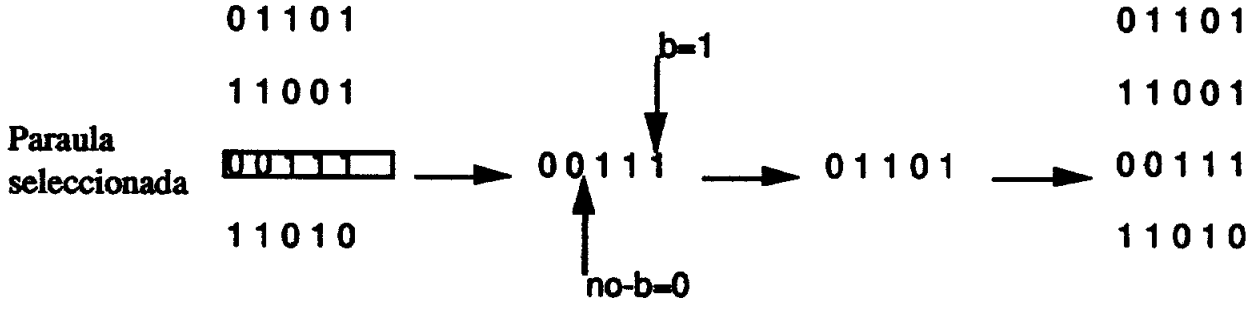
La mutació que hem estudiat anteriorment és l'anomenada mutació de bit. Amb aquest operant cadascun dels bits components d'un string podien amb certa possibilitat, generalment baixa, canviar el seu valor.

Degut a aquesta baixa taxa de mutació es sol modificar lleugerament aquest operant. Normalment la taxa de mutació utilitzada a la majoria de problemes no arriba a una mutació per individu independentment del problema i del tipus de codificació escollida. És per aquesta raó que podem variar lleugerament el nostre esquema de mutació. El que es pot fer, alternativament, és primer seleccionar un bit a l'atzar dins de l'individu i després amb certa probabilitat, molt més alta que l'anterior, decidir si ha de ser o no mutat.

Aquesta idea és el que ens ha ajudat a poder efectuar mutacions de paraules de pes constant que ens conservessin el pes.

El procediment que ens portaria a efectuar una mutació d'un codi de pes constant seria el següent: com a primer pas elegim a l'atzar quina de les n paraules del codi pot ser mutada. Una vegada trobada elegim a l'atzar una posició dins de la paraula. Suposant que el bit corresponent a aquesta posició tingui un valor b (un 1 o un 0) seleccionem una altra posició dins de la paraula que tingui un valor de no-b (Si b és un 1 no-b serà un 0 i a l'inrevés). Si aquesta segona posició escollida té el mateix valor b ho tornem a intentar fins que trobem una posició ocupada amb un valor no-b. Una vegada trobats fem que el string mutat tingui un no-b on abans tenia un b, i un b a la posició que contenia un no-b.

Per exemple si tenim el següent codi format per quatre paraules de longitud cinc i pes tres:



Primer seleccionem quina paraula de les quatre canviarem i trobem que és la tercera. Seleccionem una posició a l'atzar dins de la tercera paraula d'entre les cinc possibles. Trobem la quarta que està ocupada per un 1. Llavors seleccionem una altra posició dins de la paraula i resulta ser la cinquena que està ocupada per un altra 1. Ho tornem a provar. Aquesta vegada trobem la posició dos que està ocupada per un 0. Canviem aquests valors al string mutat i ja tenim el codi corresponent, compost de la resta de paraules sense cap variació i aquesta paraula mutada.

També vam realitzar proves amb un criteri més obert en quant a nombre de mutacions per codi. Es va provar una mutació en el que la probabilitat de mutació era per paraula i no per codi. Totes les paraules del codi tenien certa probabilitat de mutació. Els resultats van ser iguals o a vegades lleugerament inferiors als trobats amb l'esquema anterior. Per això es va aplicar aquest esquema de mutació al nostre problema.

7.1.5 Comentari de les prestacions dels operants emprats

S'han realitzat proves de cadascun d'aquests operants, tant dels tres tipus d'encreuament com els dos de mutació amb diferents probabilitats de mutació i d'encreuament.

Aquestes proves encara que no han estat exhaustives, amb l'aplicació a molts tipus de problemes diferents, si que han estat significatives en la seva aplicació al nostre problema en particular. Buscàvem ajustar el nostre sistema perquè funcionés bé més que per fer

un estudi estadístic del funcionament d'aquests operants. Els resultats que vam trobar és que el millor operant de mutació és el descrit anteriorment en el que es seleccionava una de les paraules per mutar-la. L'altre operant en el que totes les paraules podien canviar amb certa probabilitat oferia un rendiment globalment pitjor.

En quant als mètodes provats d'encreuament hem de dir que els mètodes d'encreuament codi a codi i el d'individu a individu tenien un rendiment superior al de paraula a paraula en les seves dues versions, que per altra banda donaven un resultat similar. Finalment per l'aplicació definitiva a la recerca de codis de cobertura és va utilitzar l'encreuament codi a codi ja que per les mateixes prestacions, esperades a la teoria i demostrades a la pràctica, l'encreuament individu a individu té molta més simplicitat algorísmica i rapidesa de càlcul.

L'últim que ens queda perquè el nostre sistema funcioni a ple rendiment, després d'haver trobat una funció de cost idònia i uns operants genètics que treballin apropiadament i amb eficàcia, és trobar uns paràmetres d'execució (mida de la població i probabilitats d'encreuament i de mutació òptims) i utilitzar, en el nostre cas crear, un escalat de la funció que faci que l'A.G. progressi amb rapidesa i eficiència.

Degut a que els estudis d'aquests factors són comuns tant a l'estudi de l'aplicació a codis de pes constant com de codis de cobertura, seran estudiats posteriorment. Passem ara a l'aplicació del nostre A.G. a l'obtenció de codis de cobertura.

7.2 Aplicació de l'A.G. a l'obtenció de C.C.

El nostre objectiu en aquest cas és molt semblant a l'objectiu plantejat per trobar codis de pes constant. En aquest cas el que busquem són codis que per una certa longitud de la paraula, un alfabet d'aquesta paraula donat, i un radi de cobertura R amb que cobrir l'espai que defineix el tipus de paraula del codi, tingui el mínim nombre possible de paraules. Es tracta de trobar els valors de les constants $K_q(n,R)$ on q és el nombre d'elements de que consta l'alfabet on es defineix la paraula, n és la longitud de la paraula i R és el radi de cobertura.

Com en el cas anterior no partim de 0 en la nostre recerca i ens basem en els resultats trobats per Patrick R.J.Östergard en [31], [32] i [33] en els que va trobar nous codis de cobertura mitjançant S.A. En aquests articles Östergard aconsegueix millorar diferents valors d'aquestes constants $K_q(n,R)$ trobant codis que les acompleixen mitjançant el mètode del S.A.

No es tracta d'una confrontació entre l'aplicació del S.A. i l'aplicació de l'A.G., però com en el cas anterior aquests resultats ens poden ajudar a veure si estem realitzant un bon treball. En aquest cas tenim la dificultat de que el nombre de resultats és menor i que a més estan aconsegits utilitzant tres tipus diferents de codis: codis binaris, codis q-aris i codis del tipus $F_4^1F_2^b$, no utilitzant més que uns quants codis trobats per cada tipus de codi de cobertura.

Això dificulta més la nostre recerca, ja que no només hem d'implementar tres tipus diferents d'algorismes sinó que no tenim massa base per jutjar els nostres resultats, degut a la poca quantitat de codis trobats. Per paliar aquesta situació el que hem fet és no només intentar trobar aquests codis sinó a més a més intentar trobar codis de baix nombre de paraules dels que ja es coneixen la fita teòrica.

Com podem trobar bons codis amb l'A.G.?. Primer determinarem les característiques del codi a trobar. En aquest cas, mida de la paraula, tipus de codi a trobar: binari, q-ari o bé del tipus $F_4^1F_2^b$, el nombre de paraules del que està format i la distància R a la que volem que cobreixi el seu espai.

En un principi les paraules generades no cobriran tot l'espai i tindrem paraules d'aquest espai que no estaran cobertes. La funció de cost en el nostre cas serà el nombre de paraules cobertes pel codi. una vegada l'A.G. progressi, els codis aniran cobrint tot l'espai fins que arribin, o esperem que arribin, a un punt en el que un codi de la població cobreixi la totalitat de l'espai. En aquest punt acabarà l'execució del nostre programa. El pas següent es reduir en una o en varies, si veiem que ha trobat el resultat molt fàcilment, el nombre de paraules del que consta el codi. Això ho repetirem fins que per una certa mida del codi el nostre A.G. sigui incapaç de cubrir l'espai totalment.

Com en el cas anterior començarem descrivint l'elecció de la funció de cost per després descriure els operants genètics utilitzats en aquest problema.

7.2.1 Funcions de cost en C.C.

Una gran part del temps de simulació recau en el càlcul de la funció de cost. A l'operació de selecció aquesta operació ha de realitzar-se un total de p vegades, sent p la mida de la població.

En el nostre cas la funció en si no és complicada, però si ho és el seu càlcul per el que haurem de fer les coses bé per disminuir el temps de càlcul de l'algorisme.

Per qualsevol dels diferents tipus de codis de cobertura emprats: binaris, q-aris o $F_4^1F_2^b$ tindrem la mateixa expressió per la funció de cost. Això és, el nombre de paraules que cubreix aquest codi dins del seu espai.

Un codi R-cobreix un espai quant totes les paraules de l'espai estan dins d'una distància R d'alguna paraula del codi. (sent la distància entre elles la de Hamming). Bé, doncs podríem definir el nombre de paraules que cubreix un codi a una distància R com el nombre de paraules de l'espai que estan a una distància menor o igual a R d'alguna paraula del codi.

Com poder calcular aquest nombre de paraules?. Una forma directa és generar totes les paraules de l'espai i mirar la distància a cada una de les paraules del codi. Si trobem que la distància entre aquestes paraules generades i alguna paraula del codi és menor o igual a R, la comptabilitzem com a paraula coberta pel codi. Aquest sistema de càlcul comporta un nombre d'operacions altíssim. Per un codi format per M paraules binaries de n bits hauríem de fer un total de $M \cdot 2^n$ càlculs de distància per cada individu (2^n és la mida de l'espai del codi). Això ens porta, per una mida de la població de p individus, a un total de $p \cdot M \cdot 2^n$ determinacions de distància per cada generació. Per un codi q-ari el nombre d'operacions seria $p \cdot M \cdot q^n$

Aquest és un nombre tan gran que ens pot dur a que el nostre A.G. funcioni molt lentament. Per solucionar aquest problema proposem un disseny alternatiu, que si bé és més ràpid porta aparellat un us més

gran de la memòria. El primer que fem és crear una estructura del tipus "array" d'enters que anomenem *espai de cobertura.* Aquest array d'enters tindrà una mida de q^n elements per una paraula q-aria de n dígits. Aquest array ens representarà l'espai que hem de cobrir.

L'algorisme que calcula la cobertura del codi a una distància R és el següent:

a. Inicialitzem totes les posicions de l'array espai de cobertura a 0. També inicialitzem a 0 el contador de paraules cobertes. Això ens indicarà que cap paraula està coberta.

b. Des de la primera paraula del codi fins a l'última:

b.1 Generem totes les paraules que estan a una distància menor o igual a R d'aquesta paraula del codi.

b.2 Calculem el valor binari que representa aquestes paraules generades.

b.2 Mirem si la posició corresponent a aquestes paraules generades en l'array de l'espai són un 1 o un 0.

b.2.1 Si aquesta posició és un 0 voldrà dir que aquesta paraula és la primera vegada que és generada. Incrementem el contador de nombre de paraules cobertes i col.loquem el valor corresponent a aquesta posició dins de l'array de l'espai a 1.

b.2.2 Si aquesta posició és ocupada per un 1 voldrà dir que aquesta paraula ja haurà estat generada anteriorment. i continuem, sense fer cap canvi.

Es pot observar que el que hem fet és numerar totes les paraules de l'espai i marcar paulatinament les que hem anat cobrint. Cada vegada que marquem una paraula nova ho contabilitzem i així podem saber quantes portem. Per poder numerar totes les paraules l'únic que hem hagut de fer és calcular el valor decimal que representa el valor binari de la paraula.

Per un radi de cobertura R=1, el nombre de paraules de n bits que es troben a una distància menor o igual a 1 d'una altra és n+1. Amb això podem veure que el nombre total d'operacions per generació serà de l'ordre de $p \cdot M \cdot (n+1)$. Aquest valor és sensíblement inferior al cas anterior

Veiem que aquest nou càlcul de paraules covertes comporta principalment dos aspectes diferents de l'anterior. El primer és la

realització d'una generació de paraules que estiguin a una distància menor o igual a R d'un altra i el segon és la conversió de paraules q-àries a decimal per poder comprovar si aquestes han estat cobertes. Només la primera operació ens pot portar alguna complicació. Però abans fixem-nos en un altra detall que podem millorar.

Segon l'algorisme descrit anteriorment, cada vegada que volem calcular la funció de cost d'un codi inicialitzem el nostre array de q^n posicions a 0. Això és degut a que després del primer càlcul de la funció de cost, l'array de l'espai de cobertura tindrà una barreja d'uns i de 0s que indicarà el nombre de paraules cobertes pel codi anterior. Si aprofitem això adequadament podem estalviar-nos un elevat nombre d'operacions d'inicialització d'aquesta gran matriu. Per fer-ho modifiquem lleugerament el nostre algorisme de la següent manera.

b.2 Si la posició corresponent al string no és "1" ho posem a "1" i incrementem el contador de paraules covertes.

El truc és implementar aquest "1" per a cada càlcul de la funció de cost de forma que sigui diferent cada vegada. Inicialitzem la matriu de l'espai al primer càlcul de la funció de cost a 0, el valor de l'u serà 1. Al començar un segon càlcul per un altre codi tindrem aquesta matriu plena d'uns i de 0s. El que fem és assignar un valor de 2 al 1. D'aquesta manera podem utilitzar la mateixa matriu sense inicialitzar-la. En els successius càlculs de la funció de cost pels diferents codis aquest valor s'anirà incrementant. Això vindrà limitat pel tipus sencer utilitzat a la màquina on fem córrer la simulació. Per exemple, si pot representar enters fins al valor 36000 haurem d'inicialitzar la nostre matriu cada 36000 càlculs de la funció de cost. De tota manera aquest senzill truc ens ha estalviat un gran nombre d'operacions, degut a la mida de la matriu, d'escritura a la memòria.

El següent que explicarem serà com generar totes les paraules que es troben a una distància R d'una altra. Això té la seva complicació si tenim en compte que la cobertura no ha de ser, en general, clàssica sinó que pot ser una cobertura mitjançant una matriu segons el mètode descrit per Kamps i Van Lint [26]. A més a més el fet d'utilitzar codis q-aris o codis del tipus $F_4^l F_2^b$ complicarà una mica més aquesta generació.

7.2.1.1 Generació de paraules a una distancia R d'altra

Recordem que per $s \in F_q^r$ la R-cobertura de s $S_{A,R}(s)$, utilitzant $A = (I;M) = (a_1,\cdots,a_n)$, una matriu $r \times n$ on I és la matriu identitat $r \times r$ i M és la matriu $r \times (n - r)$ amb valors de F_q, és defineix com :

$$S_{A,R}(s) = \left\{ s + \sum_{j=1}^{n} \alpha_j a_j \mid \alpha_j \in F_q, \|\alpha_j \neq 0\| \leq R, 1 \leq j \leq n \right\}$$

El significat d'aquesta fórmula és la següent: per cobrir una paraula s el que hem de fer és generar tots els α_j tal que tinguin com a módul (en aquest cas seria millor parlar de pes de Hamming) menor o igual a R. una vegada generats els multipliquem per la matriu A i trobem els valors de les paraules que cubreix sumant-les a la paraula s.

Considerem, per exemple, un cas senzill on la matriu M=0, i per tant A=I. A més considerem que les paraules del codi són binàries. Llavors tenim:

$$S_{A,R}(s) = \left\{ s + \sum_{j=1}^{r} \alpha_j a_j \mid \alpha_j \in F_2, \|\alpha_j \neq 0\| \leq R, 1 \leq j \leq r \right\}$$

Però sabem que $\sum_{j=1}^{r} \alpha_j a_j = \{ \alpha_j \}$. O sigui és un vector tal que el seu mòdul és menor o igual a R, ja que:

$$\begin{bmatrix} 1 & & & 0 \\ & 1 & & \\ & & \ddots & \\ 0 & & & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_r \end{bmatrix} = [\ \alpha_1, \quad \alpha_2 \quad ,, \ \alpha_r\ ]$$

La generació de totes les paraules a una distància menor o igual a R de s és doncs:

$$S_{A,R}(s) = \left\{ s + \{\alpha_j\} \mid \alpha_j \in F_2, \left\| \alpha_j \neq 0 \right\| \leq R, 1 \leq j \leq r \right\}$$

Així per poder trobar aquestes paraules hem de generar totes les paraules $\{\alpha_j\}$ de longitud r tal que $\left\| \alpha_j \neq 0 \right\| \leq R$ on $\alpha_j \in F_2$. Després d'haver generat aquestes paraules les sumem a s per obtenir les paraules que buscàvem.

Quant utilitzem algun valor per la matriu M el que hem de fer és generar paraules de longitud n, multiplicar-les per la matriu A i el resultat sumar-lo al valor de la paraula s per trobar quines paraules cobreix.

Per generar el conjunt de paraules tal que $\left\| \alpha_j \neq 0 \right\| \leq R$ el que es fa és crear totes les paraules possibles de longitud n amb R valors de $\{\alpha_j\}$ diferents de 0. Seguidament es creen totes les paraules possibles de longitud n amb R-1 valors de $\{\alpha_j\}$ diferents de 0. Així fins arribar a una paraula amb tots els valors de $\{\alpha_j\}$ iguals a 0.

Un problema addicional és trobar la part M de la matriu A. Per solucionar-ho hem utilitzat un A.G. utilitzant la mateixa funció de cost fem evolucionar les paraules de M. D'això en parlarem amb més detall al parlar dels operants genètics als codis de cobertura.

7.2.2 Operants genètics als C.C.

En l'estudi de l'aplicació tant de l'encreuament, com de la mutació a l'aplicació de C.C., no trobem tantes dificultats com als C.W.C. L'única dificultat ens vindrà pel fet d'utilitzar una cobertura amb una matriu. En aquest cas haurem d'operar no només amb els individus de la població, sinó també amb les components d'aquesta matriu.

7.2.2.1 Encreuament als C.C.

L'encreuament es realitza entre paraules q-aries o del tipus $F_4^l F_2^b$ que no comporten cap problema especial a l'hora de creuar-les. Malgrat tot, l'encreuament que s'ha utilitzat ha estat el mateix que el emprat

amb els C.W.C. Això és l'encreuament que anomenàvem codi a codi en el que les paraules eren indivisibles.

La raó de no haver utilitzat un encreuament simple és que al nostre algorisme l'individu està estructurat en un array bidimensional per contenir paraules separades dins del codi. Aquest tipus d'estructura s'ha creat així, en lloc d'un únic array unidimensional, sense cap separació entre les paraules, per tal de facilitar els algorismes de càlcul de la funció de cost.

Amb l'encreuament codi a codi, i utilitzant aquest tipus de dades, s'ha pogut realitzar un encreuament algorísmicament més senzill, i, tal com hem vist amb el encreuament de C.W.C., igualment efectiu.

A més, l'encreuament ha de creuar les paraules que composen M. Aquest encreuament s'ha realitzat paraula a paraula (la forma clàssica de realitzar-ho) ja que l'únic exemple en el que l'hem utilitzat té una única paraula a M. En el cas que tinguéssim més paraules l'encreuament es faria igualment. La primera paraula de la matriu M del primer individu es creuaria amb la primera paraula del segon individu, la segona paraula de M del primer individu amb la segona paraula de M del segon individu etc. Així continuaríem fins arribar a l'última paraula. Per a cada paraula s'elegeix un punt de tall diferent.

Degut al fet de que només tenim, com a referència, un únic exemple que utilitza una matriu M amb una paraula no hem pogut constatar les diferències en l'eficiència dels possibles mètodes d'encreuament de les paraules que es poden implementar. És per això que hem implementat el que era més senzill algorísmicament.

7.2.2.2 Mutació als C.C.

La mutació no porta cap problema significatiu. L'us de dades q-àries i del tipus $F_4^1F_2^b$ ens obliga a modificar la mutació clàssica lleugerament.

Com en el cas anterior de mutacions a C.W.C. triem una paraula del codi i una posició dins d'aquesta paraula abans de mutar-la. Si aquesta posició pot representar q símbols el que fem és canviar el valor actual, de forma que adopti qualsevol dels q-1 símbols restants. Per fer aquesta operació generem aleatòriament un valor entre 1 i q-1 i el sumen en

mòdul q al valor actual. Per exemple si el valor actual d'un element agafat d'un alfabet d'un alfabet té 5 elements; això és C ∈ {0,1,2,3,4} i té un valor actual de 2, generem a l'atzar entre 1 i 4.

Per exemple, suposem que generem a l'atzar el nombre 4, el resultat seria 2+4=1 en mòdul 5. L'únic problema amb el que ens podem trobar és el cas en que treballem amb estructures del tipus $F_4^1 F_2^b$. En aquest cas haurem de comprovar en quin lloc dins de la paraula ens cau la mutació i realitzar l'operació en mòdul 4 quant caigui en el primer dígit, i en mòdul 2 en la resta de posicions dins del string.

Un altre aspecte que hem de tractar és la mutació de les paraules de la matriu M, quant aquesta existeix. Aquesta mutació es realitza independentment de la realitzada amb els codis i amb la mateixa probabilitat.

Després d'haver analitzat l'aplicació de l'A.G. tant als C.C. com als C.W.C. passem a descriure un dels operants genètics més importants per arribar a trobar bons resultats en un A.G. Ens referim a l'escalat de la funció que ens assegura una més ràpida, i eficient, convergència cap a la solució del problema.

7.3 Escalat de la funció de cost

En un apartat anterior hem esmentat diferents tipus de funcions de cost. Aquestes servien per garantir la competència, en tot moment dels strings. Per un bon funcionament de l'A.G. s'ha de garantir aquesta competència en tot moment i per qualsevol funció de cost. La funció de l'escalat és doble:

Primer, si tenim una diferència relativament curta entre els diferents valors de la funció de cost l'A.G. no evolucionarà o ho farà molt lentament. Si escalem la funció, es produirà una major descendència per part dels individus de major funció de cost, en comparació amb el cas sense escalar. Per fer-ho l'escalat augmenta les diferències relatives entre els valors de la funció de cost.

Segon, si tenim una gran diferència relativa entre els diferents valors de la funció de cost ens podem trobar en una generació amb un individu amb un valor de la funció de cost molt més gran que la resta. Si això es produeix la majoria d'encreuaments es produiran amb aquest

individu, i es perdrà molta informació en la recerca amb el que podem trobar-nos molt fàcilment encallats en un màxim local. Recordem que gran part de la potència d'un A.G. es trobava en treballar amb una diversitat de punts i no amb un de sol.

Per aquest A.G. hem provat varis tipus de funcions d'escalat. A més a més hem creat un nou tipus que s'ha mostrat com un escalat molt útil i potent. Per entendre millor la diferència entre els diferents escalats existents analitzem-los amb una mica més de profunditat.

La probabilitat de que un individu x_i, amb una funció de cost $f(x_i)$ sigui seleccionat és:

$$P_{sel}(x_i) = \frac{f(x_i)}{\sum_{j=1}^{n} f(x_j)}$$

On n és la mida de la població. Podem veure que si multipliquem el valor de la funció de cost per una constant, no tindrem cap canvi en la probabilitat de selecció.

$$P_{sel}(x_i) = \frac{K \cdot f(x_i)}{\sum_{j=1}^{n} K \cdot f(x_j)} = \frac{K \cdot f(x_i)}{K \cdot \sum_{j=1}^{n} f(x_j)} = \frac{f(x_i)}{\sum_{j=1}^{n} f(x_j)}$$

Un escalat àmpliament utilitzat és l'escalat lineal de la funció de cost mitjançant la fórmula f'= af +b de forma que avg'=avg i max'=2avg. Aquest escalat està limitat a que f'>0 i per tant min>0. On max es el valor màxim aconseguit per un individu dins de la actual generació, min és el valor mínim de la generació actual i avg és el valor mitjà de tots els valors dels individus de la generació actual.

Quant aquestes restriccions no s'acompleixen es canvien els valors de a i b per tal que avg=avg' i min'=0. En qualsevol d'aquests casos la probabilitat de selecció és:

$$P_{sel}(x_i) = \frac{a \cdot f(x_i) + b}{\sum_{j=1}^{n} a \cdot f(x_j) + b} = \frac{f(x_i) + \frac{b}{a}}{\sum_{j=1}^{n} f(x_j) + \frac{b}{a}}$$

Amb el que es demostra que el que es fa realment és pujar o baixar el nivell mig de la funció afegint un valor constant.

Altra escalat que ha resultat ser molt efectiu és l'escalat de ranking. Aquest escalat ordena els individus segons el seu valor de funció de cost i li assigna valors lineals des de un valor màxim (generalment la mida de la població) fins a 1. En aquest mètode la probabilitat de selecció no dependrà en absolut de la funció i la probabilitat de que un individu sigui seleccionat depèn només de l'ordre que ocupi aquest, dins de la població.

Aquest mètode no conserva la relació entre la selecció i la funció de cost amb el que la selecció perd part del seu sentit. un altre inconvenient és que individus amb igual funció de cost, reben un tracte diferent depenen de la posició que ocupin a la població i de com realitzem l'ordenació. És un mètode, doncs, que depèn de l'algorisme d'ordenació.

La probabilitat de selecció d'un individu que ocupi la posició i dins de la població és la següent:

$$P_{sel}(x_i) = \frac{i}{\sum_{j=1}^{n} j} = \frac{2 \cdot i}{n \cdot (n+1)} = K \cdot i$$

En últim lloc analitzaré un nou tipus d'escalat creat especialment per aquesta aplicació però que degut al seu caràcter general pot ser aplicat a qualsevol altre problema. Aquest és l'escalat exponencial i té la forma següent:

$$F' = e^{K \cdot F} \quad \text{on} \quad K = \frac{K'}{max - avg}$$

Éssent max, com en els casos anteriors, el valor màxim de la funció de cost del millor individu de la població actual i avg la mitjana de tots els valors que prenen els individus de la generació actual.

Si calculem la probabilitat de selecció d'un individu que tingui el valor màxim respecte d'un individu amb un valor de la funció de cost igual a la mitjana és:

$$\frac{P_{sel}(max)}{P_{sel}(avg)} = \frac{e^{\frac{K'}{max - avg}max}}{e^{\frac{K'}{max - avg}avg}} = e^{K'}$$

Amb el que tenim :

$$K' = Ln\left(\frac{P_{sel}(max)}{P_{sel}(avg)}\right)$$

Aquest resultat és molt interessant ja que ens permet controlar directament la probabilitat de selecció del màxim respecte de la mitjana amb un paràmetre variable. Aquesta és realment l'aspecte que en realitat han tractat de realitzar, amb més o menys claredat, els altres mètodes indirectament.

Els avantatges que el nostre mètode aporta són les següents:

1. És senzill de realitzar en comparació amb el d'ordenació, ja que no realitza cap operació complexa amb els individus, excepte el càlcul del màxim i de la mitjana, que sempre es solen realitzar per tenir un control de l'A.G.
2. Controlem directament la probabilitat de selecció relativa del màxim amb respecte de la mitjana, amb el que garantim l'evolució correcta del nostre algorisme.
3. Ens permet treballar sense problemes amb valors de f(x) negatius, ja que l'exponencial sempre serà positiva.
4. Podem treballar no només amb funcions creixents sinó a més podem treballar amb funcions decreixents (amb un mínim com a valor òptim). Per fer-ho només hem de canviar el valor max pel min, ja que així l'exponencial serà negativa i serà més gran per valors petits de x. La funció seria en aquest cas la següent:

$$F' = e^{K \cdot F} \quad \text{on} \quad K = \frac{K'}{max - min}$$

Com a únic inconvenient pràctic hem trobat que hem de vigilar amb el rang de valors que la funció pren, ja que l'exponencial pot

arribar a prendre valors molt alts. Aquest alts valors podrien provocar que l'exponencial sortís del rang de valors que a la simulació pot arribar a tenir, i per tant, el nostre A.G. no funcionés adequadament.

Per evitar aquest inconvenient hem modificat lleugerament la funció d'escalat:

$$F' = e^{K\cdot(F-avg)}$$

Aquesta variació no afecta per res la probabilitat absoluta ni relativa de selecció:

$$P_{sel}(x_i) = \frac{e^{K\cdot(f(x_i)-avg)}}{\sum_{j=1}^{n} e^{K\cdot(f(x_i)-avg)}} = \frac{e^{K\cdot f(x_i)}.e^{-K\cdot avg}}{\sum_{j=1}^{n} e^{K\cdot f(x_i)}.e^{-K\cdot avg}} = \frac{e^{K\cdot f(x_i)}.e^{-K\cdot avg}}{e^{-K\cdot avg}.\sum_{j=1}^{n} e^{K\cdot f(x_i)}} = \frac{e^{K\cdot f(x_i)}}{\sum_{j=1}^{n} e^{K\cdot f(x_i)}}$$

Amb aquesta modificació evitem en part aquest inconvenient al centrar la funció en torn de la mitjana.

Pel valor màxim obtindrem :

$$F' = e^{\frac{K'}{max - avg}(max - avg)} = e^{K'}$$

Aquest valor no serà en general massa alt. De fet podem dir que pels problemes estudiats es troba un valor òptim pel valor de K' entre 2 i 3. Això implica una probabilitat relativa de selecció entre 10 i 20.

En les nostres simulacions, tant de C.C. com de C.W.C, hem utilitzat els tres escalats prèviament analitzats. El que ha mostrat millors prestacions tant de millor màxim assolit com de velocitat de convergència cap a aquest màxim ha estat l'escalat exponencial. En segon lloc en quant a eficàcia trobem l'escalat de ranking, i en últim lloc l'escalat lineal. Per tant hem de dir que en les recerques exhaustives de codis hem utilitzat l'escalat exponencial amb un valor de k' entre 2 i 3.

Ara explicarem com hem escollit els valors de les anomenades variables d'execució.

7.4 Variables d'execució

Anomenem variables d'execució a un conjunt de variables que han de ser fixades abans de l'execució del nostre A.G. i del que depèn la convergència cap a una bona solució del nostre problema.

Aquestes variables són quatre: mida de la població, probabilitat d'encreuament, probabilitat de mutació i el factor K' de l'escalat exponencial.

Degut al caràcter probabilístic del nostre mètode de trobar extrems no podem fer una única proba per tal de determinar quin és el paràmetre òptim a utilitzar en cada cas.

Per fer aquestes proves d'ajust el nostre A.G. executa varies vegades el mateix problema amb diferents llavors aleàtories, amb les mateixes variables d'execució i realitza una mitjana d'aquests resultats. Al fer aquesta mitjana s'intenta minimitzar els errors deguts al caràcter probabilístic del mètode.

El nostre A.G. implementat realitza un fitxer de resultats amb estadístiques dels valors màxims, mínim i les mitjanes de cada generació. A més a més guarda el millor codi trobat fins al moment. Aquesta estadística de resultats és la mitjana de cada generació d'un nombre fix d'execucions prèviament indicat. D'aquesta manera, observant aquestes estadístiques, podem saber amb quin conjunt de paràmetres es comporta millor el nostre algorisme.

Els resultats obtinguts ens indica que la mida de la població es pot fixar entre 200 i 500 sense observar grans canvis quant a velocitat de convergència mesurat pel màxim o per la mitjana assolida respecte del nombre de consultes a la funció de cost.

Per comprovar quins eren els millors paràmetres de probabilitat d'encreuament i de mutació es van realitzar moltes simulacions amb diferents valors. Per tal d'obtenir resultats ràpidament es van resoldre problemes semblants però de velocitat de convergència més ràpida. Per exemple, en el cas dels codis de pes constant es realitzaven simulacions amb un codi de mida menor al que volíem trobar.

Aquest fet és important, ja que ens permet ajustar aquests valors prèviament i per tant aconseguir millors resultats una vegada comenci la, de vegades llarga, execució en busca d'un bon codi.

La variable K' responsable de la probabilitat relativa de selecció va ser trobada de la mateixa manera. Per quasi tots els problemes es trobaven valors òptims de K' entre 2 i 3. Això corresponia a una taxa relativa de selecció $P_{sel}(max)/P_{sel}(avg)$ de 10 a 20. Valors més alts o més baixos no portaven cap a valors de la funció de cost més alts. Quan els valors eren més alts l'A.G. progressava molt ràpidament però quedava aturat en un mínim local, i quant tenia valors més baixos l'evolució era molt més lenta i s'assolia un màxim menor.

Podríem dir, com a resum, que vam utilitzar una població de 200 individus, un valor de K'=2.5 i unes probabilitats d'encreuament i de mutació depenent dels problemes.

Ara només ens queda mostrar els resultats obtinguts i comentar-los.

8 Conclusions i resultats

En aquest capítol es presentarà els resultats pràctics obtinguts en la realització d'aquest projecte. Per arribar a aquests resultats hem realitzat simulacions mitjançant la implementació pràctica de diferents versions d'algorismes genètics en llenguatge C. Aquestes implementacions han anat variant amb el temps. Quan anaven donant resultats l'algorisme era renovat per tal de millorar les seves prestacions.

Les simulacions s'han realitzat als ordenadors SUN (Workstation amb S.O. UNIX i entorn Windows) del departament de Matemàtica Aplicada i Telemàtica de l'E.T.S.E.T.B.

Començarem descrivint la utilització del programa en C implementat. Després justificarem el fet d'utilitzar un escalat exponencial, amb una comparació en igualtat de condicions amb la resta d'escalats existents. Continuarem amb l'estudi de les variables d'execució òptimes i per últim presentarem els resultats obtinguts tant en codis de cobertura com en codis de pes constant.

8.1 Algorismes implementats

Encara que han estat moltes les versions dels algorismes implementats, el nombre definitiu d'algorismes utilitzats per obtenir resultats han estat quatre. El primer algorisme està fet per trobar codis de pes constant i s'anomena nou.c. El segon s'utilitza per trobar codis de cobertura q-aris que no requereixen la utilització d'una matriu per la seva cobertura i s'anomena cover.c. El tercer troba codis de cobertura que utilitza una matriu M per la seva cobertura i s'anomena coverM.c. I l'últim troba codis mixt de cobertura i s'anomena coverf4.c.

Degut a la llarga execució (en temps) que tenen en general aquests algorismes no hem realitzat un menú interactiu que ens demanés les diferents variables d'execució. En lloc d'això, aquestes variables són definides mitjançant "`#define`" amb el que una vegada compilat, aquestes variables són agafades com a constant. Això ens permet executar ràpidament el programa sense que ens aparegui cap missatge per pantalla, però ens obliga a compilar de nou cada vegada que volem

canviar-les. Aquestes variables (població, individu, etc.) ocupen moltes posicions de memòria i és convenient que siguin definides, en quant a mida, avanç de que el programa sigui compilat.

Els nostres algorismes poden donar resultats per pantalla i per fitxer de sortida. La prestació de donar resultats per pantalla es pot inhibir. Això pot ser útil en el cas d'una execució en *background*. En aquest tipus d'execució el programa s'executa mentre s'està realitzant un altra tasca i per tant no ens interessa que vagin sorgint resultats a la pantalla, que ens podrien molestar.

El programa presenta els seus resultats mitjançant dos fitxers de sortida. Aquests fitxers de resultats són de text. El primer fitxer de text té com a terminació **.cod** i ens guarda el millor codi trobat fins el moment en l'execució del programa. Aquest fitxer a més té una capçalera que ens indica les probabilitats de mutació i d'encreuament utilitzades per trobar el codi, la mida de la població, el valor de la funció de cost d'aquest codi i també ens indica a quina generació ha estat trobat.

El segon fitxer de text és de dades, té com a terminació **.dad** i ens guarda els valors màxim, mínim i la mitja de cada generació. A més té una capçalera amb dades importants, com per exemple la mida de la paraula del codi, el nombre de paraules del codi, etc. Aquests valors de cada generació poden ser valors únics d'una generació, o bé mitjanes de diferents execucions. Quan el nombre d'iteracions és més gran d'u els valors corresponents a cada generació al fitxer de sortida, serà la mitja dels diferents valors a cada generació.

Aquestes mitjanes es realitzen generació a generació. Per exemple, un valor utilitzat com a dada important és el valor màxim de la funció de cost que pren un individu a una generació. Al fitxer de resultats tindrem a la generació indicada com a i la mitja dels n màxims de les n iteracions de les generacions i de les corresponents execucions.

El nostre programa permet fer execucions amb diferents variables d'execució (probabilitat de mutació, d'encreuament, i factor K de l'escalat) de forma automàtica. Per fer-ho només hem d'indicar-li el valor de la variable inicial, l'increment d'aquesta variable i el valor màxim que aquesta variable pot prendre. A més si fem que les tres variables variïn a la vegada, ens farà execucions amb totes les combinacions de variables possibles.

Per cada conjunt de variables es faran n iteracions i per cada grup d'aquestes variables tindrem un fitxer de sortida de dades diferents. El fitxer del millor codi serà únic per totes les execucions.

El nom base del fitxer de resultats és definit per nosaltres mitjançant un altra variable del tipus **#define**. A més degut a que es realitza simulacions amb diferents valors de les variables d'execució s'afegeix un índex numèric per diferenciar els diferents fitxers **.dad** que es crearan.

Així, si el nom base s'anomena **Nomfitx**, el primer fitxer creat s'anomenarà **Nomfitx0.dad**, el segon **Nomfitx1.dad**, el tercer **Nomfitx3.dad**, etc. En canvi el fitxer que ens guarda en tot moment el millor resultat és únic i no varia pel fet d'utilitzar diferents variables d'execució, anomenat-se **Nomfitx.cod**. Com hem indicat anteriorment a la capçalera d'aquest fitxer es guarda la probabilitat de mutació i d'encreuament utilitzats per trobar aquest codi, el que ens pot ajudar a decidir quin és el millor conjunt de variables d'execució.

8.1.1 Nou.c

Aquest algorisme tracta de trobar bons codis de pes constant. Al començament del llistat en C ens trobarem amb un conjunt de sentències o instruccions que comencen amb la paraula **#define**. Aquestes han estat retolades amb un comentari que ens indica "Constants relacionades amb A.G." i són les que ens definiran els tipus de codis que volem trobar i les variables d'execució de l'algorisme. Posarem un exemple d'aquestes sentències i indicarem que volen dir cadascuna d'aquestes constants:

```
#define NUMCOD      18
#define LONGCOD     23
#define PES          7

#define MAXGEN     100
#define MAXPOP     200
#define MAXITER     20

#define PMUTINIC   0.03
#define PMUTFIN    0.07
```

```
#define PMUTINC          0.01

#define PCROSSINIC       0.6
#define PCROSSFIN        1.0
#define PCROSSINC        0.1

#define KINIC            2.0
#define KFIN             2.5
#define KINC             0.5

#define NOMFITX          "NAT18"

#define PANTALLA         fals
#define RESULTAT         cert
```

La constant anomenada **NUMCOD** ens definirà el nombre de paraules del que està compost el codi. En aquest cas aquest nombre serà de 18 paraules. **LONGCOD** és la mida de la paraula del codi. És el nombre d'elements binaris del que està formada. La variable **PES** ens indica el pes de les paraules del codi. Amb aquestes tres variables ja tenim definit el tipus de codi que volem trobar.

MAXGEN és el nombre màxim de generacions que volem que generi el nostre algorisme genètic. També podem definir, amb la variable **RESULTAT**, que l'algorisme s'aturi quant trobi un resultat desitjat i per tant no s'arribin a aquest nombre de generacions.

MAXPOP és la mida de la població, o sigui el nombre d'individus que composa cada generació. Aquest valor no és crític i es sol fitxar en 200. Valors més alts requeriran valors de **MAXGEN** més petits i al inrevés valors més petits impliquen MAXGEN més alts, fins a cert límit es clar.

MAXITER és el nombre d'iteracions que es realitzaran amb cada una de les variables d'execució, que són la probabilitat de mutació, la probabilitat d'encreuament i el factor K de l'escalat exponencial.

Mitjançant les variables **PMUTINIC**, **PMUTFIN** i **PMUTINC** realitzem execucions amb diferents valors de la probabilitat de mutació de forma automàtica. En aquest cas la primera execució treballarà amb una probabilitat de mutació de 0.03 (PMUTINIC) i l'última amb una

probabilitat de mutació de 0.07 (PMUTFIN). La gradació entre aquests dos valors serà de 0.01 i ens vindrà marcat per la variable PMUTINC. Així en aquest cas el nostre algorisme realitzarà execucions amb les probabilitats de mutació 0.03, 0.04, 0.05, 0.06 i 0.07.

La mateixa utilitat tenen les variables **PCROSSINIC, PCROSSFIN i PMUTINC**. PCROSSINIC és la probabilitat d'encreuament inicial, PCROSSINC és l'increment en aquesta probabilitat i PCROSSFIN és la probabilitat màxima d'encreuament.

Les variables **KINIC, KFIN i KINC** defineixen els diferents valors que pot prendre el factor K de l'escalat exponencial. En aquest cas K només prendrà els valors 2 i 2.5.

La variable **NOMFITX**, com el seu nom indica, és la base dels noms dels fitxers de sortida. Aquests fitxers són els indicats anteriorment amb terminacions .cod i .dad.

La variable **PANTALLA** és una variable booleana que pot prendre els valors cert o fals. Quant aquesta variable pren el valor cert, ens apareixen els resultats de cada generació (el màxim, el mínim i la mitja) en pantalla. Això ens serà útil sempre que el temps d'execució no sigui molt llarg i volguem treure alguna conclusió ràpida. Si la variable pren el valor de fals el programa no traurà cap resultat per pantalla i només coneixerem l'evolució dels resultats pels fitxers de sortida.

La variable **RESULTAT** ens marcarà la forma d'execució del programa. Quant aquesta variable pren el valor cert el programa s'atura quant troba un resultat desitjable, que en el nostre cas és quant troba un codi amb una distància mínima de 10. Si aquesta variable pren el valor de fals el programa continuarà fins que esgoti el nombre màxim d'execucions per tots els valors de les variables d'execució. Per tal podem dir que utilitzarem el valor cert quant vulguem cercar un bon codi i el valor fals quant vulguem fer una estadística dels millors valors de les variables d'execució.

8.1.2 Fitxers de resultats

Ara exposaré amb detall el contingut dels fitxers de sortida, tant del fitxer de codi com del fitxer de dades, explicant el contingut de les capçaleres i de la informació obtinguda amb l'execució del programa.

En el fitxer de resultat de dades hi ha una capçalera amb la següent informació:

- **PMUTATION** = Ens indica la probabilitat de mutació de l'actual execució.
- **PCROSS** = Ens indica la probabilitat d'encreuament de l'execució actual.
- **MAXPOP** = És la mida de la població amb la que s'ha realitzat la simulació.
- **MAXITER** = És el nombre d'iteracions amb el que s'ha realitzat aquest fitxer de resultats.
- **NUMCOD** = És el nombre de paraules del que està compost el codi.
- **LONGCOD** = És la longitud de la paraula del codi
- **KPROB** = És el valor que pren la variable K de l'escalat exponencial en aquesta simulació.

Després d'aquesta capçalera ens trobem amb la informació promig en cada generació de les diferents iteracions. La informació té la següent presentació.

GEN = MAX = MIN = AVG =

Com es pot deduir fàcilment **GEN** és el nombre de la generació del que es fa la mitja de les diferents iteracions, **MAX** és la mitja dels màxims assolits a les corresponents generacions, **MIN** és el mateix però amb els mínims de cada generació i **AVG** és la mitja de les mitjanes de cada iteració a la generació corresponent.

En el fitxer que guarda el millor codi hi ha una capçalera amb la següent informació:

- **GENERACIO** = És la generació en la que s'ha assolit el codi presentat en aquest fitxer.
- **COST** = És el valor de la funció de cost d'aquest codi.
- **DMIN** = És la distància mínima d'aquest codi. Normalment la distància mínima desitjada és 10.
- **KPROB** = És el valor de K (factor de l'escalat exponencial) amb el que s'ha assolit aquest codi.
- **PMUTATION** = És el valor de la probabilitat de mutació amb el que s'ha aconseguit el codi.
- **PCROSS** = Igual però amb la probabilitat d'encreuament.
- **MAXPOP** = És la mida de la població, el nombre d'individus del que està format.

Després d'aquesta informació ens trobem, per suposat, amb el codi que hem trobat com a millor.

8.1.3 Cover.c, CoverM.c i CoverF4.c

Aquests tres programes en C són pràcticament iguals en quant a utilització al cas anterior. La forma de controlar les variables d'execució: probabilitat de mutació, probabilitat d'encreuament i factor K és igual que en el cas anterior. També és igual la forma de definir el nombre màxim de generacions, la mida de la població i el nombre màxim d'iteracions. Les diferències les trobem, lògicament, a l'hora de definir el codi a trobar.

En aquest cas ens trobarem que hem de definir les següents variables.

```
#define BASE          5

#define NUMCOD       35
#define LONGCOD       5
#define NUM_M         0
```

```
#define MAXGEN          100
#define MAXPOP          200
#define MAXITER          20

#define LONGESPAI       3125

#define RADI             2

#define PMUTINIC        0.7
#define PMUTFIN         0.7
#define PMUTINC         0.1

#define PCROSSINIC      0.6
#define PCROSSFIN       1.0
#define PCROSSINC       0.1

#define KINIC           2.0
#define KFIN            2.5
#define KINC            0.5

#define NOMFITX         "c5535"

#define PANTALLA        fals
#define RESULTAT        fals
```

Només explicarem les variables que difereixen del cas anterior. En primer lloc, la variable **BASE** és la base en el que està definida el codi. Per tant els elements que composen les paraules del codi podran prendre valors que variaran des de 0 a BASE-1. En el cas de codi del tipus mixt aquest valor s'iguala a 2.

NUMCOD i **LONGCOD** defineixen la mida del codi. **NUM_M** defineix el nombre de paraules del que està format la part M de la matriu A, quant s'utilitza una cobertura mitjançant una matriu. Quant no s'utilitza aquesta cobertura el valor ha de ser zero i la cobertura es realitzarà de forma clàssica.

LONGESPAI és la mida de l'espai que cobreix el codi que volem trobar. Per calcular-lo hem de fer l'operació: LONGESPAI es igual a BASE elevat a LONGCOD.

RADI és el radi de cobertura del codi sobre l'espai.

La resta de variables són com en el cas anterior.

8.1.4 fitxers de resultats

Els fitxer de resultats són molt similars al cas en el que volíem trobar codis de pes constant. Les úniques diferències són l'adició de la variable **RADI** que ens indica el radi de la cobertura i l'eliminació de la variable **DMIN** en el fitxer del millor codi. A més en la presentació del millor codi es presenta la part M de la matriu A, quant aquesta és utilitzada.

El fitxer de dades presenta el mateix aspecte, i el mateix contingut que el fitxer de dades del codis de pes constant.

8.2 Elecció del tipus d'escalat idoni.

Es va crear un nous tipus d'escalat degut a que els escalats utilitzats fins ara no donaven bons resultats en aquest tipus de problemes. Ni amb l'escalat lineal ni amb l'escalat de ranking s'aconseguien bons resultats. De tota manera, encara que només vam aconseguir bons resultats amb l'escalat exponencial hem fet diverses proves que ens va confirmar el que ja sabíem, que l'escalat exponencial tenia millors prestacions que la resta.

Hem realitzat diferents execucions per els mateixos valors de la probabilitat de mutació i d'encreuament amb tres tipus d'escalat: Escalat lineal, escalat de ranking i escalat exponencial. També hem inclós una execució sense escalar per valorar apropiadament l'efecte d'aquest operant.

Les figures següents ens mostren les mitjanes de cada generació sent els resultats corresponents als màxims i als mínims molt similars. Els resultats presentats són un mitja de cinc iteracions. Això és fa per augmentar la fiabilitat dels resultats.

Per un codi de pes constant, de longitud 23, 10 paraules, pes 7, amb una probabilitat de mutació i d'encreuament de 0.6 i una població de 200 individus hem obtingut els següents resultats en 200 generacions. La figura següent ens mostra els resultats obtinguts en la mitja de la població en els quatre casos d'escalat:

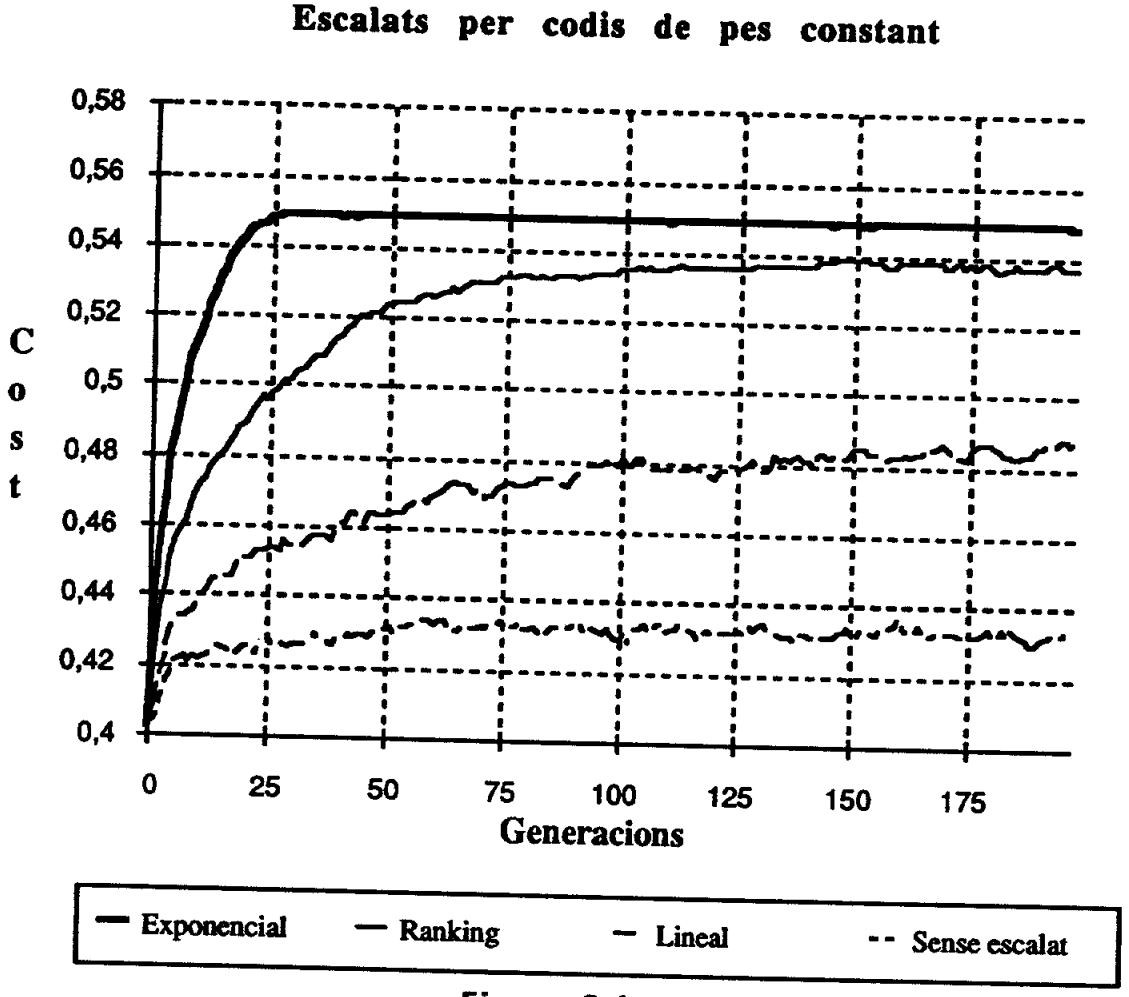


Figura 8.1

S'observa com l'escalat exponencial no només és clarament superior sinó que a més troba el resultat molt aviat, assolint el seu valor màxim (que era la solució òptima) sobre la generació 25.

Per un codi de cobertura de longitud 5, amb 12 paraules, radi de cobertura 1, amb unes probabilitats de mutació i d'encreuament de 0.5, amb una població de 100 individus i amb una execució de 200

generacions hem obtingut els següents resultats per les mitjanes de la població:

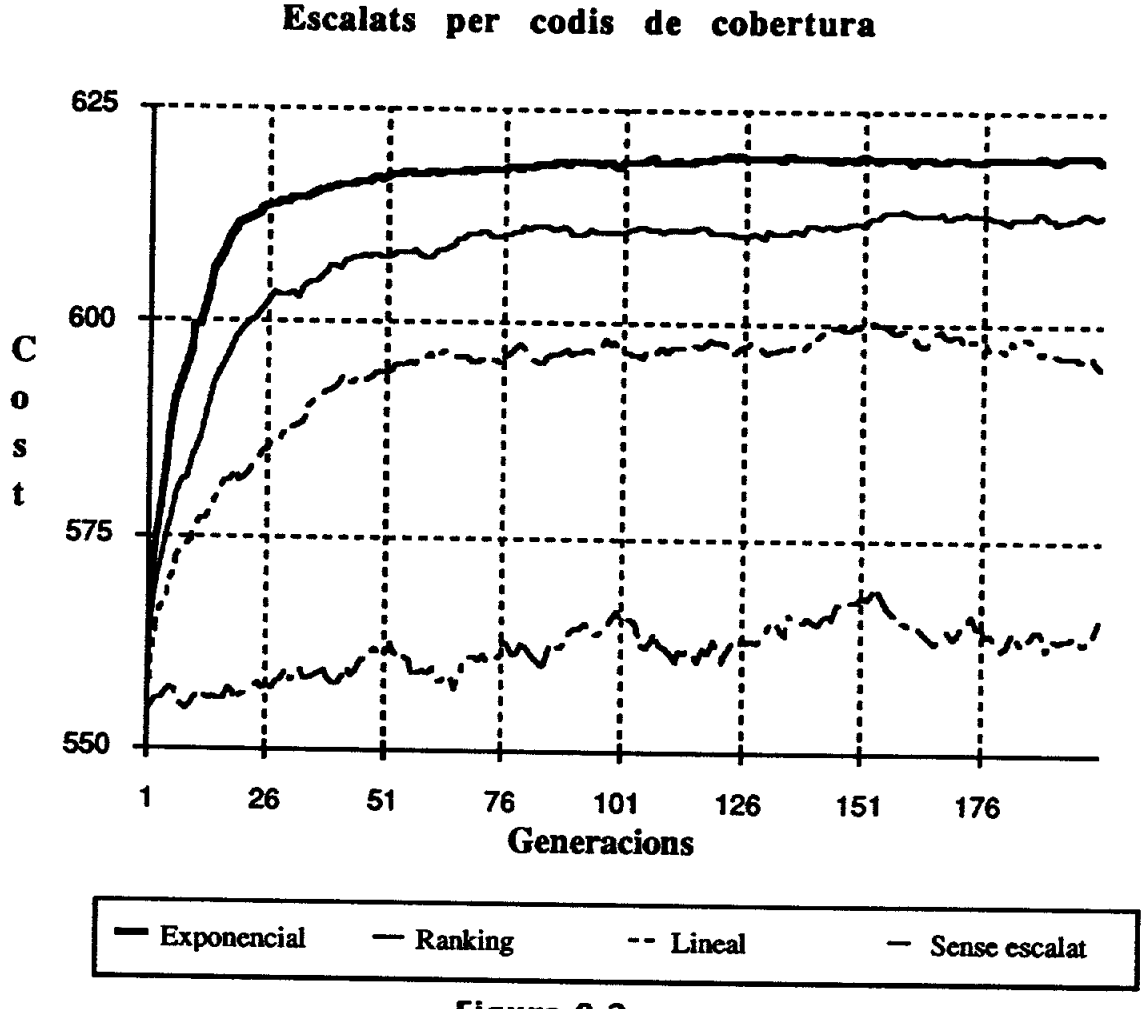


Figura 8.2

En aquest cas també l'escalat exponencial és clarament superior a la resta. També ens mostra com l'escalat de ranking ocupa el segon lloc i l'escalat lineal ocupa l'última posició, però amb millores notables sobre el cas en el que no s'utilitzava cap escalat. Com veurem a continuació és molt més crític (per la seva influència en els resultats) l'elecció d'un bon escalat que l'elecció de les probabilitats de mutació i d'encreuament.

8.4 Elecció de les variables d'execució

Per poder trobar els valors idonis tant de les probabilitats de mutació com d'encreuament es va fer un escombrat pels diferents valors de les probabilitats de mutació i d'encreuament. Aquestes mesures es van realitzar amb l'escalat exponencial i amb un valor de K=2.

Els resultats que es presentaran a continuació corresponen a les mitjanes de cada generació. Els resultats corresponents pels màxims i pel mínim són molt semblants. Per una millor fiabilitat de les dades s'han fet cinc iteracions amb cada variable d'execució per garantir la fiabilitat dels resultats.

Per últim hem de dir que aquests que es presenten no són els únics resultats obtinguts però sí els més complets, per ser els primers. A partir d'aquests s'han realitzat més per cada execució però no tan exhaustius.

Estadística1:

Elecció de les probabilitats de mutació i d'encreuament òptimes per un codi de pes constant. L'entorn de simulació va ser el següent:

- Longitud de la paraula:	23
- Nombre de paraules:	14
- Pes del codi:	7
- Mida de la població:	200
- Valor de K:	2.0
- Nombre d'iteracions:	5

Es van variar les probabilitat de mutació i d'encreuament des de 0.1 fins a 0.9. Els resultats de les mitjanes es presenten des de la figura 8.3 fins a la figura 8.8. Com es pot apreciar en aquestes simulacions el valor òptim de la mutació està en 0.3 i el d'encreuament en torn a 0.5. De tota manera podem apreciar que, en aquest exemple és molt més apreciable l'efecte de la probabilitat de mutació que la d'encreuament.

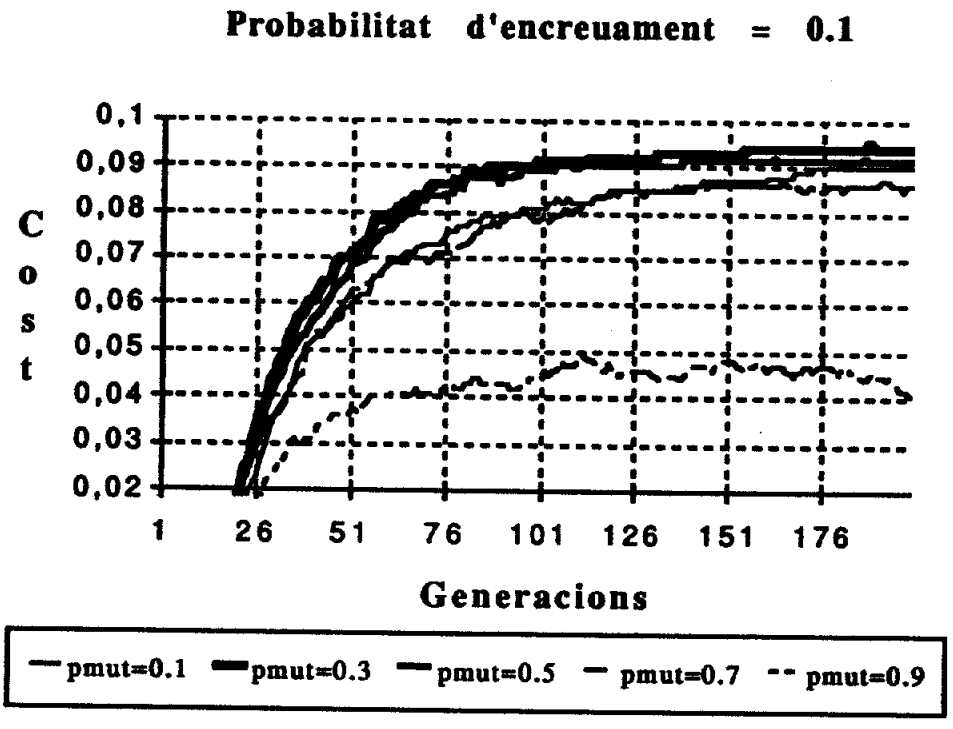


Figura 8.3

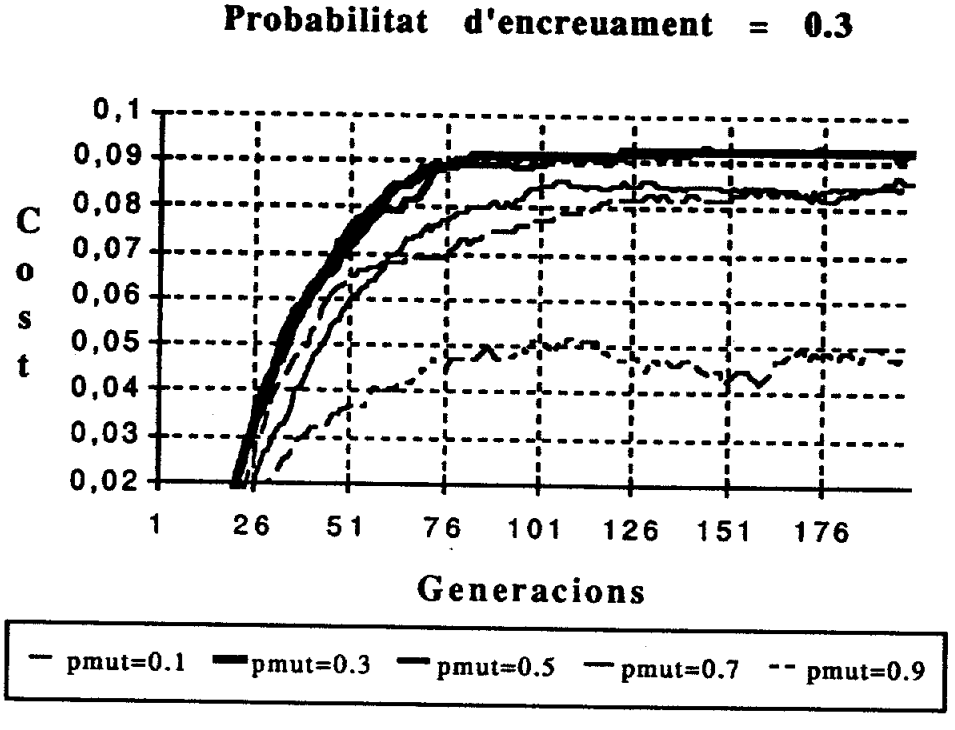


Figura 8.4

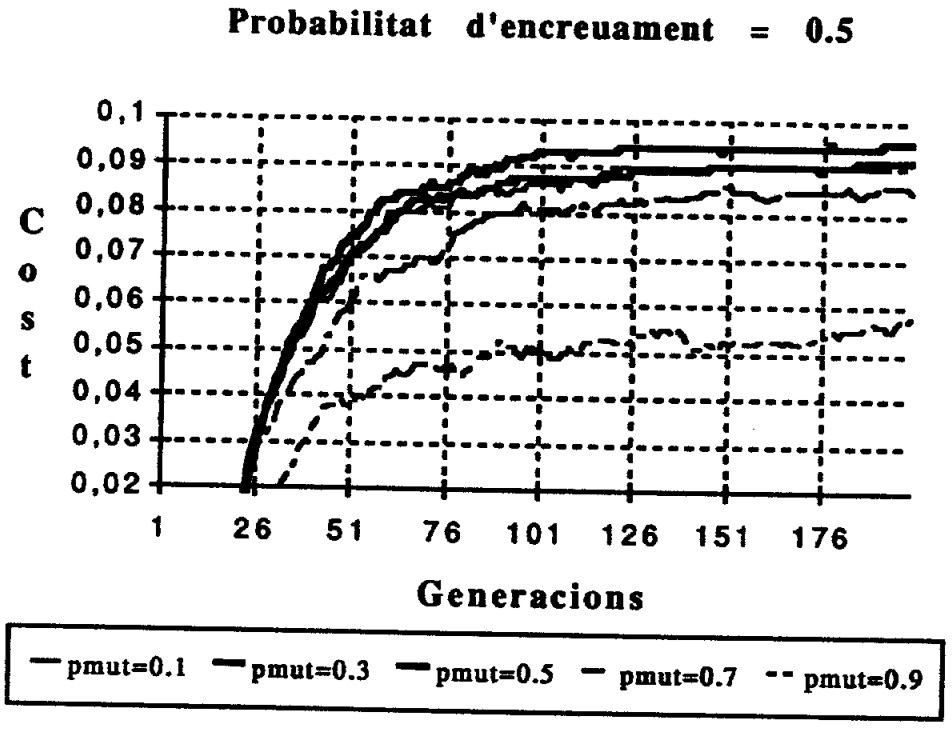


Figura 8.5

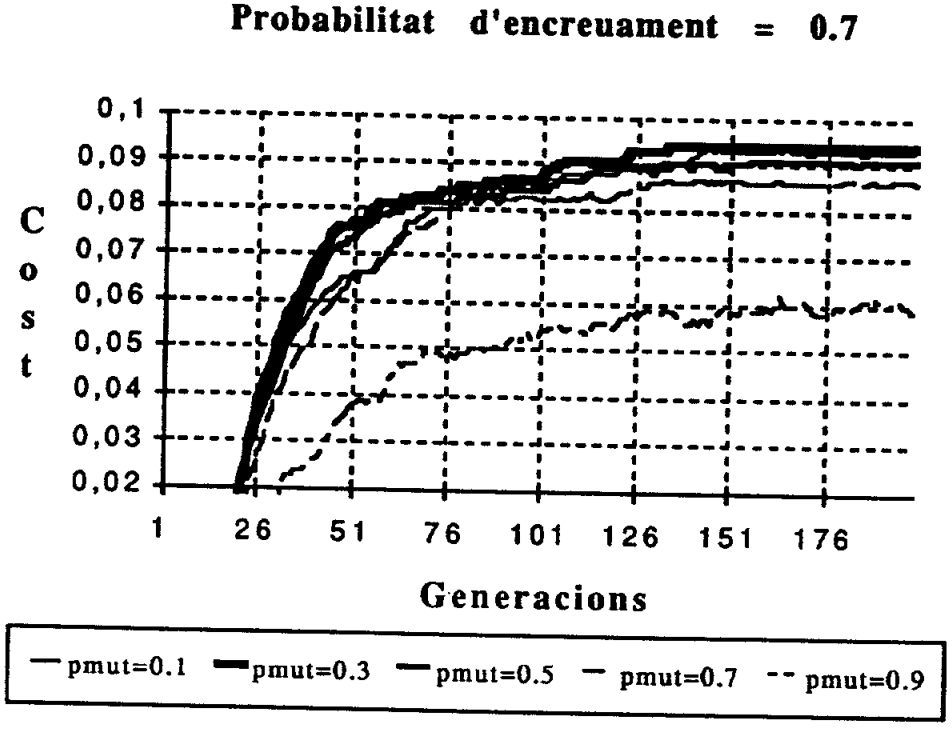


Figura 8.6

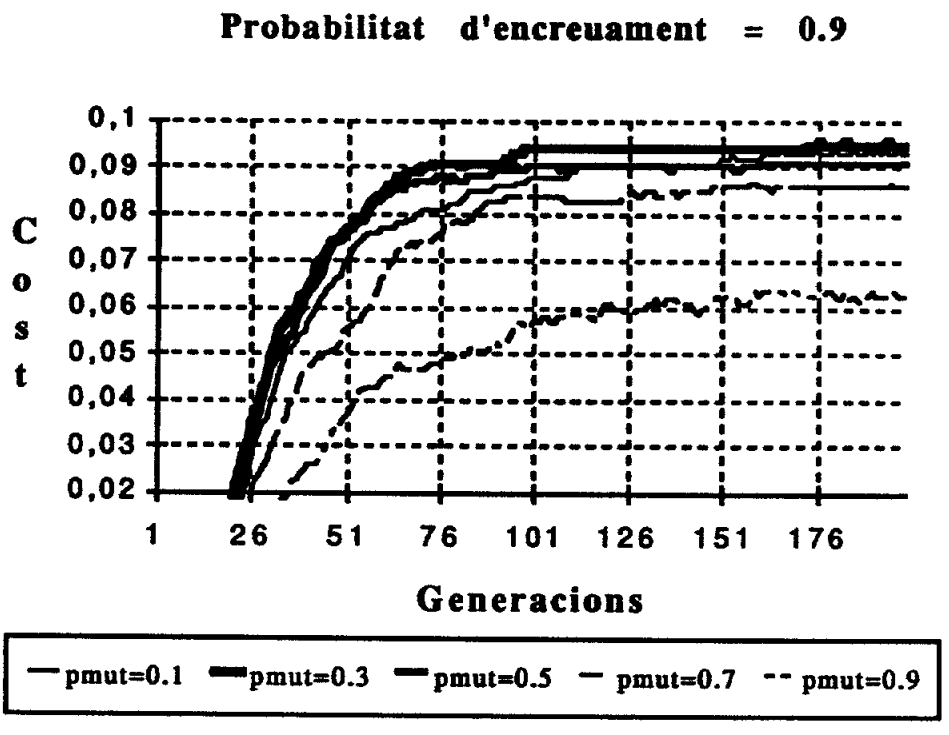


Figura 8.7

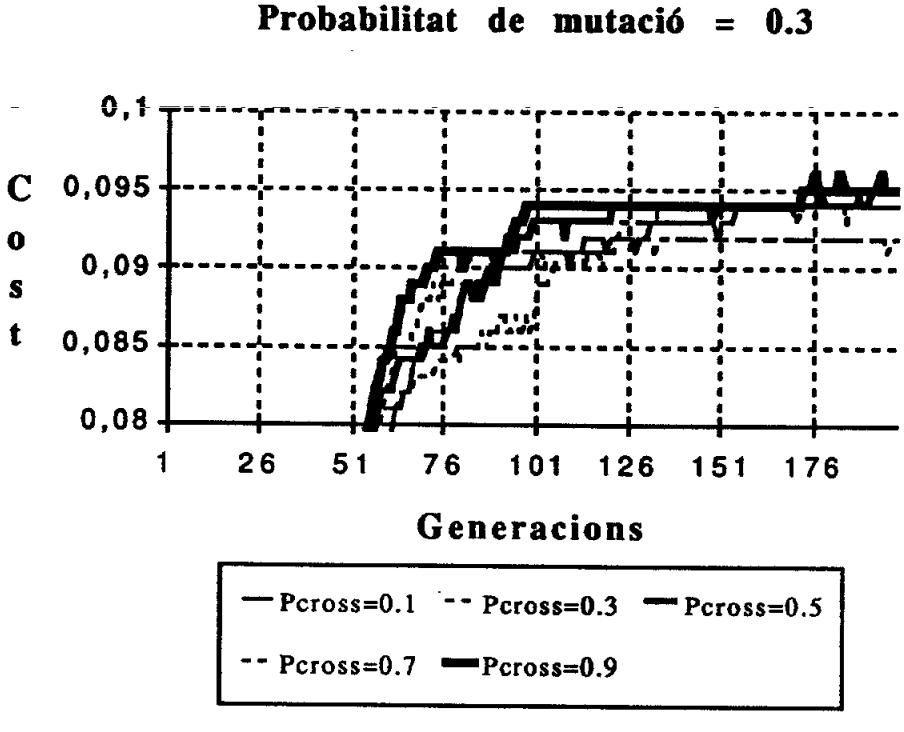


Figura 8.8

Estadística2:

Elecció de les probabilitats de mutació i d'encreuament òptimes per un codi de cobertura. L'entorn de simulació va ser el següent:

- Longitud de la paraula: 4
- Nombre de paraules: 11
- Radi de cobertura 2
- Mida de la població: 100
- Valor de K: 2.0
- Nombre d'iteracions: 5

Es van variar les probabilitat de mutació i d'encreuament des de 0.1 fins a 0.9. Els resultats de les mitjanes es presenten des de la figura 8.9 fins a la figura 8.13. Com es pot apreciar en aquestes simulacions el valor òptim de la mutació està en 0.3 i el d'encreuament en torn a 0.5. Com en el cas anterior és molt més apreciable l'efecte de la probabilitat de mutació que la d'encreuament.

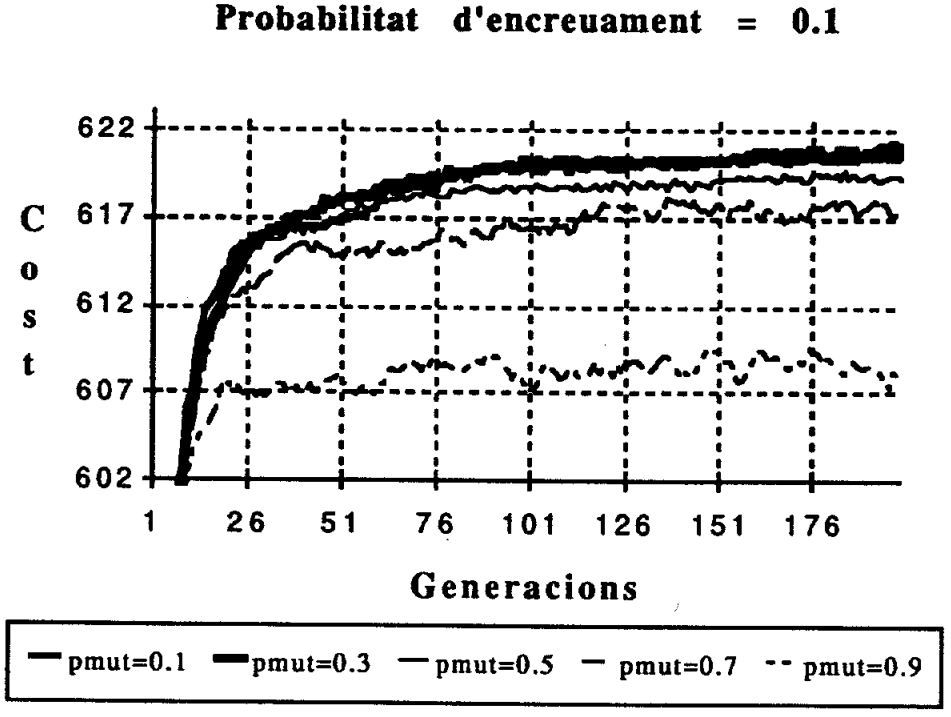


Figura 8.9

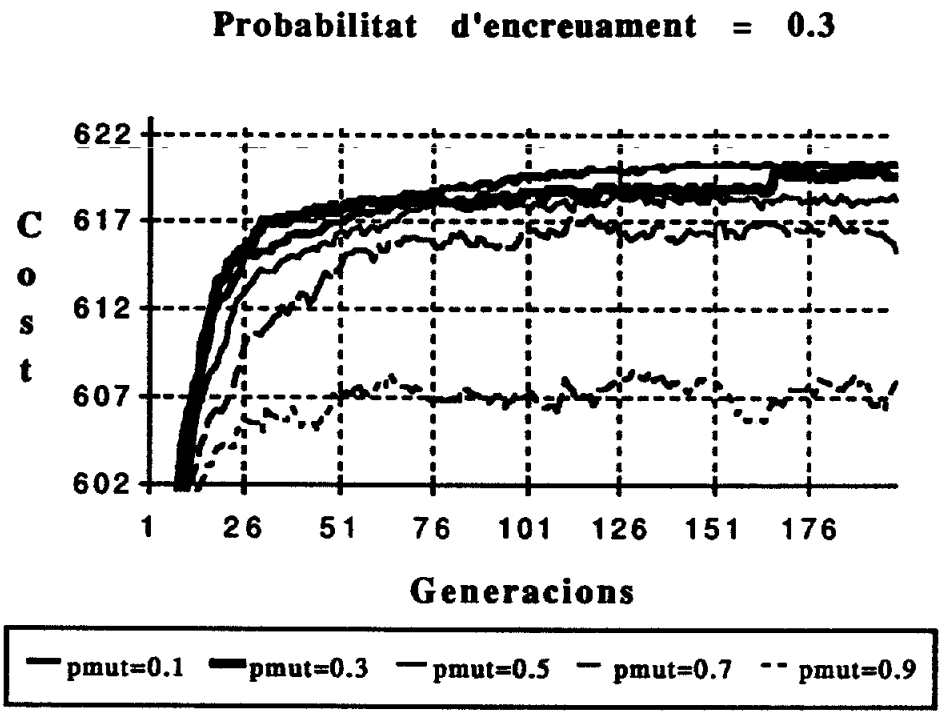


Figura 8.10

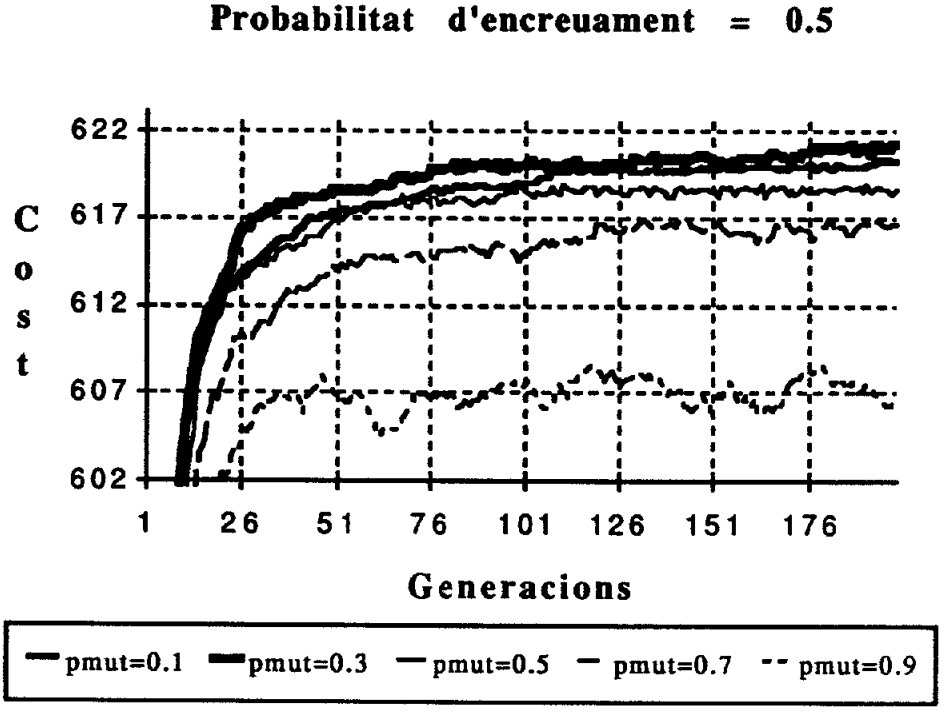


Figura 8.11

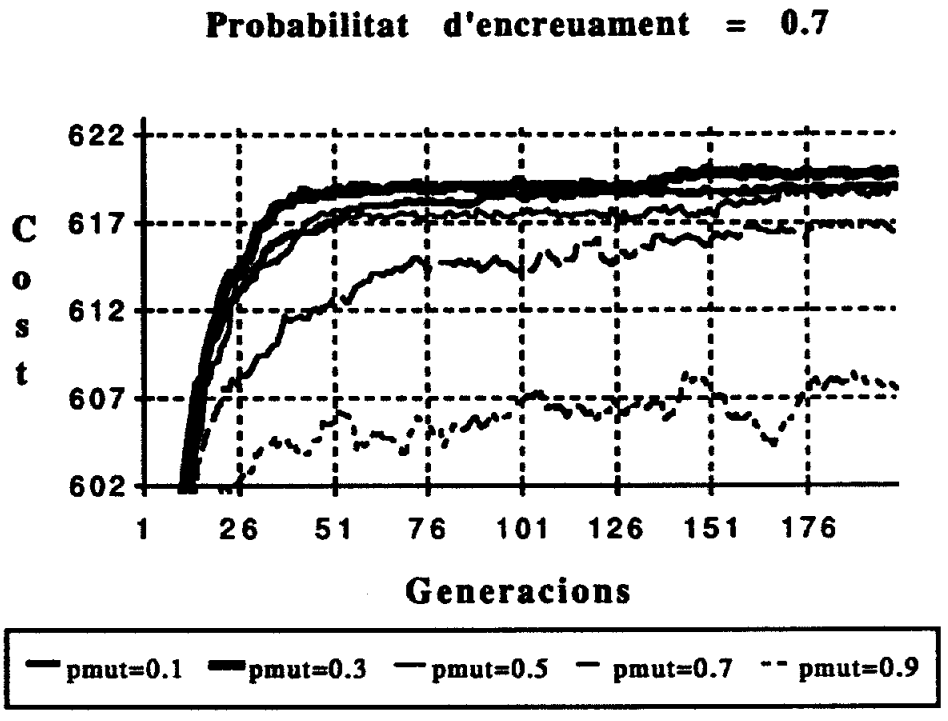


Figura 8.12

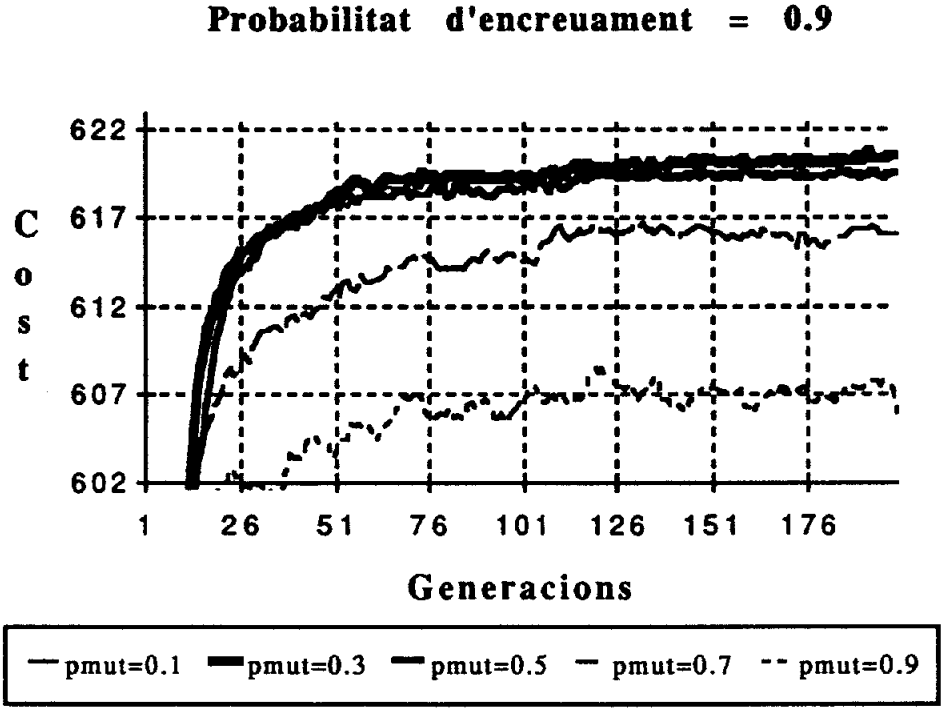


Figura 8.13

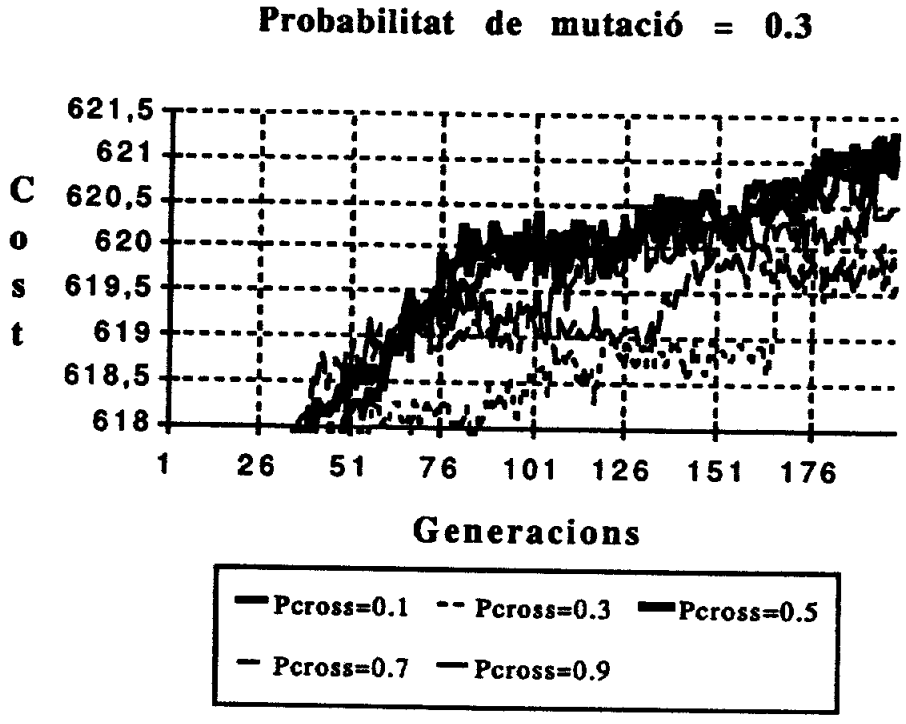


Figura 8.14

Estadística3:

Elecció del factor K de l'escalat exponencial òptim. L'entorn de simulació va ser el següent:

- Longitud de la paraula: 23
- Nombre de paraules: 14
- Pes del codi: 7
- Mida de la població: 200
- Probabilitat de mutació: 0.5
- Probabilitat d'encreuament: 0.5
- Nombre d'iteracions: 5

Es va variar el factor K des de 1.0 a 3.0. Els resultats es presenten des de les figures 8.15 a 8.17. En aquesta prova es troba un valor òptim de K per valors de K entorn a 2.6.

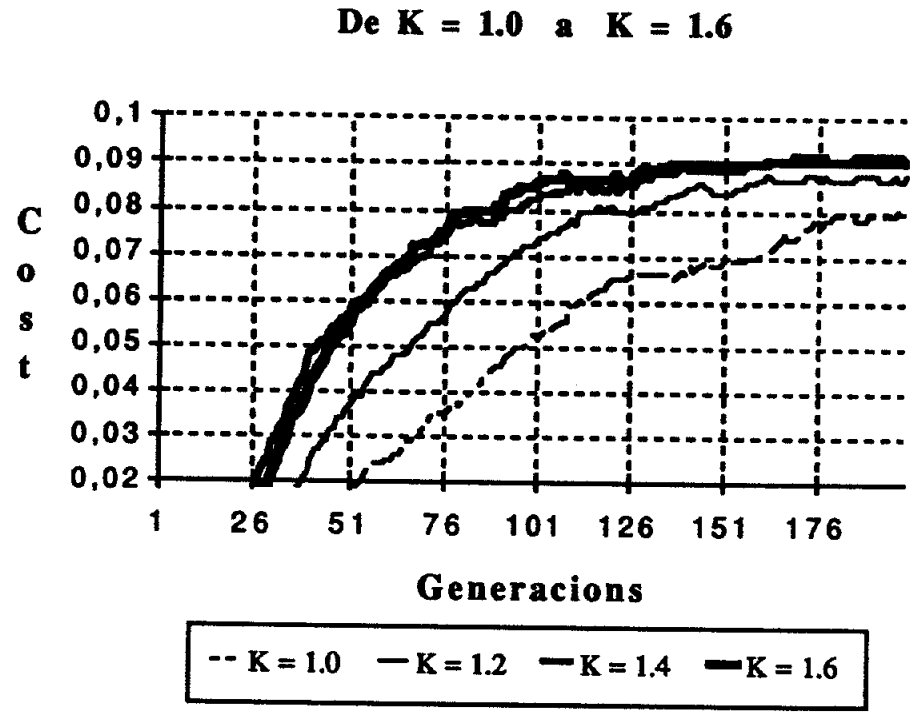


Figura 8.15

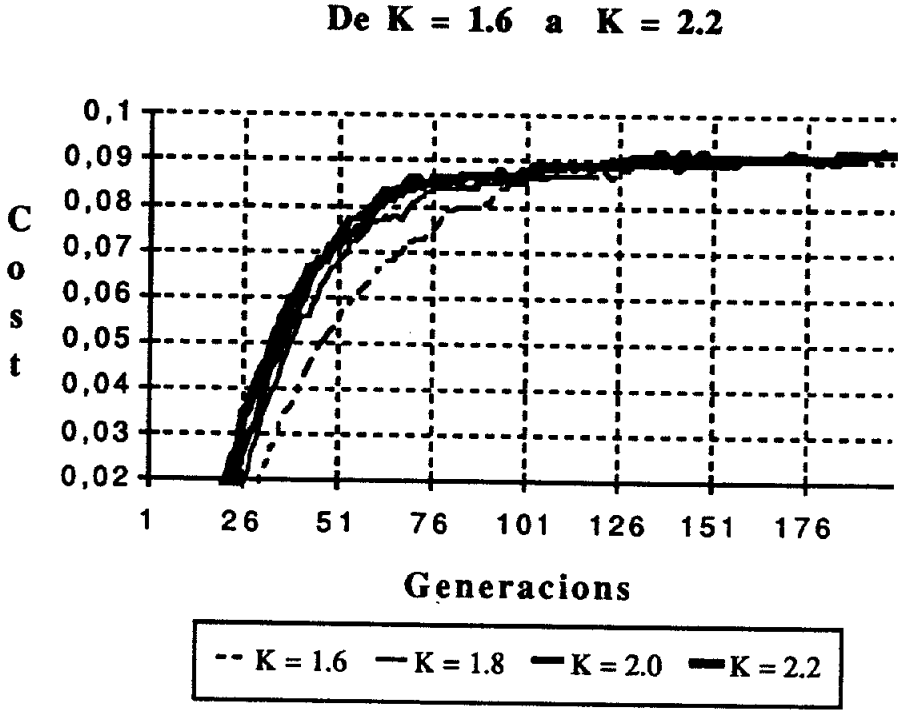


Figura 8.16

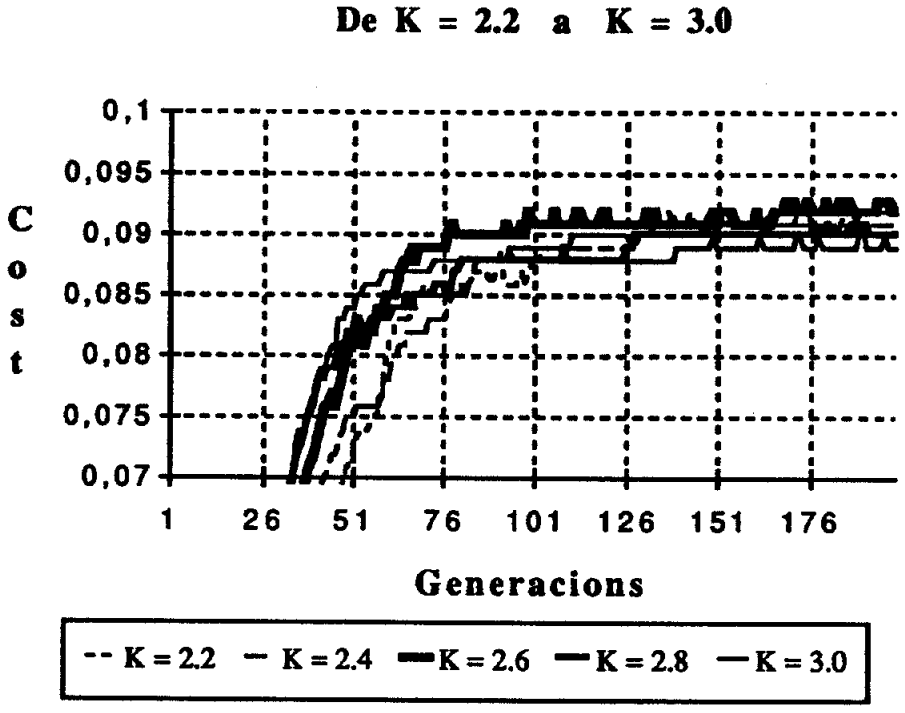

De K = 2.2 a K = 3.0
0,1
0,095
0,09
0,085
0,08
0,075
0,07
Cost
1
26
51
76
101
126
151
176
Generacions
K = 2.2
K = 2.4
K = 2.6
K = 2.8
K = 3.0

Figura 8.17

8.4 Resultats obtinguts.

Dividirem els resultats obtinguts en dos grups. En un primer grup hi inclourem els codis de pes constant trobats i en un segon grup hi inclourem els codis de cobertura.

En general hi han hagut dos formes d'atacar el problema segons la seva complexitat. Quant el problema semblava ser senzill, volem dir que el codi tenia poques paraules i per tant el temps d'execució era ràpid, intentàvem obtenir directament una bona fita i si veiem que no era així ens aproximàvem lentament per no perdre massa temps d'execució en el cas de que els resultats no fossin favorables.

Els resultats seran presentats en forma de taula i els codis seran inclosos en un anex posterior.

8.4.1 Codis de pes constant.

Com ja hem discutit anteriorment, per motius pràctics hem decidit trobar uns codis que ja havien estat trobats per altres autors. D'aquesta forma podíem tenir un bon criteri al valorar els nostres resultats. En aquest cas els codis que s'han intentat trobar han estat trets de El Gamal *et al.* [1].

Un cop realitzades les execucions de l'A.G. per trobar els codis presentem els resultats obtinguts a la següent taula.(taula 8.1). En els casos en que hem aconseguit millorar la fita anterior donada per El Gamal *et al.* ho indiquem amb la paraula **millora**. Quant hem igualat els seus resultats ho indiquem amb la paraula **igualat.** Si només ens hem aproximat, però no hem arribat a la fita ho indiquem amb la paraula **aproximat**.. I en els casos en els que hem intentat trobar el codi directament sense aproximar-nos i no ho hem aconseguit ho indicarem amb **no aconseguit.**

Codi	Simulated Ann.	Algor. Genetic	Valoració
A(21,10,9)	22 paraules	22 paraules	**igualat**
A(22,10,9)	23 paraules	25 paraules	**superat**
A(23,10,7)	18 paraules	17 paraules	aproximat
A(23,10,8)	28 paraules	----------	no aconseguit
A(23,10,9)	24 paraules	25 paraules	**superat**
A(23,10,10)	39 paraules	----------	no aconseguit
A(23,10,11)	39 paraules	39 paraules	**igualat**
A(24,10,8)	33 paraules	----------	no aconseguit
A(24,10,9)	24 paraules	27 paraules	**superat**

Taula 8.1

Com a resultat podem veure com dels nou codis que hem intentat trobar hem superat tres i hem igualat dos. Els codis no aconseguits no s'han intentat aproximat degut a que eren de gran mida. L'intent d'aproximar-nos hagués estat una despesa de temps de computació no profitós.

Hem de tenir en compte que una simulació (població 200 i 3000 generacions) evalua 600.000 vegades la funció de cost. A més el normal és que es realitzin diverses execucions per trobar el codi. Això comporta un temps de computació molt gran que no es pot desaprofitar. Si hem intentat trobar el codi i no l'hem trobat no hem cregut necessari emprar temps de computació en trobar un resultat pitjor.

8.4.2 Codis de cobertura

Els codis de cobertura trobats per altres autors són insuficients per valorar la bondat del mètode. Només disposem de quatre codis trobats per Patric. J. Östergard [31], [32] i [33] . A més aquests quatre codis podriem considerar-los de tres classes diferents. Tenim dos codis q-aris de cobertura clàssica, un codi binari de cobertura mitjançant una matriu, i un codi mixt. Amb això està clar que no tenim prou per valorar el funcionament del nostre algorisme. Per això el nostre algorisme s'ha comprovat intentant trobar codis de curta mida del que ja es coneixia la fita teòrica.

Dividirem els nostres resultats en dos grups. En el primer presentarem els resultats d'intentar trobar els codis trobats per Östergard i en el segon presentarem els codis trobats de curta mida.

Codis trobats per Östergard:

Codi	Simulated Ann	Algorisme Gen.	Valoració
(5,4,11) R=2	11 paraules	11 paraules	**Igualat**
(5,5,35) R=2	35 paraules	----------	No aconseguit
r=11, n=12 , R=2	39 paraules	----------	No aconseguit
$F_4^1F_2^7$ R=1	60 paraules	65 paraules	No aconseguit

Taula 8.2

Codis curts:

El nostre algorisme ha trobat tots els codis de menys de setze paraules i Radi de cobertura 1, de dotze paraules i radi 2 i de set paraules i radi 3 que ha intentat. Si no s'han intentat codis de major longitud ha estat perquè ja ens suposava un temps de càlcul més gran. En tots els casos s'ha arribat a la fita inferior prevista per la teoria.

Els codis trobats són els següents:

K(n,R)	R=1	R=2	R=3
n =2	2 paraules	----------	----------
n =3	2 paraules	2 paraules	----------
n =4	4 paraules	2 paraules	2 paraules
n =5	7 paraules	2 paraules	2 paraules
n =6	12 paraules	4 paraules	2 paraules
n =7	16 paraules	7 paraules	2 paraules
n =8	----------	12 paraules	4 paraules
n =9	----------	----------	7 paraules

Taula 8.3

Els codis en el que no s'indica les paraules cobertes és per que no s'ha intentatat trobar o eren massa senzills.

8.5 Valoració de resultats

Una vegada coneguts els resultats trobats en aquest projecte podem fer-ne una valoració. En primer lloc cal dir que hem assolit amb èxit la nostra fita principal, que era l'aplicació dels algorismes genètics a l'obtenció de bons codis.

Tant els operants, en el cas de paraules de pes constant, (mutació i encreuament) com la funció de cost emprada - l'escalat exponencial - (que ha mostrat ser més potent que els escalats existents fins ara), són noves aportacions d'aquest projecte. Aquests operants i la funció de cost signifiquen un nou avenç en el camp dels A.G. i tant els nous tipus d'encreuament i mutació utilitzats, com, sobretot, l'escalat exponencial poden ser aplicats per altres investigadors a resoldre nous problemes.

Hem de remarcar, en especial, el fet que l'escalat exponencial introduit en aquest projecte no només és més potent, sinó que a més té diferents avantatges sobre la resta d'escalats emprats fins ara, en les aplicacions d'A.G., com s'ha descrit en el capítol 7.

Els resultats obtinguts es comparen molt favorablement amb els trobats per altres autors amb recuita simulada.

Dels nou codis de pes constant que hem trobat tres superaven als anteriors obtinguts amb recuita simulada i dos igualaven la seva fita. A més dels quatre codis de cobertura presentats un igualava la fita anterior. En total dels tretze codis, hem aconseguit com a mínim igualar en fita sis. A més hem aconseguit divuit codis de cobertura curts òptims (degut a que la seva mida era igual a la seva fita mínima). Aquest resultat hem de considerar-lo molt bo degut a que els codis que teníem com a referència no eren codis qualsevol sinó que eren fites dins dels codis. Això feia que fossin molt difícils de trobar.

La importància dels mètodes emprats també ve donada pel fet que s'han aconsseguit bons resultats malgrat tenir dos handicaps importants en la seva aplicació.

Per una part sabem que degut a l'aleatoritat inherent en un algorisme genètic teniem només una certa probabilitat de trobar els codis. Els diferents autors publiquen només els bons resultats obtinguts

dins del total que proven, que són molts més. Degut a això, podem considerar el fet de tornar a trobar un codi que és fita dins de la seva classe, com un fet molt remarcable.

Per altra part hem de tenir en compte que aquest problema en particular no era un problema fàcil de resoldre mitjançant un A.G. En aquest problema un mateix codi pot ser rescrit de moltes maneres diferents. Si canviem d'ordre les paraules del codi, aquest continua sent el mateix codi. Igualment passa amb les columnes. Podem dir que per un codi de p paraules de longitud n tenim p! x n! formes diferents de representar-lo.

Això produeix que l'encreuament, considerat com el factor distintiu de l'algorisme genètic, pot no treballar adequadament. La raó és que podem creuar un bon codi amb si mateix i no obtenir una bona solució. Per exemple l'encreuament de qualsevol codi amb ell mateix, però amb les paraules ordenades en ordre invers, donarà amb tota seguretat un mal codi degut a que tindrà com a mínim una paraula repetida.

Aquest problema, en canvi, no afecta a la recuita simulada. La recuita simulada no treballa amb una població de punts sinó que modifica en un bit un únic codi i decideix quin és el millor (l'actual, o l'anterior) segons certes regles. Així fa evolucionar el codi fins arribar a un bon resultat.

Podem dir, com a resum, que hem trobat una manera d'aplicar el nostre algorisme genètic a l'obtenció de codis amb èxit, hem obtingut avanços importants en la tècnica dels algorismes genètics i malgrat no ser un tipus de problema fàcil pel nostre A.G., s'ha aconseguit treure uns resultats totalment comparables als millors resultats obtinguts pels altres autors amb la recuita simulada.

L'avantatge evident sobre la recuita simulada està en que els algorismes genètics són paral.lelitzables directament, i per tant les futures generacions de màquines que ja comencen a trobar-se a l'abast dels investigadors, poden emprar-los amb un esforç mínim de reprogramació.

A.1 Codis Trobats

A continuació farem un llistat amb els diferents codis trobats que hem descrit en el capítol anterior. Primer mostrarem els codis de pes constant i després mostrarem els de cobertura. Juntament amb els codis escriurem les variables d'execució amb els que s'han trobat.

A.1.1 Codis pes constant

Codi A(21,10,9) trobat:

- Nombre de paraules:	**22**
- Longitut de la paraula:	**21**
- Pes de la paraula:	**9**
- Distància mínima:	**10**
- Mida de la població:	**200**
- Nombre de generacions:	**1365**
- Probabilitat de mutació:	**0.6**
- Probabiltat de creuament:	**0.6**
- Factor K de l'escalat:	**2.0**

1. 111111111000000000000
2. 010001001100000110111
3. 010001110010101000110
4. 010110001001101000101
5. 111001000011000101010
6. 100000111100010101100
7. 001010001110001001110
8. 001000111011010000011
9. 100110010111100000010
10. 101110000000010100111
11. 101001010000001011101
12. 110010100001011010010
13. 000110101000100111010
14. 011010010100100101001
15. 000101010001110110100
16. 000000100111100011101

17. 001011100110010110000
18. 010100011010011011000
19. 000101000101011001011
20. 101000001101101110000
21. 100011001010110001001
22. 110100100110001100001

Codi (22,10,9) trobat:

- Nombre de paraules: **25**
- Longitut de la paraula: **22**
- Pes de la paraula: **9**
- Distància mínima: **10**
- Mida de la població: **200**
- Nombre de generacions: **1055**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 1111111110000000000000
2. 0011001101101000000011
3. 0100000100111010100110
4. 1010100100010001101010
5. 1000011101110100010000
6. 0101011000001100101100
7. 0110100011100011000010
8. 0101001001010001010110
9. 1000110101000010100101
10. 0000101001011110001010
11. 1001000011110000101100
12. 1110001000100001001101
13. 1100000000010111110001
14. 1011000001001101011000
15. 0100101011000000111001
16. 0001100110001101000101
17. 0100010111011001001000
18. 0011100100100110110000
19. 0010001010111101100000

20. 011101000000010011011
21. 100111000011011000000
22. 100011001000110011010
23. 000011101011000000111
24. 001000111001001011100
25. 000001000110010110101 1

Codi A(23,10,9) trobat:

- **Nombre de paraules: 25**
- **Longitut de la paraula: 23**
- **Pes de la paraula: 9**
- **Distància mínima: 10**
- **Mida de la població: 200**
- **Nombre de generacions: 91**
- **Probabilitat de mutació: 0.6**
- **Probabiltat de creuament: 0.6**
- **Factor K de l'escalat: 2.0**

1. 11111111100000000000000
2. 00010101010011000001101
3. 00110010010100001110001
4. 01100000011001000111100
5. 10100001101011000000110
6. 01100110001010001010010
7. 10010110000000110010011
8. 00111000100101100000101
9. 00000000110010011011110
10. 00001010001001011101010
11. 10100101000001101011000
12. 11001000100001110110000
13. 01100010100010010101001
14. 00000000111011101100001
15. 01011001001010100010100
16. 10011010010010000100110
17. 00001011010110110000001
18. 10010100101110010100000

19. 111000010111000110000000
20. 111100000000000001001111
21. 000111010001001011000010
22. 100000010011000000111011
23. 110011000101010000001010
24. 010001011000010110000011
25. 001011000110000100000111

Codi A(23,10,11) trobat:

- **Nombre de paraules: 39**
- **Longitut de la paraula: 23**
- **Pes de la paraula: 11**
- **Distància mínima: 10**
- **Mida de la població: 200**
- **Nombre de generacions: 854**
- **Probabilitat de mutació: 0.6**
- **Probabiltat de creuament: 0.6**
- **Factor K de l'escalat: 2.0**

1. 11111111111000000000000
2. 10011101000110110100001
3. 11110000000100011110011
4. 11000100001101101010101
5. 01010001001000110111110
6. 01000011110001011110100
7. 00010111011001000001111
8. 10000010000111110001111
9. 01010010100111101101000
10. 10000100110101011101010
11. 10011000011100111001100
12. 01101101010000110010101
13. 10001010110000110110011
14. 10110110100010001011001
15. 00011100111100000110101
16. 00101110011011001100100
17. 10010100111011100010010

18. 011001000011111110100010
19. 011011100001000011011110
20. 010110110101100000110010
21. 010010011101011000011110
22. 100110110010010011010010
23. 001110010001110100111100
24. 001111001000011100011011
25. 111000100110110100111000
26. 110000010010110011010111
27. 001000001110001010111111
28. 110110001100110010000101
29. 001101110101011010100000
30. 101110001010100101001100
31. 001000111111100110000010
32. 101010110011000000101101
33. 110011000100101011110000
34. 010111100010001011000011
35. 101001011010010101100001
36. 110001111000101010001100
37. 011010001101100101010001
38. 001010011000111011100010
39. 011100011011101000000101

Codi A(24,10,9) trobat:

- Nombre de paraules:	**27**
- Longitut de la paraula:	**24**
- Pes de la paraula:	**9**
- Distància mínima:	**10**
- Mida de la població:	**200**
- Nombre de generacions:	**85**
- Probabilitat de mutació:	**0.6**
- Probabiltat de creuament:	**0.6**
- Factor K de l'escalat:	**2.0**

1. 111111111000000000000000
2. 010000001001011001001110

3. 101000001011000100011010
4. 001011010110100010100000
5. 000110100101000010011001
6. 000001111000011011100000
7. 010100110011001000010010
8. 010011000100110001010010
9. 001000000011010011100101
10. 100101000010000001110011
11. 000001100110110100001100
12. 000101011100001100000110
13. 010000000101100110101010
14. 100010010110010000000111
15. 011000101100001110010000
16. 101001100001100001010100
17. 010111000000010000101101
18. 000010001100000101111100
19. 000100010000111100011001
20. 100100000111001111000000
21. 000000101010101100100011
22. 100110001000110110000100
23. 111000100010110010000010
24. 001011010001000101000011
25. 101100100100101000101000
26. 110000011000101000110100
27. 000011001011001010010100

A.1.2 Codis de cobertura.

Codi (5,4,11) R=2 trobat:

- **Nombre de paraules: 11**
- **Longitut de la paraula: 4**
- **Base de la paraula: 5**
- **Radi de cobertura: 2**
- **Mida de la població: 100**
- **Nombre de generacions: 48**
- **Probabilitat de mutació: 0.3**
- **Probabiltat de creuament: 0.3**
- **Factor K de l'escalat: 2.0**

1. 2342
2. 4234
3. 1220
4. 0343
5. 3101
6. 1100
7. 0122
8. 4411
9. 3414
10. 1030
11. 2003

Codi K(2,1) trobat:

- **Nombre de paraules: 2**
- **Longitut de la paraula: 2**
- **Base de la paraula: 2**
- **Radi de cobertura: 1**
- **Mida de la població: 200**
- **Nombre de generacions: 0**
- **Probabilitat de mutació: 0.6**
- **Probabiltat de creuament: 0.6**
- **Factor K de l'escalat: 2.0**

1. 00
2. 01

Codi K(3,1) trobat:

- **Nombre de paraules: 2**
- **Longitut de la paraula: 3**

- Base de la paraula: **2**
- Radi de cobertura: **1**
- Mida de la població: **200**
- Nombre de generacions: **0**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 000
2. 111

Codi K(4,1) trobat:

- Nombre de paraules: **4**
- Longitut de la paraula: **4**
- Base de la paraula: **2**
- Radi de cobertura: **1**
- Mida de la població: **200**
- Nombre de generacions: **0**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 0000
2. 1010
3. 0111
4. 01101

Codi K(5,1) trobat:

- Nombre de paraules: **7**
- Longitut de la paraula: **5**
- Base de la paraula: **2**
- Radi de cobertura: **1**
- Mida de la població: **200**
- Nombre de generacions: **9**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 00000
2. 01101
3. 11010
4. 10110
5. 10001
6. 11110

7. 00011

Codi K(6,1) trobat:

- Nombre de paraules: **12**
- Longitut de la paraula: **6**
- Base de la paraula: **2**
- Radi de cobertura: **1**
- Mida de la població: **200**
- Nombre de generacions: **157**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 000000
2. 111001
3. 010111
4. 101100
5. 111010
6. 000010
7. 000001
8. 110100
9. 111011
10. 001111
11. 100111
12. 011100

Codi K(7,1) trobat:

- Nombre de paraules: **16**
- Longitut de la paraula: **7**
- Base de la paraula: **2**
- Radi de cobertura: **1**
- Mida de la població: **200**
- Nombre de generacions: **52**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 0000000
2. 0100110
3. 1011001
4. 0010011
5. 1001010
6. 1010110
7. 0110101

8. 1111111
9. 0011100
10. 1110000
11. 0101001
12. 1000101
13. 1101100
14. 0111010
15. 1100011
16. 0001111

Codi K(3,2) trobat:

- Nombre de paraules: **2**
- Longitut de la paraula: **3**
- Base de la paraula: **2**
- Radi de cobertura: **2**
- Mida de la població: **200**
- Nombre de generacions: **0**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 000
2. 100

Codi K(4,2) trobat:

- Nombre de paraules: **2**
- Longitut de la paraula: **4**
- Base de la paraula: **2**
- Radi de cobertura: **2**
- Mida de la població: **200**
- Nombre de generacions: **0**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 0000
2. 1110

Codi K(5,2) trobat:

- Nombre de paraules: **2**
- Longitut de la paraula: **5**
- Base de la paraula: **2**
- Radi de cobertura: **2**

- Mida de la població: **200**
- Nombre de generacions: **0**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 00000
2. 11111

Codi K(6,2) trobat:

- Nombre de paraules: **4**
- Longitut de la paraula: **6**
- Base de la paraula: **2**
- Radi de cobertura: **2**
- Mida de la població: **200**
- Nombre de generacions: **1**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 000000
2. 101000
3. 110111
4. 011111

Codi K(7,2) trobat:

- Nombre de paraules: **7**
- Longitut de la paraula: **7**
- Base de la paraula: **2**
- Radi de cobertura: **2**
- Mida de la població: **200**
- Nombre de generacions: **137**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 0000000
2. 0011111
3. 1101100
4. 1110101
5. 0111011
6. 1011011
7. 1100110

Codi K(4,3) trobat:

- Nombre de paraules: **2**
- Longitut de la paraula: **4**
- Base de la paraula: **2**
- Radi de cobertura: **3**
- Mida de la població: **200**
- Nombre de generacions: **0**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 0000
2. 0100

Codi K(5,3) trobat:

- Nombre de paraules: **2**
- Longitut de la paraula: **5**
- Base de la paraula: **2**
- Radi de cobertura: **3**
- Mida de la població: **200**
- Nombre de generacions: **0**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 00000
2. 11010

Codi K(6,3) trobat:

- Nombre de paraules: **2**
- Longitut de la paraula: **6**
- Base de la paraula: **2**
- Radi de cobertura: **3**
- Mida de la població: **200**
- Nombre de generacions: **0**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 000000
2. 011111

Codi K(7,3) trobat:

- Nombre de paraules: **2**
- Longitut de la paraula: **7**
- Base de la paraula: **2**
- Radi de cobertura: **3**
- Mida de la població: **200**
- Nombre de generacions: **1**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 0000000
2. 1111111

Codi K(8,3) trobat:

- Nombre de paraules: **4**
- Longitut de la paraula: **8**
- Base de la paraula: **2**
- Radi de cobertura: **3**
- Mida de la població: **200**
- Nombre de generacions: **11**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 00000000
2. 01101011
3. 11110111
4. 10011100

Codi K(9,3) trobat:

- Nombre de paraules: **7**
- Longitut de la paraula: **9**
- Base de la paraula: **2**
- Radi de cobertura: **3**
- Mida de la població: **100**
- Nombre de generacions: **128**
- Probabilitat de mutació: **0.6**
- Probabiltat de creuament: **0.6**
- Factor K de l'escalat: **2.0**

1. 000000000
2. 110111011
3. 101000000
4. 011101111
5. 100010101
6. 110111110
7. 001010000

A.2 Algorismes implementats

En aquest annex es presenten els llistats dels algorismes imlementats i que es troben a disposició del que els vulgui utilitzar. Els programes implementats es denominen nou.c cover.c coverM.c i coverf4.c. Degut a que moltes parts d'aquests algorismes són iguals no inclourem la totalitat dels procediments, deixant indicat únicament la capçalera.

A.2.1. nou.c

```
/*****************************************************/
/*                    NATURA                         */
/*                Algoritme Genetic                  */
/*             per codigs de pes constant            */
/*****************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#define RANDOM(c) ((int) (rand() /2147483647.0*(c)))

/* Constants relacionades amb A.G. */

#define NUMCOD      14          /* Nombre de paraules del codi.    */
#define LONGCOD     23          /* Dimensio de la paraula .        */
#define PES          7          /* Pes constant de la paraula.     */

#define MAXGEN      200         /* Nombre maxim de generacions */
#define MAXPOP      200         /* Tamany de la poblacio.          */
#define MAXITER       5         /* Nombre d'iteracions amb cada
                                 variable d'execucio del programa   */

/* Variables d'execucio del programa */

#define PMUTINIC    0.5         /* La probabilitat de mutacio varia */
#define PMUTFIN     0.5         /* de PMUTINIC a PMUTFIN amb */
#define PMUTINC     0.2         /* increments de  PMUTINC          */

#define PCROSSINIC  0.5         /* El mateix pero amb la           */
#define PCROSSFIN   0.5         /* probabilitat de creuament       */
#define PCROSSINC   0.2
```

```
#define KINIC        1.0          /* Variacions del factor d'excalat  */
#define KFIN         3.0          /* exponencial                      */
#define KINC         0.2

#define NOMFITX      "nou14K"     /* Nom basic del fitxer de sortida  */

/* Formes d'execucio del programa */

#define PANTALLA     FALS         /* Sortiran dades per pantalla  */
#define RESULTAT     FALS
/* El programa acabara quant trobi un resultat. */
/* NOTA: La ultima estadistica no es podra acabar.      */

/* Constants utils */

#define CERT      1
#define FALS      0
#define NL        printf("\n")

/* Definicio de tipus */

typedef unsigned char BOOL;
typedef BOOL ALLELE;
typedef ALLELE CROMOSOMA[ NUMCOD ][ LONGCOD ];
/* Conjunt de pararules codi */


typedef struct {                    /* Definicio del tipus individu     */
      CROMOSOMA chrom;
      double  fitness;
      } INDIVIDUAL;

typedef INDIVIDUAL POBLACIO[ MAXPOP ]; /* Conjunt d'individus
que formen una poblacio.*/

typedef struct {
      float max;
      float avg;
      float min;
      } DADESGEN;              /* Dades estadistiques d'una generacio.
                                  Es el promig de MAXITER generacions. */


typedef DADESGEN DADESPOP[ MAXGEN ]; /* Estadistica de totes les
                                        generacions.*/
```

```
/* Aqui venen les variables */

POBLACIO oldpop,newpop;
DADESPOP  estadist;
int       gen,millor,mot,dmin,cont,iter;
char      fitxerdad[20],fitxercod[20];
float max,min,avg,prob,pmut,pcross,kprob;
double    sumfitness,cost,millorcost;
FILE   *dades,*codi;

/* Ara ve el programa */

/*****************************************************/

int flip()

/* Torna cert si es cara amb probabilitat prob */

{
if ((prob)==1.0)
   return(CERT);
else
    return(  (RANDOM(10000)) < (int)(prob*10000.0) );
}

/*****************************************************/

int select(pop)
POBLACIO pop;

/* Selecciona un individu de la poblacio depenent de la seva funcio
de cost mitjancant el metode de la ruleta de la sort. */

{
float randp;
double   partsum=0;
int   j=0;

randp=(float)RANDOM(10000)/10000.0;
randp=randp*sumfitness;
do {
    partsum =partsum+pop[j].fitness;
  j++;
  }
   while ((partsum <randp)&& (j !=MAXPOP));
```

```
return (j-1);
}

/*************************************************/

void  crossover(father,mother,son,daughter)
CROMOSOMA  father,mother,son,daughter;

/* Creuem dos individus amb probabilitat pcross */

{
int i,c,
    lloc1,lloc2,pos,
   j=0;

prob=pcross;
if ( flip() )
  {
   mot=RANDOM(NUMCOD);      /* Aquesta es la paraula per on tallem
*/
  for (i=0; i<LONGCOD; i++)
     {
      for (c=0; c<mot;c++)
         {
          son[c][i]=father[c][i];
           daughter[c][i]=mother[c][i];
         }
     for (c=mot; c<NUMCOD;c++)
         {
          son[c][i]=mother[c][i];
           daughter[c][i]=father[c][i];
         }
      }
  }
    else
     for (i=0;i<LONGCOD;i++)
         for (c=0;c<NUMCOD;c++)
              {
                son[c][i]=father[c][i];
                 daughter[c][i]=mother[c][i];
              }
}

/*************************************************/

void  mutation(chromos,mutchromos)
```

```
CROMOSOMA chromos,mutchromos;

/* Cambiem un bit del codi amb probabilitat pmut */

{
int  i,c,lloc,b;

for (c=0;c<NUMCOD;c++)
   for (i=0; i<LONGCOD;i++)
         mutchromos[c][i]=chromos[c][i];

prob=pmut;
if ( flip() )
   {
  c=RANDOM(NUMCOD);
  lloc=RANDOM(LONGCOD);
    b=chromos[c][lloc];
    mutchromos[c][lloc]= !(b);
  do { lloc=RANDOM(LONGCOD);
       } while(chromos[c][lloc]==b);
    mutchromos[c][lloc]=b;
   }
}

/*****************************************************/

void  calc_objfunc(crom)
CROMOSOMA crom;

/* Calcul de la funcio de cost.  */

{
int i,j,c,d;

dmin=LONGCOD;
cost=0.0;
for (i=0 ; i<(NUMCOD-1) ; i++)
   for (j=i+1 ; j<NUMCOD ;j++)
       {
        d=0;
      for (c=0;c<LONGCOD;c++)
           if (crom[i][c] !=crom[j][c]) d++ ;
         if (dmin>d) dmin=d;
        if (d>10) d=10;
           cost+=1/((float)((d*d)+1)); /* Evitem divisions per zero */
       }
```

```
cost=(1-cost);
}

/*************************************************/

void  generation()
/* Creem un nova generacio d'igual tamany que l'anterior */
{
int  i,c,mate1,mate2,j=0;
CROMOSOMA  parent1,parent2,child1,child2,
          mutchild1,mutchild2;

do {
     /* Seleccionem dos cromosomes diferents */
     mate1=select(oldpop);
     do {   mate2=select(oldpop); } while(mate2==mate1);

    /* Els creuem */

crossover(oldpop[mate1].chrom,oldpop[mate2].chrom,child1,child2);

    /* Els mutem  */
      mutation(child1,newpop[j].chrom);

     /* I calculem quina es la seva funcio de cost. */
      mutation(child2,newpop[j+1].chrom);
     calc_objfunc(newpop[j].chrom);
     newpop[j].fitness=cost;
     calc_objfunc(newpop[j+1].chrom);
     newpop[j+1].fitness=cost;
   j+=2;
  }
while (j<MAXPOP); /* Aixi fins que els tinguem tots */
}


/*************************************************/

void  statistic(pop)
POBLACIO pop;

/* Aquest procediment calcula el maxim,el minim i la mitja de la
funcio de cost dels individus de la poblacio. A mes calcula la suma
de
tots els costos i fa l'escalat de la funcio. */
```

```
{
int j,ord[MAXPOP],surt;
float k;

millor=0;
max=min=sumfitness=pop[0].fitness;
for (j=1;j<MAXPOP;j++)
   {
       sumfitness+=pop[j].fitness;
      if (pop[j].fitness > max )
      { max=pop[j].fitness;millor=j;}
     if (pop[j].fitness < min ) min=pop[j].fitness;
   }
avg=sumfitness/MAXPOP;

/* Escalat LINIAL : Ordena els costos de menor a major i li donem
com a
nova funcio de cost el lloc que ocupa .



for (j=0;j<MAXPOP;j++) ord[j]=j;
do {
    surt=0;
   for (j=0;j<(MAXPOP-1);j++)
          if (pop[(ord[j])].fitness>pop[(ord[j+1])].fitness)
          {
              surt=ord[j];ord[j]=ord[j+1];ord[j+1]=surt;
            surt=1;
          }
    } while (surt !=0);
 sumfitness=0;
 for (j=0;j<MAXPOP;j++)
    {
      pop[(ord[j])].fitness=j+1;
      sumfitness+=j+1;
    }*/

 /* ESCALAT  EXPONENCIAL  : Fem un escalat exponencial. Aixi la
 probabilitat  del  maxim  respecte  de  la  mitja  es  de  exp(k).
 prob(max)/prob(avg)=exp(k)  */

 k=kprob/(max-avg);
 sumfitness=0;
 for (j=0;j<MAXPOP;j++)
    {
```

```
        pop[j].fitness=exp(k*(pop[j].fitness-avg));
        sumfitness+=pop[j].fitness;
    }

/* ESCALAT AL MINIM : Baixem la funcio al nivell zero.

sumfitness=0;
for (j=0;j<MAXPOP;j++)
    {
        pop[j].fitness=pop[j].fitness - min;
        sumfitness+=pop[j].fitness;
    } */
}

/*************************************************/

void inicializa(oldpop)
POBLACIO oldpop;
/* Inicialitzem la poblacio aleatoriament i calculo la seva
funcio de cost. */
{
int j,c,p,lloc;
char    num[6];

for (j=0; j<NUMCOD; j++)
    for (c=0; c<LONGCOD; c++)
        for (p=0; p<MAXPOP; p++)
            oldpop[p].chrom[j][c]=0;

for (p=0; p<MAXPOP; p++)
    for (j=1; j<NUMCOD; j++)
        for (c=0; c<PES; c++)
            {
            do { lloc=RANDOM(LONGCOD);
                } while( oldpop[p].chrom[j][ lloc ]==1);
            oldpop[p].chrom[j][lloc]=1;
            }
/* Poso la primera pararula constat */
for (p=0; p<MAXPOP; p++)
    for ( c=0; c<PES; c++)
        oldpop[p].chrom[0][c]=1;

for (p=0; p<MAXPOP; p++)
    {
```

```
    calc_objfunc(oldpop[p].chrom);
    oldpop[p].fitness=cost;
  }


}

/*************************************************/

void inicifitxer()

/* Inicialitzem el fitxer de sortida de dades i l'estadistica
que hi sortira */

{
int gene;
char   num[6];

strcpy(fitxerdad,NOMFITX);  /* Afegim als noms dels fitxers les
                               terminacions  */
strcpy(fitxercod,NOMFITX);  /* adients i a mes el numero d'execucio
                               al fitxer  de dades. */
sprintf(num,"%d",cont);
strcat(fitxerdad,num);
strcat(fitxerdad,".dad");
strcat(fitxercod,".cod");
dades=fopen(fitxerdad,"w");
fprintf(dades,"
/*************************************************\n");
fprintf(dades,"        /                NATURA                        \n");
fprintf(dades,"         /             fitxer de resultats              \n");
fprintf(dades,"
/*************************************************\n");
fprintf(dades,"\n");
fprintf(dades,"              PMUTATION= %2.4f     PCROSS=%2.4f
MAXPOP=%d    MAXITER=%d\n",pmut,
pcross,MAXPOP,MAXITER);
fprintf(dades,"\n");
fprintf(dades,"                  NUMCOD= %d       LONGCOD=%d
KPROB=%2.2f\n",NUMCOD,LONGCOD,kprob);
fprintf(dades,"\n");
fclose(dades);

for (gene=0;gene<MAXGEN;gene++)      /* Inicialitzo l'estadistica */
   {
     estadist[gene].avg=0;
```

```
    estadist[gene].max=0;
    estadist[gene].min=0;
  }
}

/*************************************************/

void reporta()

/* Reportem per pantalla i/o fitxer les dades que hem trobat */

{
int i,c;

estadist[gen].avg+=avg/MAXITER;     /* Afeguim dades a l'estadistica
*/
estadist[gen].min+=min/MAXITER;
estadist[gen].max+=max/MAXITER;

if (PANTALLA)
  {
   printf("\n");
   printf("      GEN = %4d    MAX= %9.3f    MIN =%9.3f    AVG=%9.3f
DMIN=%d \n"
       ,gen,max,min,avg,dmin);
  }


if ((gen==(MAXGEN-1))&&(iter==(MAXITER-1)))
  {
    dades=fopen(fitxerdad,"a");
  for (i=0;i<MAXGEN;i++)
     {
       fprintf(dades,"\n");
        fprintf(dades,"      GEN = %4d    MAX= %9.3f    MIN =%9.3f
AVG=%9.3f  \n"
       ,i,estadist[i].max,estadist[i].min,estadist[i].avg);
     }
   fclose(dades);
  }

calc_objfunc(oldpop[millor].chrom);

if ((cost > millorcost )||(dmin>8)) /* sempre que troba un codi millor
el guarda */
  {
```

```
    millorcost=cost;
    codi=fopen(fitxercod,"w");
    fprintf(codi,"
/*****************************************************\n");
   fprintf(codi,"      /                NATURA                    /\n");
   fprintf(codi,"      /             Codi de major cost.          /\n");
    fprintf(codi,"
/*****************************************************\n");
    fprintf(codi,"\n");
   fprintf(codi,"      GENERACIO= %4d   COST=%9.3f    DMIN=%2d
KPROB=%2.2f\n",gen,cost,dmin,kprob);
    fprintf(codi,"\n");
    fprintf(codi,"              PMUTATION= %2.4f    PCROSS=%2.4f
MAXPOP=%d\n",pmut,
   pcross,MAXPOP);
    fprintf(codi,"\n");
  for (i=0;i<NUMCOD;i++)
     {
       fprintf(codi,"\n");
      fprintf(codi,"                    %2d. ",i+1);
    for (c=0;c<LONGCOD;c++)
          fprintf(codi,"%d",oldpop[millor].chrom[i][c]);
     }
   if (RESULTAT)          /*   condicio d'acabament del programa.*/
    {
     if (dmin>8)
       {
       gen =MAXGEN;
        iter=MAXITER;
        kprob=KFIN;
     if (PANTALLA)
         printf("\n\n\n\n\n      EUREKA !!!!!!!!!!. \n\n\n\n\n");
       }
    }
   fclose(codi);
  }
}

/*************************************************/
void main()
/* Programa principal */
{
int i,j,c;
long ara;

srand( time(&ara)%37 ); /* Poso la llavor aleatoria */
```

```
cont=0;                          /* Inicialitzo el contador d'execucions. */
millorcost=0;                    /* Inicialitzo el millor cost trobat */

 /* Execucions amb les diferents variables d'execucio. */

for (kprob=KINIC; kprob<(KFIN+(KINC/2));kprob+=KINC)
for                    (pcross=PCROSSINIC;
pcross<(PCROSSFIN+(PCROSSINC/2));pcross+=PCROSSINC)
     for ( pmut=PMUTINIC; pmut<(PMUTFIN+(PMUTINC/2)); pmut
+=PMUTINC )
     {
      inicifitxer();
     cont++;
      for ( iter=0; iter<MAXITER; iter++)
        {
         gen=0;
           inicializa(oldpop); /* Inicialitzo la poblacio  */
           statistic(oldpop);  /* Fem l'estadistica        */
          reporta();           /* Reportem el resultat     */

          /* Creem MAXGEN noves generacions */

        for (gen=1;gen<MAXGEN;gen++)
          {
              generation();   /* Trobem una nova generacio */
            /* Copiem la nova generacio a la vella . */
          for (i=0;i<MAXPOP;i++)
             {
            for (j=0;j<NUMCOD;j++)
              for (c=0;c<LONGCOD;c++)
                    oldpop[i].chrom[j][c]=newpop[i].chrom[j][c];
               oldpop[i].fitness=newpop[i].fitness;
             }
            statistic(oldpop);
           reporta();
          }
        }
     }
}

/************************************************/

/* I això,xò,xò,xò és tot amics !!!! */
```

A.2.2. cover.c

```
/*************************************************/
/*                NATURA                         */
/*            Algoritme Genetic                  */
/*         per codis de covertura.               */
/*************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#define RANDOM(c) ((int) (rand() /2147483647.0*(c)))

/* Constants relacionades amb A.G. */

#define BASE          5          /* Base del codi .                */
#define NUMCOD        11         /* Nombre de paraules del codi.
*/
#define LONGCOD       4          /* Dimensio de la paraula .       */
#define NUM_M         0          /* Definicio del tipus            */
#define NUMCOL        (NUM_M+LONGCOD) /* de la matriu de
covertura .    */

#define MAXGEN        200        /* Nombre maxim de generacions.
*/
#define MAXPOP        100        /* Tamany de la poblacio.         */
#define MAXITER       5          /* Nombre d'iteracions amb cada
variable
                                    d'execucio del programa       */

#define LONGESPAI     625        /* Longitud de l'espai de recerca
                                    =BASE^ LONGCOD                */
#define RADI          2          /* Radi de covertura del codi     */

/* Variables d'execucio del programa */

#define PMUTINIC      0.1        /* La probabilitat de mutacio varia
*/
#define PMUTFIN       0.9        /* de  PMUTINIC a PMUTFIN amb
*/
#define PMUTINC       0.2        /* increments de  PMUTINC
*/

#define PCROSSINIC    0.1        /* El mateix pero amb la          */
```

```
#define PCROSSFIN      0.9          /* probabilitat de creuament          */
#define PCROSSINC      0.2

#define KINIC          2.0          /* Variacions del factor d'excalat  */
#define KFIN           2.0          /* exponencial                      */
#define KINC           0.2

#define NOMFITX      "c5411"        /* Nom basic del fitxer de sortida
*/

/* Formes d'execucio del programa */

#define PANTALLA       FALS         /* Sortiran dades per pantalla
*/
#define RESULTAT       FALS         /* El programa acabara quant
trobi  */
/* un resultat. NOTA: La ultima estadistica no es podra acabar.
*/

/* Constants utils */

#define CERT      1
#define FALS      0
#define NL        printf("\n")

/* Definicio de tipus */

typedef unsigned char BOOL;
typedef BOOL ALLELE;
typedef ALLELE CODI[LONGCOD];
 /* definicio de paraula d'un codi    */
typedef CODI CROMOSOMA[ NUMCOD ];
/* Conjunt de paraules codi */
typedef int  ESPAI[LONGESPAI];        /* Espai de covertura  */
typedef CODI MATRIU[NUMCOL];          /* Matriu de covertura*/

typedef struct {                      /* Definicio del tipus individu */
      CROMOSOMA chrom;
      double  fitness;
      } INDIVIDUAL;

typedef INDIVIDUAL POBLACIO[ MAXPOP ];
                  /* Conjunt d'individus que formen una poblacio. */
typedef  struct  {
      float max;
      float avg;
```

```
        float min;
          } DADESGEN;                    /* Dades estadistiques d'una
generacio. Es el promig de MAXITER generacions. */

typedef DADESGEN DADESPOP[ MAXGEN ];
/* Estadistica de totes les generacions.  */

/* Aqui venen les variables */

POBLACIO oldpop,newpop;
DADESPOP estadist;
ESPAI    esp;
MATRIU   mat;
int      uno,gen,millor,cont,iter;
float    kprob,max,min,avg,prob,pmut,pcross;
double   sumfitness,cost,millorcost;
char     fitxerdad[20],fitxercod[20];
FILE   *dades,*codi;

/* Ara ve el programa */

/*************************************************/

int flip()

------------- Com en Nou.c -------------

int select(pop)

------------- Com en Nou.c -------------

void crossover(father,mother,son,daughter)

------------- Com en Nou.c -------------

void mutation(chromos,mutchromos)

------------- Com en Nou.c -------------

void calc_objfunc(crom)
CROMOSOMA crom;
/* Calcul de la funcio de cost. En aquest cas es el nombre de
paraules que cubreix el codi */
{
int n,e;
```

```
cost=0;
uno++;          /* "uno" es el valor de la marca de paraula coberta */

if (uno>32000)
  {
  for (e=0;e<LONGESPAI;e++) esp[e]=0;
  uno=1;
  }

for (n=0;n<NUMCOD;n++)
   {
      switch (RADI)   /* Anem cobrim l'espai amb cada distancia  */
         {
            case 3 : cubreix3(crom[n]);
            case 2 : cubreix2(crom[n]);
            case 1 : cubreix1(crom[n]);
            case 0 : cubreix0(crom[n]);
         }
   }


}
/*************************************************/

cubreix0(paraula)
CODI paraula;
/* Cobrim l'espai amb distancia zero */
{
int para;

para=val_dec(paraula);
if (esp[para]!=uno)  /* Cobrim l'espai si no esta ja cobert  */
     {
       esp[para]=uno;
     cost++;
     }
}

/*************************************************/

cubreix1(paraula)
CODI paraula;
/* Cobrim l'espai amb distancia u */
{
int c1,b1,para,c;
CODI par;
```

```
for (c1=0;c1<NUMCOL;c1++)    /* Trobem totes les paraules que estan
a */
    for (b1=1;b1<BASE;b1++) /* distancia u de "paraula"              */
      {
     for (c=0;c<LONGCOD;c++)
        par[c]=((mat[c1][c]*b1)+paraula[c])%BASE;
         para=val_dec(par);
        if (esp[para]!=uno)
        {
          esp[para]=uno;
        cost++;
        }
      }
}

/*************************************************/

cubreix2(paraula)
CODI paraula;
/* Cobrim l'espai amb distancia dos */
{
int  c1,c2,b1,b2,c,para;
CODI par;

for (c1=0;c1<(NUMCOL-1);c1++) /* Trobem totes les paraules que
estan a */
for (b1=1;b1<BASE;b1++)       /* distancia dos de "paraula"          */
   for (c2=c1+1;c2<NUMCOL;c2++)
    for (b2=1;b2<BASE;b2++)
      {
     for (c=0;c<LONGCOD;c++)
         par[c]=((mat[c1][c]*b1)+(mat[c2][c]*b2)+paraula[c])%BASE;
         para=val_dec(par);

        if (esp[para]!=uno)
        {
          esp[para]=uno;
        cost++;
        }
      }
}

/************************************************/

cubreix3(paraula)
```

```
CODI paraula;
/* Cobrim l'espai amb distancia tres */
{
int c1,c2,c3,b1,b2,b3,c,para;
CODI par;

for (c1=0;c1<(NUMCOL-2);c1++)      /* Trobem totes les paraules que
estan a */
for (b1=1;b1<BASE;b1++)              /* distancia tres de "paraula"
*/
   for (c2=c1+1;c2<(NUMCOL-1);c2++)
   for (b2=1;b2<BASE;b2++)
      for (c3=c2+1;c3<NUMCOL;c3++)
       for (b3=1;b3<BASE;b3++)
         {
        for (c=0;c<LONGCOD;c++)

par[c]=((mat[c1][c]*b1)+(mat[c2][c]*b2)+(mat[c3][c]*b3)+paraula[c])%
BASE;
            para=val_dec(par);

           if (esp[para]!=uno)
           {
             esp[para]=uno;
           cost++;
           }
         }
}

/*************************************************/

int  val_dec(para1)
CODI para1;
/* Troba el valor decimal que representa la paraula para1
en la base BASE */
{
int  c,sum,pow;

sum=0;
pow=1;
for (c=0;c<LONGCOD;c++)
   {
    sum +=(para1[c]*pow);
   pow*=BASE;
   }
return  (sum);
```

```
}

/*************************************************/

void  generation()

-------------- Com en Nou.c -------------

void  statistic(pop)

-------------- Com en Nou.c -------------

void  inicializa(oldpop)
POBLACIO oldpop;
/* Inicialitzem la poblacio aleatoriament i calculo la seva
funcio de cost. Tambe inicialitzo la matriu de covertura. */
{
int j,c,p;
char    num[6];

uno=32001;
for (c=0;c<NUMCOL;c++)
   {
  for (j=0;j<LONGCOD;j++) mat[c][j]=0;
   mat[c][c]=1;
   }

for (p=0; p<MAXPOP; p++)
   {
  for (c=0; c<LONGCOD; c++)
      for (j=0; j<NUMCOD; j++)
           oldpop[p].chrom[j][c]=RANDOM(BASE);
     calc_objfunc(oldpop[p].chrom);
     oldpop[p].fitness=cost;
   }
}

/*************************************************/

void  inicifitxer()
/* Inicialitzem el fitxer de sortida de dades i l'estadistica
que hi sortira */
{
int  gene;
char    num[6];
```

```
strcpy(fitxerdad,NOMFITX); /* Afegim als noms dels fitxers les
terminacions */
strcpy(fitxercod,NOMFITX); /* adients i a mes el numero d'execucio
al fitxer */
sprintf(num,"%d",cont);     /* de dades. */
strcat(fitxerdad,num);
strcat(fitxerdad,".dad");
strcat(fitxercod,".cod");
dades=fopen(fitxerdad,"w");
fprintf(dades,"
/*****************************************************\n");
fprintf(dades,"          /                NATURA                    \n");
fprintf(dades,"           /             fitxer de resultats           \n");
fprintf(dades,"
/*****************************************************\n");
fprintf(dades,"\n");
fprintf(dades,"                PMUTATION= %2.4f    PCROSS=%2.4f
MAXPOP=%d    MAXITER=%d\n",pmut,
pcross,MAXPOP,MAXITER);
fprintf(dades,"\n");
fprintf(dades,"                    NUMCOD= %d       LONGCOD=%d
KPROB=%2.2f\n",NUMCOD,LONGCOD,kprob);
fprintf(dades,"\n");
fclose(dades);
for (gene=0;gene<MAXGEN;gene++) /* Inicialitzo l'estadistica */
   {
     estadist[gene].avg=0;
     estadist[gene].max=0;
     estadist[gene].min=0;
   }
}

/*************************************************/

void  reporta()
/* Reportem per pantalla i/o fitxer les dades que hem trobat  */
{
int i,c;

estadist[gen].avg+=avg/MAXITER; /* Afeguim dades a l'estadistica
*/
estadist[gen].min+=min/MAXITER;
estadist[gen].max+=max/MAXITER;
if (PANTALLA)                  /* Visualitzacio per pantalla. */
  {
    printf("\n");
```

```
  printf("     GEN = %4d   MAX= %5.0f   MIN =%5.0f   AVG=%5.3f \n"
      ,gen,max,min,avg);
  }


if ((gen==(MAXGEN-1))&&(iter==(MAXITER-1)))
  {
    dades=fopen(fitxerdad,"a");
  for (i=0;i<MAXGEN;i++)
     {
        fprintf(dades,"\n");
         fprintf(dades,"      GEN = %4d    MAX= %9.3f    MIN =%9.3f
AVG=%9.3f  \n"
        ,i,estadist[i].max,estadist[i].min,estadist[i].avg);
     }
    fclose(dades);
  }

calc_objfunc(oldpop[millor].chrom);

if (cost > millorcost ) /* sempre que troba un codi millor el guarda
*/
  {
    millorcost=cost;
    codi=fopen(fitxercod,"w");
    fprintf(codi,"
/*****************************************************\n");
   fprintf(codi,"      /                NATURA                    \n");
   fprintf(codi,"      /             Codi de major cost.           \n");
    fprintf(codi,"
/*****************************************************\n");
     fprintf(codi,"\n");
      fprintf(codi,"          GENERACIO= %4d          COST=%9.3f
RADI=%d\n",gen,cost,RADI);
     fprintf(codi,"\n");
           fprintf(codi,"                          ITERACIO=    %2d
KPROB=%2.2f\n\n",iter,kprob);
      fprintf(codi,"          PMUTATION= %2.4f     PCROSS=%2.4f
MAXPOP=%d\n",pmut,
    pcross,MAXPOP);
     fprintf(codi,"\n");
   for (i=0;i<NUMCOD;i++)
      {
        fprintf(codi,"\n");
       fprintf(codi,"                    %2d. ",i+1);
     for (c=0;c<LONGCOD;c++)
```

```
            fprintf(codi,"%d",oldpop[millor].chrom[i][c]);
        }
 if (RESULTAT)          /*    condicio d'acabament del programa.*/
   {
    if (cost==LONGESPAI)
      {
      gen =MAXGEN;
       iter=MAXITER;
       kprob=KFIN;
       if (PANTALLA)
            printf("\n\n\n\n\n\n\n\n                    Eureka !!!!!!!!.\n\n\n\n\n\n");
      }
   }
    fclose(codi);
   }
}

/*****************************************************/

void main()

---------- Com en Nou.c ----------
```

A.2.3 coverM.c

```
----------  Capçalera com en Cover.c ----------

int flip()

---------- Com en Nou.c ----------

int select(pop)

---------- Com en Nou.c ----------

void crossover(father,mother,son,daughter)
CROMOSOMA father,mother,son,daughter;

/* Creuem dos individus amb probabilitat pcross */

{
int i,c,lloc;

prob=pcross;
if ( flip() )
```

```
  {
   lloc=RANDOM(NUMCOD);                    /* Aquesta es la paraula per on
tallem  */
  for (i=0; i<LONGCOD; i++)
     {
     for (c=0; c<lloc;c++)
        {
         son[c][i]=father[c][i];
          daughter[c][i]=mother[c][i];
        }
     for (c=lloc; c<NUMCOD;c++)
        {
         son[c][i]=mother[c][i];
          daughter[c][i]=father[c][i];
        }
     }
   if (NUM_M>0)                          /* Creuem les paraules de la matriu */
     {
    lloc=RANDOM(LONGCOD);
    for (c=NUMCOD; c<(NUMCOD+NUM_M); c++)
        {
        for (i=0; i<lloc;i++)
           {
            son[c][i]=father[c][i];
             daughter[c][i]=mother[c][i];
           }
       for (i=lloc; i<LONGCOD;i++)
           {
            son[c][i]=mother[c][i];
             daughter[c][i]=father[c][i];
           }
       }
     }
    else
     for (i=0;i<LONGCOD;i++)
        for (c=0;c<(NUMCOD+NUM_M);c++)
           {
             son[c][i]=father[c][i];
              daughter[c][i]=mother[c][i];
           }
}

/************************************************/
```

```
void  mutation(chromos,mutchromos)
CROMOSOMA chromos,mutchromos;

/* Cambiem un bit del codi amb probabilitat pmut */

{
int  i,c;

for (c=0;c<(NUMCOD+NUM_M);c++)
   for (i=0; i<LONGCOD;i++)
         mutchromos[c][i]=chromos[c][i];

prob=pmut;
if ( flip() )
   {
  c=RANDOM(NUMCOD+NUM_M);
  i=RANDOM(LONGCOD);
    mutchromos[c][i]= (chromos[c][i]+1+RANDOM(BASE-1))%BASE;
      }
}

/*************************************************/
void  calc_objfunc(crom)
CROMOSOMA crom;

/* Calcul de la funcio de cost. En aquest cas es el nombre de
paraules que cubreix el codi */

{
int  n,e,c,m;

cost=0;
uno++;                  /* "uno" es el valor de la marca de paraula coberta
*/
if  (uno>32000)
   {
   for (e=0;e<LONGESPAI;e++) esp[e]=0;
    uno=1;
   }
if (NUM_M>0)  /*     Copiem el valor de la matriu particular.  */
   {
   for (m=0;m<NUM_M;m++)
     for (c=0;c<LONGCOD;c++)
        mat[LONGCOD+m][c]=crom[NUMCOD+m][c];
   }
```

```
for (n=0;n<NUMCOD;n++)
   {
      switch (RADI)       /* Anem cobrim l'espai amb cada distancia  */
         {
            case 3 : cubreix3(crom[n]);
            case 2 : cubreix2(crom[n]);
            case 1 : cubreix1(crom[n]);
            case 0 : cubreix0(crom[n]);
         }
   }


}
/*************************************************/

cubreix0(paraula)

---------- Com en Cover.c ----------

cubreix1(paraula)

---------- Com en Cover.c ----------

cubreix2(paraula)

---------- Com en Cover.c ----------

cubreix3(paraula)

---------- Com en Cover.c ----------

int  val_dec(para1)

---------- Com en Cover.c ----------

void  generation()

---------- Com en Cover.c ----------

void  statistic(pop)

---------- Com en Cover.c ----------

void  inicializa(oldpop)

---------- Com en Cover.c ----------
```

```
void inicifitxer()

---------- Com en Cover.c ----------

void reporta()

/* Reportem per pantalla i/o fitxer les dades que hem trobat */

{
int i,c;

estadist[gen].avg+=avg/MAXITER;          /* Afeguim dades a
l'estadistica */
estadist[gen].min+=min/MAXITER;
estadist[gen].max+=max/MAXITER;
if (PANTALLA)      /* Visualitzacio per pantalla.*/
  {
   printf("\n");
  printf("   GEN = %4d  MAX= %5.0f  MIN =%5.0f  AVG=%5.3f \n"
      ,gen,max,min,avg);
  }


if ((gen==(MAXGEN-1))&&(iter==(MAXITER-1)))
  {
    dades=fopen(fitxerdad,"a");
  for (i=0;i<MAXGEN;i++)
     {
        fprintf(dades,"\n");
         fprintf(dades,"      GEN = %4d   MAX= %9.3f   MIN =%9.3f
AVG=%9.3f  \n"
        ,i,estadist[i].max,estadist[i].min,estadist[i].avg);
     }
   fclose(dades);
  }
calc_objfunc(oldpop[millor].chrom);
if (cost > millorcost )        /* sempre que troba un codi millor el
guarda */
  {
   millorcost=cost;
   codi=fopen(fitxercod,"w");
   fprintf(codi,"
/*************************************************\n");
  fprintf(codi,"      /                NATURA                    /\n");
  fprintf(codi,"      /             Codi de major cost.          /\n");
```

```
   fprintf(codi,"
/**************************************************\n");
    fprintf(codi,"\n");
     fprintf(codi,"          GENERACIO= %4d          COST=%9.3f
RADI=%d\n",gen,cost,RADI);
    fprintf(codi,"\n");
          fprintf(codi,"                    ITERACIO=   %2d
KPROB=%2.2f\n\n",iter,kprob);
     fprintf(codi,"          PMUTATION= %2.4f     PCROSS=%2.4f
MAXPOP=%d\n",pmut,
   pcross,MAXPOP);
    fprintf(codi,"\n\n");
  if (NUM_M>0)
    {
     fprintf(codi,"       Matriu M  emprada : \n");
   for (i=NUMCOD;i<(NUMCOD+NUM_M);i++)
      {
        fprintf(codi,"\n");
       fprintf(codi,"                  %2d. ",(i+1-NUMCOD));
     for (c=0;c<LONGCOD;c++)
           fprintf(codi,"%d",oldpop[millor].chrom[i][c]);
      }
    }
   fprintf(codi,"\n\n       Codi trobat : \n");
  for (i=0;i<NUMCOD;i++)
    {
      fprintf(codi,"\n");
     fprintf(codi,"                  %2d. ",i+1);
   for (c=0;c<LONGCOD;c++)
         fprintf(codi,"%d",oldpop[millor].chrom[i][c]);
    }

  if (RESULTAT)          /*   condicio d'acabament del programa.*/
    {
    if (cost==LONGESPAI)
      {
      gen =MAXGEN;
       iter=MAXITER;
       kprob=KFIN;
       if (PANTALLA)
            printf("\n\n\n\n\n\n\n\n          Eureka !!!!!!!!.\n\n\n\n\n");
      }
    }

    fclose(codi);
  }
```

```
}

/*************************************************/

void main()

------------- Com en Cover.c -------------
```

A.2.4 coverf4.c

```
------------- Capçalera com en Cover.c -------------

int flip()

------------- Com en Cover.c -------------
int select(pop)

------------- Com en Cover.c -------------

void crossover(father,mother,son,daughter)

------------- Com en Cover.c -------------
void mutation(chromos,mutchromos)

CROMOSOMA chromos,mutchromos;

/* Cambiem un bit del codi amb probabilitat pmut */

{
int i,c,lloc;

for (c=0;c<NUMCOD;c++)
   for (i=0; i<LONGCOD;i++)
         mutchromos[c][i]=chromos[c][i];

prob=pmut;
if ( flip() )
   {
  c=RANDOM(NUMCOD);
  lloc=RANDOM(LONGCOD);
  if (lloc==(LONGCOD-1))
       mutchromos[c][lloc]=(chromos[c][lloc]+1+RANDOM(3))%4;
      else
           mutchromos[c][lloc]= (chromos[c][lloc]+1)%2;
```

```
    }
}

/*************************************************/
void calc_objfunc(crom)

-------------- Com en Cover.c --------------

cubreix0(paraula)

-------------- Com en Cover.c --------------

cubreix1(paraula)
CODI paraula;

/* Cobrim l'espai amb distancia u */

{
int c1,b1,para,c;
CODI par;

for (c1=0;c1<NUMCOL;c1++)
                    /* Trobem totes les paraules que estan a */
   {                          /* distancia u de "paraula"          */
  for (c=0;c<(LONGCOD-1);c++)
        par[c]=(mat[c1][c]+paraula[c])%2;
    for (b1=1;b1<4;b1++)
      {
      par[LONGCOD-1]=((mat[c1][LONGCOD-1]*b1)+paraula[LONGCOD-
1])%4;
        para=val_dec(par);
        if (esp[para]!=uno)
        {
          esp[para]=uno;
        cost++;
        }
      }
   }
}

/*************************************************/

cubreix2(paraula)
CODI paraula;

/* Cobrim l'espai amb distancia dos */
```

```
{
int  c1,c2,b1,b2,c,para;
CODI par;

for (c1=0;c1<(NUMCOL-1);c1++)
                        /* Trobem totes les paraules que estan a   */
    for (c2=c1+1;c2<NUMCOL;c2++) /* distancia dos de "paraula" */
      {
     for (c=0;c<(LONGCOD-1);c++)
           par[c]=(mat[c1][c]+mat[c2][c]+paraula[c])%2;
       for (b1=1;b1<4;b1++)
       for (b2=1;b2<4;b2++)
          {
          par[LONGCOD-1]=((mat[c1][LONGCOD-
1]*b1)+(mat[c2][LONGCOD-1]*b2)+
           paraula[LONGCOD-1])%4;
             para=val_dec(par);
            if (esp[para]!=uno)
            {
              esp[para]=uno;
            cost++;
            }
          }
      }
}

/*****************************************************/

cubreix3(paraula)
CODI paraula;

/* Cobrim l'espai amb distancia tres */

{
int  c1,c2,c3,b1,b2,b3,c,para;
CODI par;

for (c1=0;c1<(NUMCOL-2);c1++)
                        /* Trobem totes les paraules que estan a */
    for (c2=c1+1;c2<(NUMCOL-1);c2++)
                        /* distancia tres de "paraula"*/
      for (c3=c2+1;c3<NUMCOL;c3++)
          {
          for (c=0;c<(LONGCOD-1);c++)
                par[c]=(mat[c1][c]+mat[c2][c]+mat[c3][c]+paraula[c])%2;
```

```
            for (b1=1;b1<4;b1++)
            for (b2=1;b2<4;b2++)
            for (b3=1;b3<4;b3++)
              {
              par[LONGCOD-1]=((mat[c1][LONGCOD-1]*b1)+
              (mat[c2][LONGCOD-1]*b2)+(mat[c3][LONGCOD-1]*b3)+
              paraula[LONGCOD-1])%4;
                 para=val_dec(par);
                if (esp[para]!=uno)
                {
                  esp[para]=uno;
                cost++;
                }
              }
          }

}

int  val_dec(para1)

---------- Com en Cover.c ----------

void  generation()

---------- Com en Cover.c ----------

void  statistic(pop)

---------- Com en Cover.c ----------

void  inicializa(oldpop)
POBLACIO oldpop;

/* Inicialitzem la poblacio aleatoriament i calculo la seva
funcio de cost. Tambe inicialitzo la matriu de covertura. */

{
int j,c,p;
char    num[6];

uno=32001;
for (c=0;c<NUMCOL;c++)
   {
   for (j=0;j<LONGCOD;j++) mat[c][j]=0;
    mat[c][c]=1;
   }
```

```
for (p=0; p<MAXPOP; p++)
   {
  for (j=0; j<NUMCOD; j++)
      {
      for (c=0; c<(LONGCOD-1); c++)
           oldpop[p].chrom[j][c]=RANDOM(2);
       oldpop[p].chrom[j][LONGCOD-1]=RANDOM(4);
       }
      calc_objfunc(oldpop[p].chrom);
      oldpop[p].fitness=cost;
    }
}

/*************************************************/

void inicifitxer()

---------- Com en Cover.c ----------

void reporta()

---------- Com en Cover.c ----------

void main()

---------- Com en Cover.c ----------
```

Bibliografia

[1] Abbas A. El Gamal, senior member, IEEE, Lane A. Hemachandra, Itzhak Shperling, and Victor K. Wei, member, IEEE, "Using Simulated Annealing to design good codes.", IEEE Transactions of Information theory, Vol. 33, No. 1, January, 1987

[2] A. Blokhuis and C.W.H. Lam, "More covering by rook domains", J. Combin. Theory Ser. A 36 (1984) 240-244.

[3] A. Bruce Carlson, "Communications Systems", Mc Graw-Hill Company.Lawrence Davis, "Handbook of genetic Algorithms", Van Nostrand Reinhold (New York) 1991.

[4] A. E. Brouwer, James B. Shearer, N.J.A. Sloane, fellow, IEEE, and Warren D. Smith, "A new table of constant weight codes.", IEEE Transactions of information theory, vol36, No. 6, November 1990.

[5] Bellman, R., "Adaptative control processes: A guided tour", Princeton, NJ: Princeton University Press,1961.

[6] Bagley, J. D. "The behavior of adaptative systems which employs genetic and correlation algoritms", (Doctoral dissertation, University of Michigan) Dissertation Abstracts International 28(12),5106B. 1967.

[7] Baker, J. E. "Adaptative selection metods for genetic algorithms", Proceedings of an International Conference on Genetic Algorithms and Their Applications, 101-111, 1985.

[8] Bethke, A. D. " Genetic Algorithms as function optimizers", (Doctoral dissertation, University of Michigan) Dissertation Abstracts International 41(9), 3503B, 1981.

[9] Brindle, A. "Genetic Algorims for function optimitation" Unpublished doctoral dissertation, University of Alberta, Edmonton, 1981.

[10] Cavicchio, D.J. "Adaptative search using simulated evolution". Unpublished doctoral dissertation, University of Michigan, Ann Arbor, 1970.

[11] Davis, L (Ed), "Genetic algorithms and simulated annealing", London: Pitman, 1987.

[12] Davis, L. "Applying adaptative algorithms to epistatic domains", Proceedings of the 9th International Joint Conference on Artificial Intelligence, 162-164, 1985.

[13] Davis, L. "Job shop scheduling with genetic algorithms" Proceedings of an International Conference on Genetic Algorithms and their Applications, 136-140, 1985.

[14] Davis, L. & Smith, D. (1985). "Adaptative design for layout synthesis", Texas Instruments international report), Dallas: Texas Instruments, 1985.

[15] De Jong, K.A. "An analisys of the behavior of a class or genetic adaptative systems", (Doctoral dissertation, University of Michigan). Dissertation Abstracts International 36(10), 5140B, 1975.

[12] Frantz, D. R. "Nonlinearities in genetic adaptative search", Doctoral dissertation, University of Michigan). Dissertation Abstract International,33(11), 5240B-5241B, 1972.

[13] Forrest, S. "Documentation for PRISONERS DILEMMA and NORMS programs that use the genetic algorithm.", unpublished manuscript, University of Michigan, Ann Arbor, 1985.

[14] Gerhard J. M. Van Wee, "Improved sphere bounds on the covering radius of codes.", IEEE Transaction on information theory, Vol. 34, No. 2, March 1988.

[15] Gillies, A. M."Machine learning procedures for generating image domain feature detectors", Unpublished doctoral dissertation, University of Michigan, Ann Arbor, 1985.

[16] Goldberg D. E., "Genetic Algorithms in Search, Optimization, and machine learning",.Addison Wesley, 1989.

[17] Goldberg, D.E., "Optimal initial population size for binary-coded genetic algorithms", (TCGA Report No. 85001). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms, 1985.

[18] Goldberg, D. E. & Lingle, R. "Alleles, loci, and the travelling salesman problem." Proceedings of an International Conference on Genetic Algorithms and Their Applications, 154-159, 1985.

[19] Golberg, D. E. & Richardson, J. "Genetic algorithms with sharing for multimodal function optimization". Genetic algorithms and their applications: Proceedings of the second International Conference on Genetic Algorithms, 41-49, 1987.

[20] Goldberg, D.E. & Smith, R.E. "Nonstationary function optimization using genetic algorithms with dominance and diploidy", Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms, 59-68, 1987.

[21] Grosso, P. B. "Computer simulation of genetic adaptation: Parallel Subcomponent interaction in a multilocus model." Doctoral dissertation, University of Michigan), 1985.

[22] Holland, J.H. "Hierarchical descriptions of universal spaces and adaptative systems", Technical report ORA projects 01252 and 08226, Ann Arbor: University of Michigan, Department of Computer and Communications Sciences, 1968.

[23] Holland, J.H. "Genetic Algorithms and the optimal allocation of trials", SIAM Journal of Computing, 2(2), 88-105, 1973.

[24] Holland, J.H. "Adaptation in natural and artificial systems", Ann Arbor: The university of Michigan Press, 1975.

[25] Hollstien, R. B. "Artificial genetic adaptation in computer control systems", (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 32(3),1510B, 1971.

[26] H.J.L. Kamps and J.H. van Lint, "A covering problem", in: Colloq. Math. Soc. János Bolyai; Hung. Combin. Theory and Appl (Balantonfüred, Hungary,1969) 679-685.

[27] J.H. van Lint Jr, "Covering radius problems", M. Sc. Thesis, Eindhoven University of Technology (1988).

[28] Lawrence Davis, "Handbook of genetic Algorithms", Van Nostrand Reinhold (New York) 1991.

[29] Oliver, I. M., Smith, D. J., & Holland, J. R. C. "A study of permutation crossover operators on the travelling salesman problem", Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms, 224-230, 1987.

[30] O. Taussky and J. Todd, "Covering theorems for groups.", Ann. Soc. Polon. Math., vol 21, pp.303-305, 1948.

[31] Patrick R. J. Östergård,"A new binary code of length 10 and covering radius 1.", IEEE Transactions of information theory, Vol. 37, No. 1, January 1991.

[32] Patrick R. J. Östergård, "Upper bound for q-ary covering codes.", IEEE Transactions of Information theory, Vol. 37, No. 3, May, 1991.

[33] Patrick R. J. Östergård, "Upper Bounds for the football Pool and other covering Problems.",

[34] Patrick R. J. Östergård, "Using Simulated Annealing in the design of covering codes."

[35] Perry, Z. A. "Experimental study of speciation in ecological niche theory using genetic algorithms", (Doctoral dissertation, University of Michigan). Dissetation Abstracts International, 45(12), 3870B, 1984.

[36] Rosenberg, R. S. "Simulation of Genetic populations with biochemicals properties" (Doctoral dissertation, University of Michigan), Dissertation Abstracts International 28(7), 2732B, 1967.

[37] Smith, D. "Bin packing with adaptative search", Proceedings of an International Conference on Genetic Algorithms and Their Applications, 202-206, 1985.

[38] Smith, R. E. "Diploid genetic algorithms for search in time varying environments", Proceedings of the 25th Annual Southeast Regional Conference of the ACM,175-178, 1987.

[39] Smith, R. E. "An investigation of diploid genetic algorithms for adaptative search of nonstationary functions", Unpublished master's thesis, University of Alabama, Tuscaloosa, 1988.