Índex

1 Introducció

1.1 Esdeveniments esportius multitudinaris

Els esdeveniments esportius són actualment fenòmens de masses als que hi acudeixen grans quantitats de persones. Aquesta enorme afluència d'espectadors requereix una bona organització i control per tal que no sorgeixin problemes en el transcurs del mateix.

En alguns d'aquests esdeveniments esportius, com per exemple les olimpíades, els grans premis de Fórmula 1, els mundials o les eurocopes de futbol, els espectadors que volen assistir-hi no compren un seient físic de l'estadi o recinte sinó que compren el dret a ocupar una zona amb unes característiques determinades. Per aquest fet, una de les responsabilitats dels comitès organitzadors d'aquests esdeveniments consisteix en la distribució de les localitats del recinte per a les persones que han adquirit aquest dret.

La distribució de les persones es converteix en un procés complex quan es parla de milers o desenes de milers de tiquets a repartir en múltiples blocs i zones. Una altra dificultat apareix degut al fet que sovint, les reserves de seients no les fa una única persona per reservar un sol seient. És més habitual que la reserva sigui d'un conjunt de seients per a un grup de persones. Fins i tot en els casos esmentats anteriorment bona part de les localitats van destinades a patrocinadors que aprofiten aquests esdeveniments per fer-hi publicitat i que per contracte els organitzadors es comprometen a proporcionar-los certes quantitats d'entrades a repartir pels membres de l'empresa. També hi ha altres col·lectius que ocupen algunes localitats com són per exemple els membres de seguretat, els primers auxilis, membres del comitè organitzador, invitacions especials, etc.

1.2 Fórmula 1

La Fórmula 1 és un d'aquests esdeveniments esportius on s'ha de realitzar la distribució dels espectadors ja que el públic que hi assisteix, quan ha adquirit les entrades no portaven associades un seient físic del circuit. Aquesta associació s'ha de fer en un procés independent, el qual es realitza un cop es té tota o part de la informació de les entrades sense seient assignat venudes i la seva comesa és efectuar adequadament aquesta vinculació.

Aquest és un dels esdeveniments esportius amb més afluència d'espectadors que existeix actualment. En alguns grans premis es pot arribar a superar el centenar de miler de persones. Les persones poden ocupar dos tipus de zones clarament diferenciades. Un tipus de zones són les formades per grades amb seients i l'altre és l'anomenada zona 'pelouse'. En les zones 'pelouse' no hi ha seients; generalment són espais amb gespa, on la gent va ocupant l'espai disponible a mida que va arribant.

A més a més del públic general hi ha un conjunt d'entrades destinades als patrocinadors de l'esdeveniment. Els espònsors de la Fórmula 1 tenen dret a adquirir un número d'entrades de forma gratuïta. A part de la publicitat, aquestes empreses aprofiten els grans premis per realitzar contactes i accions de màrqueting amb els seus clients.

També hi ha altres col·lectius que ocupen algunes localitats com per exemple els membres de seguretat, els de primers auxilis, membres de la FIA, proveïdors oficials, socis de màrqueting, etc.

En el següent gràfic es pot veure una relació dels diferents grups que constitueixen entitats independents en el procés del tiquetatge.

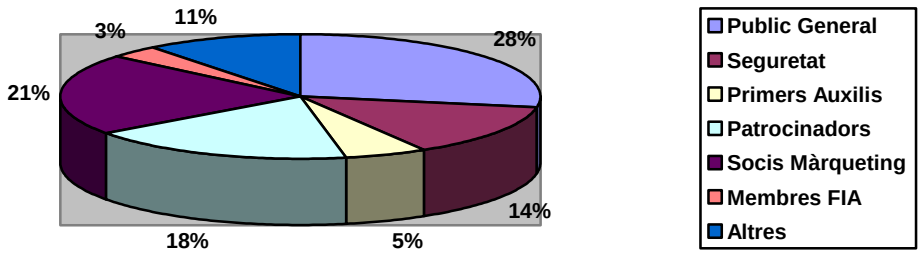


Figura 1.1 Gràfic d'entitats independents de tiquetatge

Per a realitzar la distribució de seients la FIA (Federació Internacional de l'Automòbil) disposa d'un protocol on s'especifiquen tot un seguit de regles i restriccions que s'hauran de complir. Aquest protocol fa que la distribució d'espectadors sigui un procés molt complexe de realitzar manualment.

1.3 Motivació

Fins ara la distribució dels espectadors als seients físics del circuit s'ha fet manualment. En aquesta feina hi havien d'intervenir un bon nombre de persones dedicant-hi una gran quantitat de temps. La tasca no és només llarga i repetitiva sinó també complexa degut al gran nombre de restriccions a tenir en compte segons imposen les normes de la FIA, obligant molts cops a desfer part o inclús tot el procés de distribució i tornar a començar per poder finalment satisfer totes les regles. A més a més al final de la distribució mai s'està prou segur de que es compleixin totes les regles.

Per aquesta raó, el fet de disposar d'un sistema automàtic que dugui a terme aquesta tasca efectivament i complint amb totes les especificacions prefixades resultaria en una gran ajuda pels responsables de l'organització, constituint un estalvi considerable de recursos humans i de temps.

1.4 Objectius

L'objectiu d'aquest projecte és l'elaboració d'un mètode o algorisme que realitzi el procés de la distribució d'espectadors de forma efectiva i automàtica.

L'algorisme de distribució d'espectadors es pot entendre com el conjunt de tasques destinades a assignar de forma efectiva i correcta segons els criteris i normes de la FIA els seients d'un circuit de Fórmula 1 als tiquets o conjunts de tiquets venuts per a un gran premi concret del calendari de la FIA.

Aquest algorisme tindrà un conjunt de dades d'entrada:

- Informació del circuit (blocs, seients, categories, etc)
- La llista de tiquets venuts, anomenats ordres (conjunt de tiquets)
- Les regles del protocol de la FIA que regulen l'assignació d'ordres a seients

La sortida ha de ser un conjunt d'assignacions dels tiquets als seients que compleixi les regles determinades a l'entrada.

L'algorisme ha de ser flexible en el sentit de que ha de poder proporcionar distribucions diferents en funció d'una sèrie de paràmetres que es defineixen per cada regla del protocol.

Hi pot haver casos en què moltes distribucions diferents compleixin les restriccions; l'objectiu ideal de l'algorisme serà trobar la millor d'aquestes distribucions, segons uns criteris de valoració que s'hauran de definir. Per altra banda alguns conjunts de dades d'entrada poden resultar impossibles d'assignar íntegrament satisfent les regles, per aquesta raó, en aquests casos l'algorisme haurà de ser capaç de trobar la millor solució possible del problema.

Degut a la complexitat del problema trobar la millor assignació resultarà inviable en termes de temps d'execució, per tant l'objectiu real de l'algorisme serà trobar la una solució propera a la òptima que serà millor en funció del temps que es disposi per a l'execució.

1.5 Àmbit i abast

L'àmbit d'aquest projecte es troba en la intel·ligència artificial. La intel·ligència artificial disposa de múltiples tècniques i mètodes per tal de resoldre aquests tipus de problemes el més eficientment possible.

Per realitzar la distribució d'espectadors es representarà el problema com un Problema de Satisfacció de Restriccions (CSP – *Constraint Satisfaction Problem*) i es resoldrà utilitzant tècniques generals de satisfacció de restriccions, mitjançant la utilització de funcions heurístiques i mecanismes de poda (*branch & bound*) per explorar amb criteri el desmesurat espai de cerca. També s'utilitzaran algorismes basats en *region growing* durant el procés d'exploració i mecanismes de *backtracking* alhora de millorar els resultats obtinguts inicialment.

Aquest projecte s'enfocarà a resoldre el problema de la distribució d'espectadors per als circuits que formen part del calendari de la Formula 1 a l'any 2003, tot i que la solució serà aplicable a qualsevol altre circuit, o més àmpliament a qualsevol recinte o estadi que estigui

format per un conjunt de seients distribuïts per blocs on s'hi hagin de distribuir els espectadors seguint aquestes mateixes (o menys) regles.

1.6 Estructura d'aquest document

Aquest document es composa a més de la introducció de 4 capítols: Requeriments, Disseny, Implementació, Resultats i Conclusions. En el primer capítol es defineixen quins són els requeriments del projecte: s'especificaran les dades d'entrada, les tasques a realitzar i les dades de sortida. En el segon es descriurà el disseny de l'aplicació; primer es farà un l'anàlisi dels diferents mètodes per solucionar el problema i seguidament es justificarà i detallarà l'algorisme proposat. En aquest disseny s'introdueix el concepte de funció de fitness per mesurar el grau de bondat d'una solució, la qual es definirà en aquest mateix capítol. El capítol d'implementació mostrarà com s'ha implementat el disseny decidit i al capítol de resultats es comentaran i s'analitzaran els resultats obtinguts. Finalment s'extrauran les conclusions d'aquest projecte i es proposaran algunes millores i treballs futurs.

2 Requeriments

2.1 Introducció

El propòsit d'aquest capítol és especificar els requeriments funcionals de l'algorisme de distribució d'espectadors. És a dir, determinar quines han de ser les dades amb les que ha de treballar el procés, com s'han de tractar aquestes dades i quin tipus de sortida s'ha de proporcionar.

Per a un circuit de Fórmula 1, es venen dos tipus bàsics d'entrades. Unes són les que van destinades a les grades que hi ha repartides per tot el recorregut de la pista. Les grades es composen de blocs on cada bloc és un conjunt de seients disposats matricialment en files i columnes. L'altre tipus d'entrades són per les zones 'pelouse' del circuit. En aquestes zones no hi ha seients; generalment són espais amb gespa, on la gent va ocupant l'espai disponible a mida que va arribant.

A la zona de pelouse no s'hi ha de fer cap tipus de distribució, són els organitzadors de cada circuit els que hauran de controlar que el número d'entrades venudes per les diferents zones pelouse del circuit no sobrepassin la seva capacitat. Per tant el projecte se centrarà en la distribució dels espectadors a les zones amb seients materials.

2.2 Dades d'entrada

En aquesta secció es detallen totes les dades que necessitarà l'algorisme per tal de realitzar la distribució dels espectadors a les grades del circuit. Aquestes dades es poden classificar com a dades del circuit, dels clients i dels tiquets.

2.2.1 El circuit

Per a cada cursa l'algorisme necessitarà les dades de com estan distribuïts els seients del circuit i quines característiques tenen.

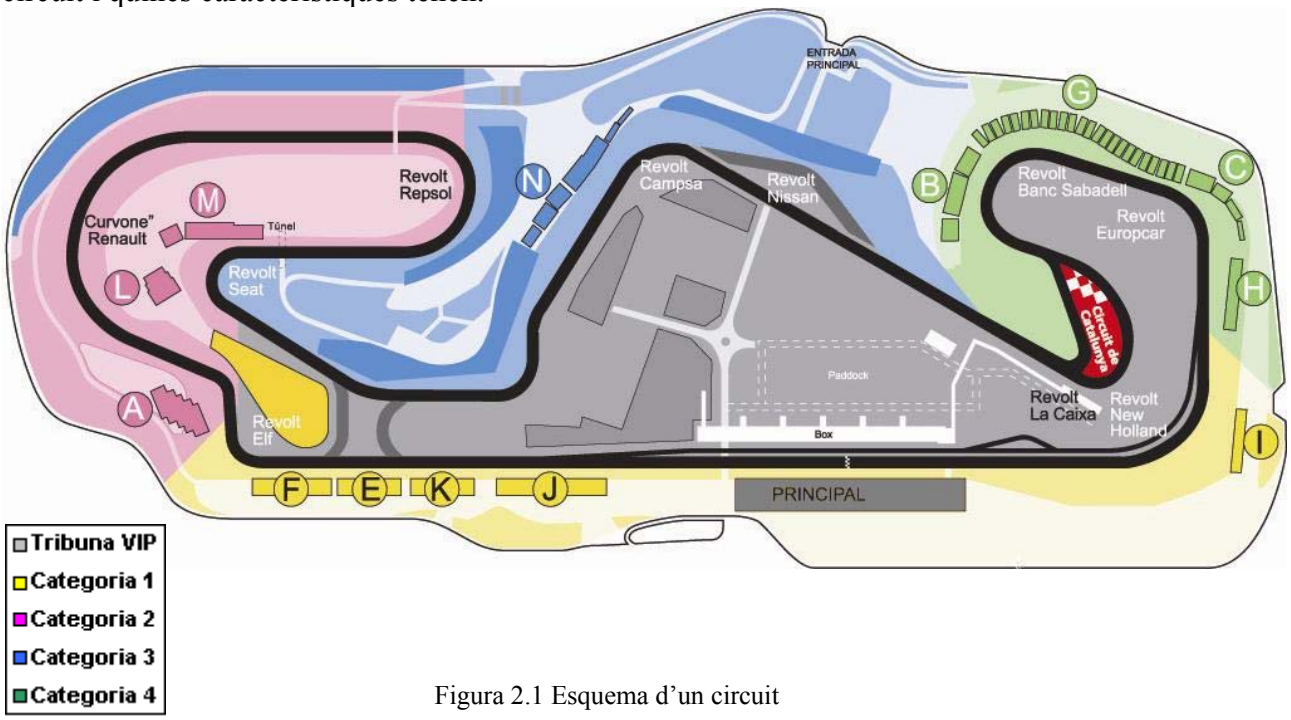


Figura 2.1 Esquema d'un circuit

En la figura 2.1 observem on es troben les grades en el circuit. Les grades d'un circuit (marcades amb una lletra) estan composades per un o més blocs. Els blocs són conjunts de seients contigus disposats en files i columnes. Cada bloc pertany a una categoria (VIP, Categoria 1, 2, 3, ...). Aquesta categoria representa la classe del bloc. La categoria VIP estarà destinada a personalitats importants, i les categories 1, 2, 3, ... estaran destinades a la resta de públic tenint en compte que la categoria 1 serà la més alta i per tant de millor qualitat. Cada bloc del circuit tindrà un rànquing que classificarà els blocs del circuit de millor a pitjor seguint una sèrie de criteris preestablerts pel comitè organitzador.

Exemple de taula de blocs d'un circuit:

Bloc	Categoria	Nombre de Seients	Rànquing
1	VIP	200	1
2	VIP	100	2
3	1	500	3
4	1	500	4
5	2	20	10
6	3	80	5
100	4	400	120

Taula 2.2 Exemple de taula de blocs d'un circuit

Cada bloc està format per files on cada fila té una sèrie de seients. Els blocs poden tenir els seients distribuïts de moltes formes diferents. Un bloc es representa amb una matriu de *n* files i *m* columnes on cada casella representa un seient. La matriu pot no tenir totes les caselles disponibles, per exemple quan alguns dels seients no existeixen degut a l'estructura del bloc (en els casos que el bloc no sigui quadrat o amb diferents nombres de seients per cada fila).

Exemple de bloc:

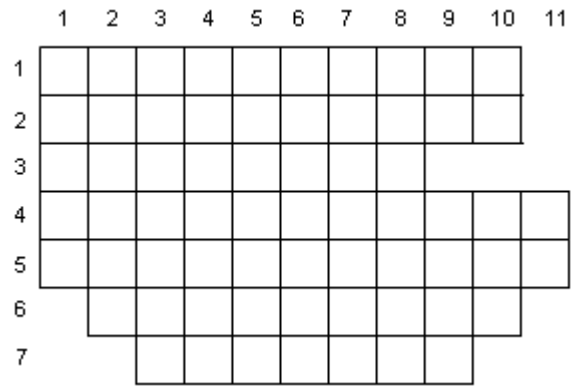


Figura 2.3 Exemple de la matriu d'un bloc

Cada seient del bloc té un conjunt d'atributs:

- Identificadors del seient: identifica el seient de forma única en tot el circuit. L'identificador està format per diversos atributs:

Bloc	Fila	Columna	X	Y

  Els camps Bloc, Fila i Columna identifiquen cada seient concret dins del circuit segons la numeració que té el circuit en qüestió. Els altres dos camps X, Y representen la posició del seient en la matriu anteriorment descrita. Aquests atributs serviran per tenir coneixement sobre quins seients són contigus.

- Categoria: Es contemplaran un seguit de categories de seient diferents depenent de la seva situació/qualitat: VIP, Categoria 1, Categoria 2, Categoria 3, ...

- Sector: Dins del circuit, independentment de la categoria, es poden distingir diversos sectors que tindran unes característiques una mica diferents. Per exemple, trobarem el sector 'Confortable' on els seients són encoixinats, el sector 'Hospitality' que té dret a càtering o el sector majoritari anomenat 'Standard' on hi ha els seients normals.

- Type: Els seients estan classificats en 'Complimentary' i 'Purchasable'. Els 'Complimentary' són uns seients específics del circuit que l'organització regala i els 'Purchasable' són tota la resta de seients que hi ha a la venda.

- Status: En un circuit hi poden haver seients amb visibilitat parcial o nul·la de la pista, o que estan ocupats per la seguretat o les televisions. Quan un seient estigui en aquesta situació aquest atribut ens indicarà que el seient està reservat o no està disponible i això voldrà dir que no es podrà assignar aquest seient a cap tiquet normal. En el cas de visibilitat parcial ens podem trobar que hi hagi una sèrie de tiquets venuts per seients amb aquest status.

- Price Type: els seients tenen un tipus de preu, que va des del preu regular per defecte, fins a preus amb un percentatge de descompte. Cada tiquet indicarà a quin tipus de preu s'ha d'assignar.

- Rank: Cada seient té un rànquing representat per un número. Aquest rànquing representa la qualitat del seient que vindrà determinada pel posicionament del seient dins del bloc i de la seva visibilitat del la pista. Aquesta numeració es farà seguint una metodologia que no entra dins de l'abast d'aquest projecte. El número més petit correspon al rànquing més alt. Varis seients del mateix bloc poden tenir el mateix rànquing.

- Assignation: L'algorisme pot rebre d'entrada blocs amb algunes localitats ocupades (ja sigui perquè s'han assignat manualment o perquè s'utilitzen dades d'entrada que són realment assignacions parcials fetes per aquest mateix o un altre algorisme). Aquest fet s'indicarà en aquest atribut. En el cas que el seient estigui ocupat, en aquest camp hi apareixerà l'identificador del client que l'ocupa, en cas contrari romandrà buit.

Exemple de llista de seients:

Bloc	Fila	Col	X	Y	Categ	Sector	Type	Status	Price Type	Rank	Assign.
1	1	1	1	1	VIP		Purchasable	Std	Regular	1	
1	1	2	1	2	VIP		Purchasable	Std	Regular	2	
1	10	10	10	10	VIP		Purchasable	Std	Regular	10	
10	1	1	1	3	1		Purchasable	Obstr	Regular	20	
5	1	1	1	1	1	Confort	Purchasable	Std	Regular	40	
6	5	5	5	5	1	SkyBox	Purchasable	Std	Regular	100	
6	6	1	6	1	1		Complimentary	Std	Regular	101	
6	6	2	6	2	1		Purchasable	Killed	Regular	102	
11	7	7	7	10	1		Purchasable	Std	10% Discount	103	RedBull
30	12	1	13	4	2		Purchasable	Std	Regular	500	RedBull

Taula 2.4 Exemple de taula de seients

2.2.2 Els clients

Els clients són els col·lectius d'individus que posseeixen entrades pels circuits. Aquests clients tindran un conjunt de tiquets, anomenats requests, a la seva disposició en una o diverses categories. Aquests requests es separaran en conjunts més petits anomenats ordres d'unes mides concretes tal com s'explica en l'apartat 2.2.3.

L'univers de clients està estructurat en varis nivells. El nivell superior és el de segments de mercat, el qual classificarà als clients en funció del segment de mercat en el que se trobin. En aquest primer nivell podrem trobar als espònsors, televisions, públic general, etc. El segon nivell (Grups de Clients) dividirà als clients del primer nivell segons el nom de l'empresa o organització a la que pertanyen. I al tercer nivell (Grups de venda) trobarem les diferents delegacions de l'empresa amb dret a posseïr tiquets.

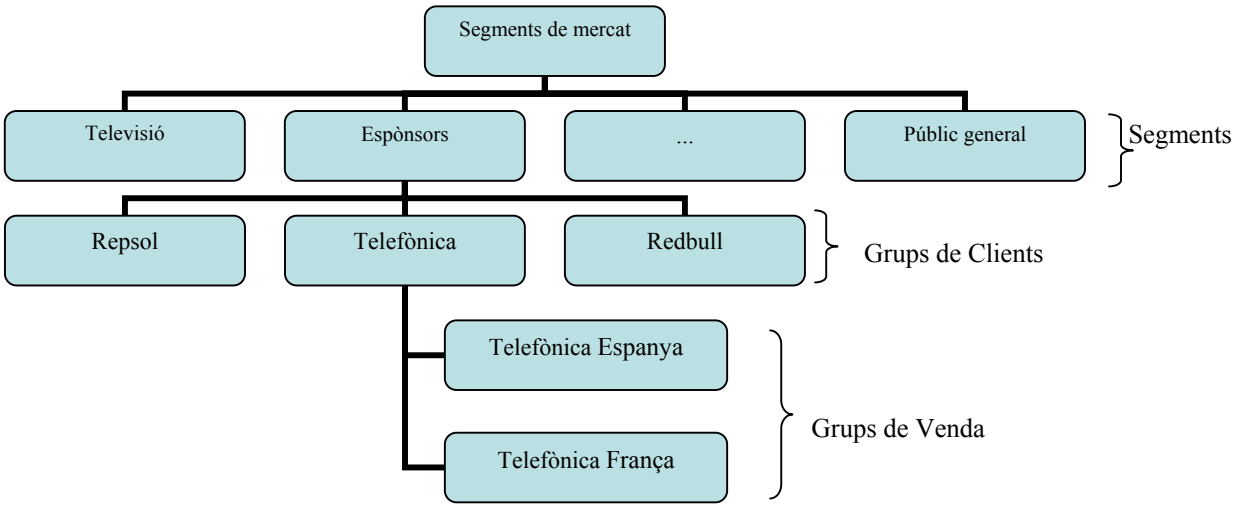


Figura 2.5 Diagrama de nivells de clients

Exemple de taula de clients:

Client Id	Nom client	Segment de Mercat
1	Telefònica	Espònsor
2	Repsol	Espònsor
3	Redbull	Espònsor
4	Apple	Espònsor
5	Toshiba	Espònsor
6	Movistar	Espònsor
10	TV3	Televisió
11	Televisió de Girona	Televisió
100	Josep Puig	Públic General

Taula 2.6 Exemple de taula de clients

2.2.3 Els tiquets

Els diferents grups de venda enviaran les comandes (Requests) amb les quantitats de tiquets que desitgin. Per exemple, podrem dir que el grup de venda Telefònica Espanya ha realitzat una Request de 500 tiquets.

Les Request es partiran en subgrups de tiquets que es diferenciaran pel sector a on han d'anar assignats. A aquestes particions les anomenarem Grups d'ordres de tickets (TOG - Ticket Order Groups). Per exemple, podrem dir que la Request de Telefònica Espanya s'ha dividit en 3 TOG: 50 tiquets de categoria 1 en el sector 'Tribuna VIP', 350 tiquets de categoria 1 en el sector 'Confortable' i 100 tiquets de Categoria 1 en el sector 'Standard'. La realització d'aquesta partició no entra dins l'abast del projecte i, per tant, l'algorisme rebrà com a entrada el conjunt de TOG.

Cada TOG tindrà un conjunt d'atributs que definiran en quin tipus de seients ha de ser assignat:

Request	TOG	Client Id	Number of Tickets	Category	Type	Price Type	Dispersion	Rank

El 'Request' identifica l'ordre d'un client, i el 'TOG' identifica dins d'una Request els diferents subgrups de tiquets amb unes característiques concretes de confortabilitat. Per tant, per cada Request tindrem 1 o més TOGs. El 'Client Id' identificarà el client que ha realitzat l'ordre i el 'Number of Tickets' marcarà quants tiquets té el TOG. La 'Category' marcarà a quina categoria de seients ha d'anar assignat aquest TOG. El 'Type' indica el tipus de tiquet, que pot ser Complimentary (regalat) o Purchasable (comprat). El 'Price Type' indicarà si és un tiquet comprat normal (Regular) o si se li ha aplicat algun tipus de descompte pel motiu que sigui (per exemple, per visibilitat limitada de la pista).

L'atribut 'Dispersion' és un flag que podrà estar activat o no. Quan estigui activat significarà que s'hauran de dividir els tiquets del TOG corresponent en subgrups d'unes mesures concretes i que aquests subgrups en conjunt no podran assignar-se junts sinó que s'hauran de dispersar pel circuit en diferents blocs o si són al mateix bloc de manera que estiguin separats amb una distància mínima.

Finalment, el 'Rank' ens marcarà la prioritat d'assignació de les TOG als seients, essent un rank petit una alta prioritat. Els ranks no són únics, per tant, podrem tenir diferents TOG amb el mateix rank, indicant que són igual de prioritaris.

A continuació es detalla com seria un exemple d'una llista de TOG:

Request	TOG	Client Id	Number of Tickets	Category	Type	Price Type	Dispersion	Rank
1	1	1	50	1	Purchasable	Regular	False	1
1	2	1	250	1	Purchasable	Regular	True	6
2	1	2	10	1	Complimentary	Regular	False	1
2	2	2	30	1	Purchasable	Regular	False	6
2	3	2	150	1	Purchasable	Regular	True	11
3	1	8	40	1	Purchasable	Regular	False	10
3	2	8	150	1	Purchasable	10%	True	135

Taula 2.7 Exemple de taula de tiquets

2.2.3.1 Sector

Podem observar que molts dels atributs dels tiquets es corresponen amb els dels seients. Però n'hi ha un que no apareix als TOG, aquest és el sector. Els TOG no porten directament escrit el sector on s'han de seure. Aquesta dada s'haurà d'extreure d'unes matrius de compatibilitat que relacionaran als TOG amb els sectors en funció del seu Rank. La matriu de compatibilitat té aquesta forma:

SECTOR / **TOG Rank**	Tribuna VIP	Confortable	Category 1	Category 2	Category 3	Category 4	Etc.
1	x						
5		x					
10			x	x			
Etc.							

Taula 2.8 Exemple de matriu de compatibilitat

Per tant, com s'observa en aquest exemple, els TOG que tinguin un rank d'entre 1 i 4 se'ls assignaran seients del sector Tribuna VIP. En canvi els que el tinguin de 5 a 9 els reservarem seients confortables.

2.3 Algorisme

L'algorisme de distribució d'espectadors ha d'implementar una manera ràpida i fiable d'assignar les ordres (TOG) als seients. Ha de ser capaç de produir una assignació complerta i correcte per a tot un circuit en un temps limitat. Per definir quan una assignació és correcte la FIA (Federació Internacional de l'Automòbil) estableix una sèrie de regles i restriccions que han de complir les assignacions vàlides. Algunes regles són fixes i les altres són configurables.

Hi pot haver casos en què puguin haver-hi vàries solucions d'assignació diferents que compleixin amb les restriccions imposades. D'altra banda hi pot haver altres situacions en què les restriccions siguin tan restrictives o les dades d'entrada tan saturades que no hi hagi cap manera per realitzar una assignació complerta amb aquestes circumstàncies. En ambdós casos l'algorisme ha de ser capaç de trobar la millor assignació possible. Per determinar quan una solució és millor que una altra es definirà una funció de fitness amb la qual cada solució obtindrà un valor, essent el valor més alt la millor solució.

2.3.1 Regles

FEDERATION INTERNATIONALE DE L'AUTOMOBILE

El protocol de la FIA estableix quines regles s'han de complir a l'hora de distribuir les ordres pels seients. Hi ha regles que s'hauran de complir sempre, i que les dades d'entrada (seients disponibles i ordres) seran compatibles perquè això sigui possible. Aquestes regles seran les que obligaran a assignar els tiquets d'una determinada categoria als seients de la mateixa categoria. També inclouen seients que són d'un tipus determinat als quals s'hi han d'assignar els tiquets corresponents.

També hi haurà regles que es podrà decidir si es vol que es compleixin o no, i els seus paràmetres podran ser modificats. Aquestes regles no obligatòries marquen les pautes per realitzar una distribució correcte tant des del punt de vista dels espectadors com dels organitzadors.

Des del punt de vista de l'espectador es vol que els seients assignats siguin els que tenen millor visibilitat de la pista (per tant millor rànquing) i que la distribució dels seients d'una mateixa ordre sigui el màxim d'unida i agrupada possible, assignant tots els tiquets junts, no tots en una única i llarga fila sinó en diferents files però minimitzant les distàncies de tots els seients entre ells (distància tots amb tots).

Per altra banda, des del punt de vista dels organitzadors els criteris de distribució són ben diferents. Per exemple no voldran que tots els millors seients (els de rànquing més alt) siguin assignats a un sol client amb un gran nombre d'entrades quan hi hagi altres clients amb el seu mateix rànquing, sinó que preferiran que es divideixi aquest gran nombre d'entrades en grups més petits i repartir els seients de forma equitativa per a tots els clients del mateix rànquing. O encara que el rànquing d'aquest client amb moltes entrades sigui més alt i per rànquing li pertoquin els millors seients, hi haurà certs clients amb els que es voldrà que les seves ordres no es distribueixin mai juntes al mateix bloc, ja que això podria provocar per exemple que tots els individus es vestissin amb la mateixa indumentària i fessin publicitat que no han pagat. Per tant els clients marcats amb l'atribut de dispersió activat se'ls hauran d'assignar les ordres en

diferents blocs, o si es vol en un bloc amb unes distàncies entre elles suficients perquè no es pugui produir aquest fet. Un altre criteri que recentment han volgut tenir en compte els organitzadors és que quan no es venen totes les entrades disponibles es distribueixin els tiquets uniformement per tots els blocs per donar així la sensació des de fora (a les televisions per exemple) que el circuit és ple.

Aquest segon tipus de regles no són tan estrictes com les anteriors i s'intentaran seguir però si no és possible trobar una solució d'assignació complint-les, hi haurà l'opció de poder anar relaxant els seus paràmetres o desactivant progressivament seguint un odre preestablert fins que es pugui trobar una solució.

Tot seguit es definiran més concretament totes les regles classificant-les en els dos grups anteriorment esmentats: Regles obligatòries i Regles que poden desactivar-se o ser relaxades.

2.3.1.1 Regles obligatòries

1 Totes les ordres s'han d'assignar a seients de la seva respectiva 'Category'.

2 Totes les odres s'han d'assignar a seients del seu 'Status'.

Aquestes 2 regles obligatòries no poden ser mai desactivades, i les dades d'entrada seran compatibles amb elles. Per exemple el nombre de seients d'una determinada 'Category' o d'un 'Status' sempre serà més gran o igual al nombre de tiquets venuts del mateix.

2.3.1.2 Regles que poden desactivar-se o bé ser relaxades

3 Les ordres s'han d'assignar al 'Sector' que els marqui la matriu de compatibilitat.

4 Les odres s'han d'assignar a seients del seu 'Type'.

5 Les ordres s'han d'assignar a seients del seu 'Price Type'.

6 Els TOG s'han de dividir en subgrups (Ticket Order Subgroups - TOS) amb una mida estàndard, una desviació respecte aquesta mida i un residu. Per exemple, si la mida estàndard és de 30, amb una desviació de ±5, i amb un residu de 4; si hi ha un TOG de 124 unitats, aquest pot ser partit en tres grups de 30 i un de 34, o també en 4 grups de 35, 25, 30 i 34. D'altra banda si el TOG fos de 126 unitats, podria ser partit en 4 grups de 30 i un de 6. Això significa que es podran crear grups heterogenis de diferents quantitats de tiquets sempre i quan es compleixin el màxim, el mínim i el residu.

7 S'han d'assignar primer els tiquets de rànquing més alt. És a dir, que als tiquets de rànquing més alt els han de tocar els seients de rànquing també més alt (sempre que no s'infringeixi alguna altra regla).

8 Hi haurà un màxim i un residu de seients consecutius assignats en una fila per a un mateix TOS. Per exemple si el màxim se seients consecutius per fila és 10 i el residu 3, un TOS de 33 tiquets es pot assignar en 3 files de 10 i una de 3, o en 2 files de 10 i una de 13.

9 Mai pot quedar una fila amb un sol seient assignat (quan assignem una ordre de més d'un tiquet).

							Y
				Y	Y	Y	Y
						X	X
					X	X	X
X	X	X	X	X			

En els exemples la "X" representa una bona assignació i la "Y" una d'incorrecta.

10 Si tots els tiquets d'un TOS no caben en el mateix bloc, la odre es podrà partir sempre i quan el nombre de tiquets de cada subgrup superi un nombre mínim d'unitats.

11 Dos TOS del mateix client amb el flag de dispersió activat (s'escollirà una de les dues opcions):

- No podran ser assignats mai al mateix bloc
- Podran anar en el mateix bloc sempre i quan es deixi una distància mínima entre els dos TOS (mesurada en seients).

12 S'ha d'evitar deixar seients buits als laterals de les files.

X	X	X		Y	Y	Y	
	Y	Y	Y		X	X	X
X	X	X	X				
	Y					X	X
Y	Y	Y				X	X

13 Hi ha d'haver l'opció de distribuir uniformement els seients per intentar no donar mai la sensació que algunes zones dels blocs del circuit estan buides.

X		X	X	X		Y	Y	Y	Y	Y
X	X	X		X		Y	Y	Y	Y	Y
X		X	X	X						
X	X	X		X						
X		X	X	X		Y	Y	Y	Y	Y

2.3.1.3 Relaxacions

L'algorisme podrà tenir una llista de relaxacions. Una relaxació significa o bé desactivar alguna de les regles no obligatòries o bé modificar algun dels seus paràmetres. Quan l'algorisme es trobi en situacions on no és possible assignar les ordres complint les regles fixades a l'inici, provarà d'efectuar alguna relaxació per tal d'intentar prosseguir amb el procés d'assignació. La llista de relaxacions definirà l'ordre en què l'algorisme anirà relaxant o desactivant automàticament regles en els casos que calgui.

2.3.2 Configuracions de l'algorisme

L'algorisme serà configurat explícitament amb una sèrie de paràmetres que li indicaran de quina manera ha de ser executat. Els paràmetres de l'algorisme seran els següents:

- Activació / desactivació explícita de cadascuna de les regles no obligatòries.
- Paràmetres de cadascuna de les regles que en necessitin (Seients màxims consecutius per fila, residu, distància mínima entre ordres d'un mateix client, etc).
- Llista de relaxacions (podrà estar buida, indicant que no se'n poden fer).

També s'haurà de configurar per paràmetre on es troben les dades d'entrada i on s'han de desar les dades de sortida.

Una altra configuració de l'algorisme serà la condició de finalització d'una assignació concreta. Tindrem tres possibles condicions de finalització:

- Només és una considera solució de l'algorisme l'assignació de totes les ordres.
- Es considera solució de l'algorisme una assignació parcial de les ordres amb un informe del conjunt d'ordres no assignades.
- Quan l'algorisme es trobi amb que no pot assignar la totalitat de les ordres, relaxarà o desactivarà automàticament una sèrie de regles definides prèviament per tal d'aconseguir assignar-les. Aquesta configuració donarà com a resultat, a part de les ordres assignades i les no assignades (en el cas que no s'hagin pogut assignar totes), un informe on s'especificarà quines regles han estat relaxades o desactivades en les assignacions.

L'execució de l'algorisme serà en "temps real" entenent per temps real que el procés d'assignació podrà ser interromput en qualsevol moment per l'usuari. En aquest moment l'algorisme mostrarà la millor solució trobada fins al moment i una funció de fitness donarà una mesura de la bondat de la solució. Altres formes configurables per aturar l'execució automàticament seran:

- quan l'algorisme hagi trobat la primera solució
- quan l'algorisme hagi trobat *n* solucions diferents
- quan l'algorisme hagi trobat una solució que superi un fitness determinat
- quan hagi transcorregut un temps determinat.

També s'haurà de configurar un paràmetre *m* que indicarà quantes solucions diferents ha d'emmagatzemar l'algorisme. Per tant, el resultat de l'algorisme serà de com a màxim *m* solucions, que seran les *m* millors assignacions trobades i mesurades segons la funció de fitness.

2.4 Dades de sortida

Un cop el procés s'ha completat, tindrem una llista de seients assignats a ordres. Aquesta solució anirà associada a un valor de fitness. Aquest valor determinarà el grau de bondat de la solució i vindrà calculat per una funció que ha de tenir en compte tots els criteris utilitzats a l'hora de decidir què és una bona solució i què no ho és. Aquest grau de bondat pot ser per exemple un número entre 0 i 100.

Depenent de com s'hagi configurat l'algorisme, també es pot donar com a resultat una llista amb les odres que no s'han pogut assignar i un report amb el log de l'execució on quedarà clar quines regles s'han desactivat o relaxat i en quin moment.

Exemple de dades de sortida:

Valor de la funció de fitness: 90
Ordres assignades:

Block	Row	Seat	Request	TOG	TOS	Resultat
1	1	1	1	1	1	Ok
1	1	2	1	1	1	Ok
1	1	3	1	1	1	Ok
24	12	15	68	4	2	Relaxat R1

Taula 2.9 Exemple de taula d'ordres assignades

Ordres no assignades:

Request	TOG	TOS	Client Id	Number of Tickets	Category	Rank
23	1	1	12	21	3	18
24	1	1	18	40	4	61
24	1	2	18	30	4	19

Taula 2.10 Exemple de taula d'ordres no assignades

2.5 Altres requeriments

L'aplicació s'ha de desenvolupar amb Microsoft Visual Studio .NET, per poder així ser integrada fàcilment amb els sistemes presents en els equips informàtics actuals.

Per tal de poder conèixer l'estat d'una execució, la interfície disposarà de:

- un semàfor que indicarà quan s'ha trobat la primera solució
- un marcador que indicarà el percentatge de resolució de la primera solució
- una taula on s'indicaran els valors de fitness de les *m* millors solucions que s'hagin trobat.

L'algorisme també ha de tenir la possibilitat d'assignar un conjunt d'ordres a un únic bloc o un conjunt de blocs determinat sense tenir en compte la totalitat dels seients disponibles. L'algorisme rebrà com a entrada una llista d'ordres i un circuit (o un conjunt dels blocs oportuns) que pot estar buit o amb algunes localitats ocupades.

3 Disseny

3.1 Introducció

En aquest capítol s'analitzaran les diferents tècniques i mètodes que hi ha per resoldre el problema de la distribució dels espectadors complint amb els requeriments exposats al capítol anterior. Es decidirà quina tècnica s'escull i es justificarà l'elecció.

3.2 Arquitectura del sistema

L'aplicació estarà formada per quatre blocs bàsics: un primer que servirà per configurar l'algorisme i els altres tres que seran els encarregats de carregar les dades, realitzar l'assignació i emmagatzemar els resultats:

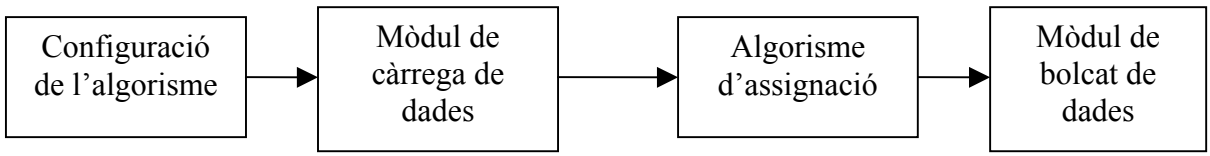


Figura 3.1 Esquema general del sistema

3.2.1 Configuració de l'algorisme

Aquest bloc estarà format per una interfície on l'usuari configurarà l'algorisme. La configuració inclou tots els aspectes exposats a l'apartat 2.3.2 del capítol de requeriments: especificar les dades d'entrada (blocs, seients, ordres), determinar les regles que s'hauran de complir en l'assignació, la condició de finalització, l'ordre de les relaxacions, etc.

3.2.2 Mòdul de càrrega de dades

El mòdul de càrrega de dades serà l'encarregat de proporcionar a l'algorisme les dades necessàries per a realitzar l'assignació. Les dades que l'algorisme necessita per tal de realitzar una assignació són el blocs del circuit, els seients de cada bloc, les ordres de compra de tiquets, els clients que compren, la matriu de compatibilitat i les categories a les que pertanyen els blocs i els seients. Les dades que es passaran a l'algorisme seran:

- Conjunt de blocs
- Conjunt de seients
- Conjunt d'ordres
- Conjunt de clients (Market segment, Customer Groups, Group Sales customer)
- Matriu de compatibilitat
- Conjunt de categories

Aquestes dades s'introduiran en l'estructura de dades interna de l'algorisme, seguint aquest model entitat-relació:

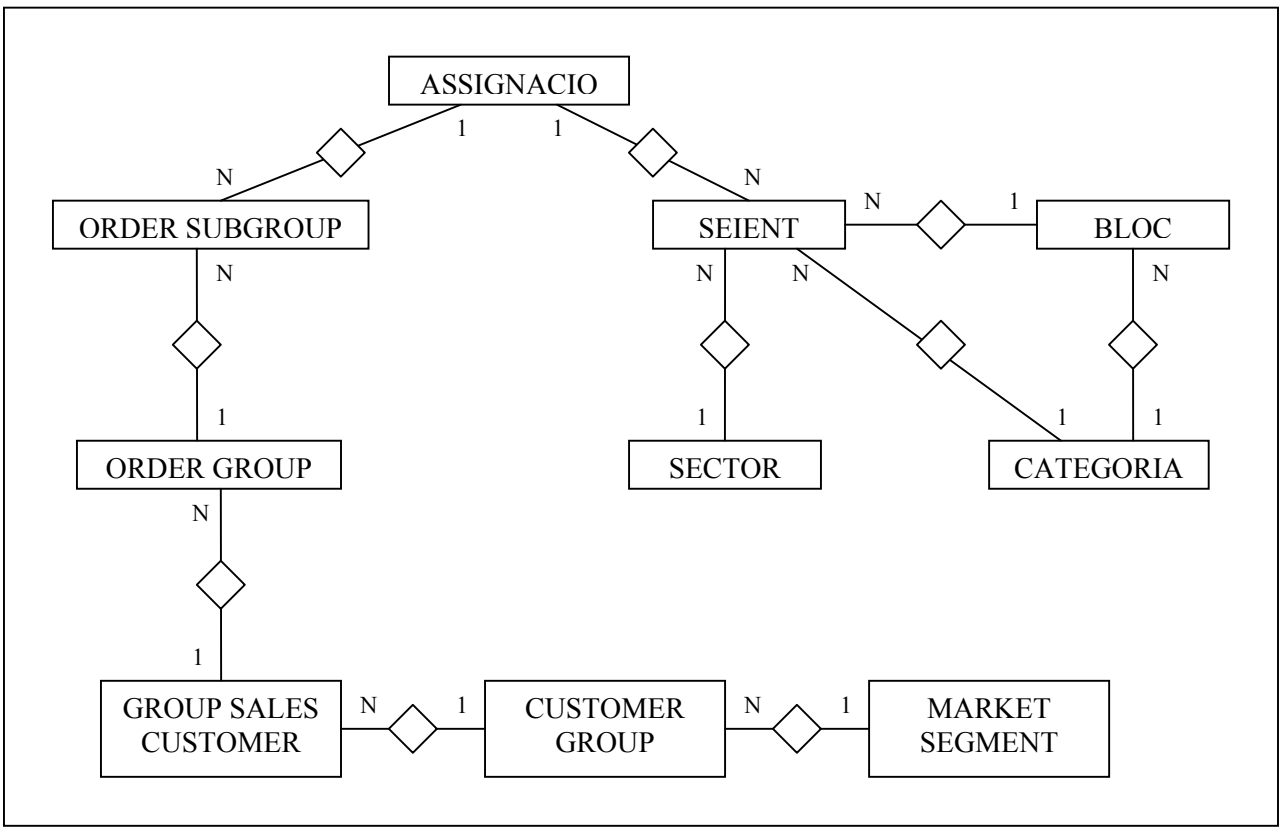


Figura 3.2 Diagrama Entitat - Relació

La majoria de dades s'entraran directament al sistema:

- Market Segments
- Customer Groups
- Group Sales Customer
- Ordres (Order group)
- Seients
- Blocs
- Categories

I les altres s'hauran de generar.

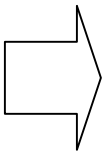
- Order subgroup
- Sectors
- Assignació

3.2.2.1 Order subgroup

Aquesta taula és una expansió de la taula Order group. Com indica la regla no obligatòria número 6 cadascuna de les ordres de la taula Order group (TOG) es partirà en un conjunt d'order subgroups (TOS) amb una mida estàndard (en l'exemple 40) determinada a la configuració per l'usuari.

Llista de TOG

Client	Tiquets	TOG
Apple	20	1
Telefonica	300	2
RedBull	40	3
Fanta	210	4
Repsol	380	5
Gillette	40	6
Hyundai	2	7
Marlboro	180	8
PHILIPS	6	9
Movistar	8	10



Llista de TOS

Client	Tiquets	TOS
Apple	20	1
Telefonica	40	2
Telefonica	40	3
Telefonica	40	4
Telefonica	40	5
Telefonica	40	6
Telefonica	34	7
Telefonica	33	8
Telefonica	30	9
Telefonica	3	10
RedBull	22	11
RedBull	18	12
Fanta	40	13
Fanta	40	14
Fanta	40	15
Fanta	40	16
Fanta	40	17
Fanta	10	18
Repsol	40	19
Repsol	40	20
Repsol	40	21
Repsol	40	22
Repsol	40	23
Repsol	40	24
Repsol	40	25
Repsol	40	26
Repsol	30	27
Repsol	22	28
Repsol	6	29
Repsol	2	30
Gillette	40	31
Hyundai	2	32
Marlboro	46	33
Marlboro	46	34
Marlboro	38	35
Marlboro	25	36
Marlboro	25	37
PHILIPS	6	38
Movistar	8	39

Figura 3.3 Exemple de creació de la taula de TOS

Una de les raons d'aquesta partició com s'explica a l'apartat 2.3.1 del capítol de requeriments és que els organitzadors no volen que els seients de rànquing més alt siguin assignats tots per a un sol client amb un gran nombre d'entrades. Per tant a més a més de partir en TOS, aquestes particions s'hauran de repartir amb un mètode de disgregació, és a dir, que s'aniran barrejant al màxim possible les particions (TOS) entre els diferents clients.

Passos del mètode de disgregació:

1 Per a cada client calcular el percentatge d'ordres que se li han d'assignar respecte el total d'ordres del client (inicialment, tots ells seran 100%).
2 Agafar el més gran (tiquets) TOS amb el major percentatge de tiquets per assignar.
3 Si hi ha més d'un client amb el mateix percentatge agafar primer el client amb més tiquets per assignar.
4 Repetir 1, recalculant percentatges i repetir el procés fins que tots els TOSs hagin estat processats.

Aquesta disgregació es farà a l'inici de l'algorisme i llavors s'anirà modificant l'ordre dels TOS en funció de les necessitats però sempre tenint en compte les regles.

List of Orders to Allocate		
Customer	Tickets	TOS
Apple	20	1
Telefonica	40	2
Telefonica	40	3
Telefonica	40	4
Telefonica	40	5
Telefonica	40	6
Telefonica	34	7
Telefonica	33	8
Telefonica	30	9
Telefonica	3	10
RedBull	22	11
RedBull	18	12
Fanta	40	13
Fanta	40	14
Fanta	40	15
Fanta	40	16
Fanta	40	17
Fanta	10	18
Repsol	40	19
Repsol	40	20
Repsol	40	21
Repsol	40	22
Repsol	40	23
Repsol	40	24
Repsol	40	25
Repsol	40	26
Repsol	30	27
Repsol	22	28
Repsol	6	29
Repsol	2	30
Gillette	40	31
Hyundai	2	32
Marlboro	46	33
Marlboro	46	34
Marlboro	38	35
Marlboro	25	36
Marlboro	25	37
PHILIPS	6	38
Movistar	8	39

List of Orders Ranked		
Customer	Tickets	TOS
Repsol	40	19
Telefonica	40	2
Fanta	40	13
Marlboro	46	33
RedBull	22	11
Gillette	40	31
Apple	20	1
Movistar	8	39
PHILIPS	6	38
Hyundai	2	32
Repsol	40	20
Telefonica	40	3
Fanta	40	14
Repsol	40	21
Marlboro	46	34
Telefonica	40	4
Repsol	40	22
Fanta	40	15
Telefonica	40	5
Repsol	40	23
Marlboro	38	35
Repsol	40	24
Telefonica	40	6
RedBull	18	12
Fanta	40	16
Repsol	40	25
Telefonica	34	7
Repsol	40	26
Marlboro	25	36
Fanta	40	17
Telefonica	33	8
Repsol	30	27
Marlboro	25	37
Telefonica	30	9
Repsol	22	28
Fanta	10	18
Repsol	6	29
Telefonica	3	10
Repsol	2	30

Figura 3.4 Exemple de resultat del mètode de disgregació

L'algorisme rebrà com a entrada els TOS disgregats (part dreta) i els assignarà en aquest ordre començant pel primer fins a l'últim.

3.2.3 Assignació

La taula d'assignacions de l'esquema entitat-relació contindrà les solucions del procés, és a dir l'associació de cada tiquet a un seient del circuit. Aquesta és la taula que haurà de generar l'algorisme.

3.2.4 Mòdul de bolcat de dades

El mòdul de bolcat de dades serà l'encarregat d'emmagatzemar els resultats de l'algorisme emmagatzemats a la taula d'assignacions en un o varis fitxers de sortida.

Podem arribar a generar fins a tres dades de sortida depenent de com hagi estat configurada l'aplicació. Aquestes són les ordres assignades, les ordres no assignades i un registre de les incidències rellevants ocorregudes durant el procés.

3.3 Algorisme

L'algorisme el que ha de fer a grans trets és assignar cadascun dels tiquets de les ordres a seients físics del circuit. Aquesta assignació s'ha de realitzar complint amb les regles i restriccions imposades per la FIA.

Hi pot haver moltes solucions d'assignació diferents que compleixin amb les restriccions. L'algorisme no es limitarà a trobar-ne una, sinó que haurà de ser capaç de trobar la millor assignació possible.

El fet de trobar la millor assignació possible no és gens fàcil. La intel·ligència artificial proporciona diferents tècniques i mètodes per mirar de trobar sinó la millor, una distribució prou bona. Tot seguit es presentaran diferents mecanismes aplicables al problema de la distribució d'espectadors:

3.3.1 Alternatives

1. CBR (Case Based Reasoning)

   La idea principal dels CBR [Aamodt & Plaza 94] es basa en resoldre nous problemes adaptant solucions anteriors que es van fer sevir per resoldre antigues situacions del mateix problema. En el nostre cas, si tenim exemples resolts anteriorment de diferents blocs amb ordres assignades correctament, podem intentar assignar les dades d'entrada (seients, tiquets venuts) actuals a partir d'assignacions de blocs i ordres semblants.

   Tot i que és una solució viable creiem que no és convenient degut a que el protocol de la FIA està en constant evolució. Aquest dinamisme provocaria que la majoria de casos emmagatzemats no fossin útils per a l'assignació actual.

2. Sistemes experts

   Els sistemes experts són programes que simulen el procés de resolució de problemes que empra un expert. Emmagatzemen coneixement sobre un domini específic i ben definit i el fan servir per resoldre problemes en aquest domini mitjançant la realització d'inferències lògiques [Rich 1991].

   Per resoldre el problema de la distribució d'espectadors amb un sistema expert hauríem de definir primer el domini, que en aquest cas serien els seients i els tiquets. I també hauríem de definir les regles lògiques. No s'han de confondre les regles dels sistemes experts amb les regles d'assignació definides al capítol de requeriments. Les regles que utilitzen els sistemes experts són fórmules lògiques amb les quals es poden realitzar inferències a partir de les entrades per generar una solució. Aquestes regles haurien de contemplar tots els casos d'entrada possibles i la inferència obtinguda d'aquestes hauria de ser capaç d'assignar totes les ordres donant una solució final òptima.

   El problema intentant enfocar el problema amb aquesta tècnica és que la quantitat de regles necessàries per poder obtenir la solució òptima contemplant qualsevol conjunt possible d'entrada seria extremadament immensa, ja que no es pot construir una regla senzilla i genèrica que ens assigni sempre les ordres òptimament.

3. Algorismes genètics

   Un algorisme genètic (AG) és una tècnica de programació que imita l'evolució biològica com a estratègia per a resoldre problemes [Luis 1993]. Donat un problema específic a resoldre, l'entrada de l'AG és un conjunt inicial de solucions al problema, codificades d'alguna manera, i una mètrica o funció de fitness que avalua quantitativament a cada candidata. D'aquest conjunt inicial de solucions se'n conserven les millors i se'ls permet reproduir-se amb una petita probabilitat de mutar, generant així una nova població de solucions que en conjunt serà millor que la primera. La teoria és que aquestes successives generacions convergeixen cap a la solució òptima.

   És molt difícil aplicar aquesta tècnica al nostre cas. En primer lloc, per generar les successives generacions s'han de combinar solucions (amb un mètode de creuament) i produir mutacions amb l'objectiu de crear descendents millors. Confeccionar un mètode de creuament per el nostre cas no seria gens fàcil. En segon lloc, l'AG necessita una població inicial de solucions que després anirà millorant. Normalment la primera població es genera en els AG de forma aleatòria, però fent-ho així en el nostre cas es generarien solucions completament errònies (amb valors de fitness inevitablement iguals a zero en totes les solucions). Si per altra banda, volem generar una població inicial de solucions amb un valor de fitness acceptable, la tasca de generar cada individu de la població és ja prou complexa com per requerir l'aplicació d'alguna altra tècnica d'IA.

4. CSP

   Un problema de satisfacció de restriccions (CSP) és un problema de decisió amb un nombre finit d'opcions on es disposa d'un conjunt fixat de decisions a fer. Cada decisió involucra triar una de les opcions del conjunt. Cada restricció (constraint) restringeix la combinació d'opcions que es poden prendre simultàniament. La tasca consisteix en prendre totes les decisions satisfent totes les restriccions. Als CSP on algunes solucions són millors que les altres, l'objectiu és trobar la solució òptima [Tsang 1993].

La opció que hem triat ha estat representar el problema com un Problema de Satisfacció de Restriccions (Constraint Satisfaction Problem - CSP) i utilitzar tècniques generals de satisfacció de restriccions per tal de generar possibles solucions. En el següent apartat s'explicarà més a fons com aplicar el CSP al nostre cas.

3.3.2 Constraint Satisfaction Problems (CSP)

Els CSP formalment especifiquen un conjunt de decisions a prendre i un conjunt de restriccions que limiten les possibles combinacions de decisions vàlides. Les decisions es descriuen mitjançant variables, i a cada variable se li pot assignar un valor del seu domini de valors. Les restriccions es descriuen com a relacions que especifiquen quines assignacions de valors es poden fer per cada variable.

Si ens ho mirem des del punt de vista del problema en qüestió, les variables serien seient i espectador. Una decisió seria un conjunt d'aquestes dues variables, és a dir, l'assignació d'un espectador a un seient. El domini de valors de la variable seient seria el conjunt de seients del circuit i el domini d'espectador seria cadascunes de les persones que han reservat un seient. Finalment, les restriccions serien les regles establertes per la FIA definides a l'apartat 2.3.1 del capítol de requeriments.

Una vegada tinguem definides les relacions que ens marquen les restriccions del problema, mitjançant una tècnica de satisfacció de restriccions buscaríem la solució òptima. Dins d'aquestes tècniques, les més habituals són les que construeixen arbres de decisió, i utilitzen heurístiques per arribar a una solució.

Concretament el que faria l'algorisme basat en CSP és anar construint un arbre amb totes les possibles solucions al problema i, una vegada construït, buscar la solució òptima segons un criteri preestablert (funció de fitness). En la construcció de l'arbre, els nodes representarien una assignació d'un espectador a un seient. De cada node en sortiran un seguit de fills, que representaran les possibles assignacions d'espectadors al següent seient. Així successivament fins a assignar tots els espectadors. Cada branca de l'arbre donarà una possible solució al problema. A més a més, cada node tindrà associat un valor que indicarà el nivell d'adequació de l'espectador al seient. Així doncs, el nivell d'adequació d'una solució concreta vindrà determinada per la suma de nivells d'adequació de tots els nodes que formen la branca de l'arbre. Finalment, la solució òptima al problema serà la branca que tingui el nivell d'adequació més alt.

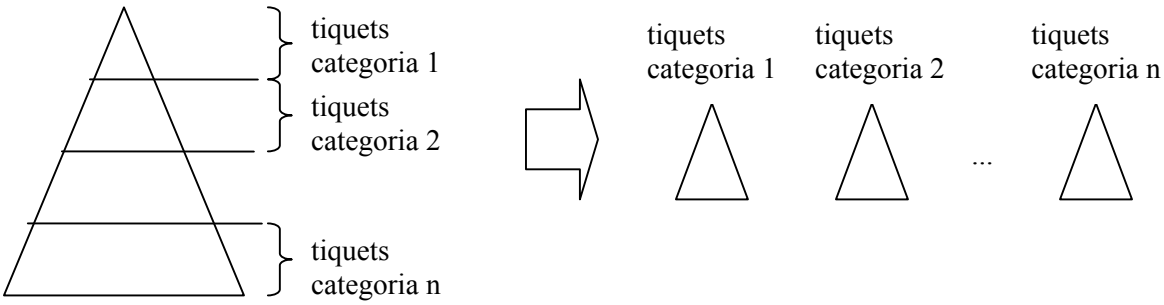
El problema de les tècniques de satisfacció de restriccions és que tenen una complexitat computacional NP. Això significa que el temps per resoldre el problema creix de forma exponencial amb el volum de dades de l'entrada. Principalment, això és degut a que només podrem garantir que s'ha trobat la solució òptima al problema si calculem totes les solucions possibles. Per tant l'elaboració d'una tècnica exhaustiva per cercar la solució òptima no serà adequada ja que el temps de computació necessari seria inviable tècnicament.

Però els organitzadors tampoc exigeixen trobar la solució òptima, per aquest motiu l'objectiu serà trobar sinó la solució òptima una de propera. Per fer-ho utilitzarem tècniques per millorar l'eficiència de computació dels algorismes de cerca en CSP. Dins de les tècniques més habituals per a millorar l'eficiència de la construcció d'arbres de decisió hi podem trobar el Branch and Bound, estimadors de poda i multitud de tècniques heurístiques [Kumar 1992]. L'algorisme disposarà d'algunes d'aquestes tècniques per tal de millorar l'eficiència.

3.3.3 Esquema de l'algorisme

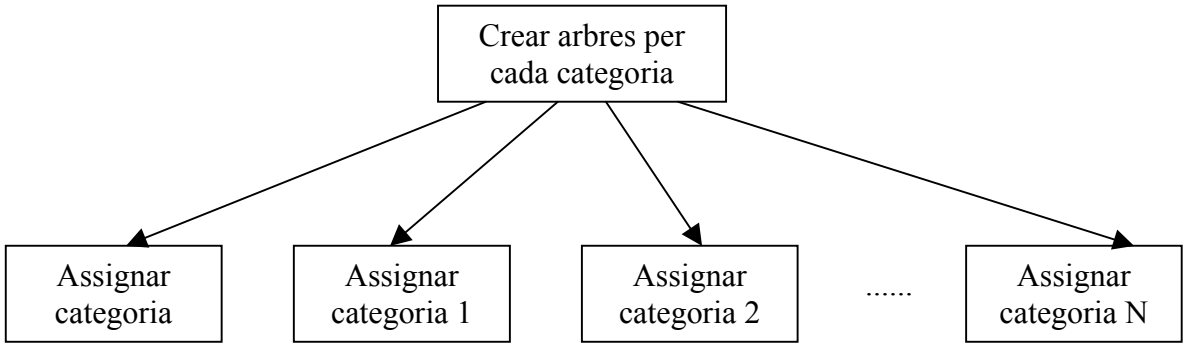
L'algorisme ha d'assignar totes les ordres als seients necessaris. Com s'ha dit en l'apartat anterior representarem l'algorisme CSP com una cerca en un espai d'estats. L'espai d'estats és un arbre de decisió on cada node representa una assignació d'un tiquet a un seient.

Les primera de les regles obligatòries ens diu que les categories dels seients i dels tiquets han de coincidir obligatòriament. Aquesta restricció la podem fer servir per dividir l'espai de cerca en tantes categories com haguem de considerar.

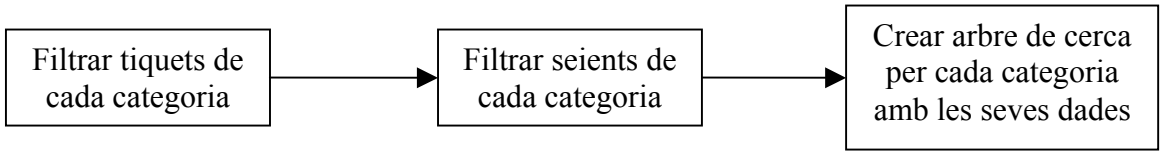


Aquesta divisió del problema té l'avantatja que l'espai de cerca es redueix enormement al dividir un sol arbre en *n* arbres més petits. La cerca en cadascun d'aquests subarbres s'efectuarà paral·lelament, s'aniran recorrent al mateix temps tots els arbres. La solució serà una fusió dels resultats de les cerques de tots els arbres.

Així doncs, l'algorisme haurà de crear tants processos independents com categories s'hagin d'assignar.



Cada procés tindrà només les dades corresponents a la seva categoria (seients, ordres). Per tant el mètode de "Crear arbres per cada categoria" per cada subarbre ha de fer:



3.3.4 Procés d'assignació d'una categoria

Aquest és el procés fonamental del projecte. El procés rep com a entrades els seients d'una categoria i els tiquets o ordres venuts per a aquesta categoria i ha d'assignar cada tiquet a un seient físic complint les restriccions.

L'objectiu de l'algorisme és trobar la millor assignació. La funció de fitness que s'exposa a l'apartat 3.4 d'aquest mateix capítol ens dona un grau de bondat d'una solució. Per tant l'algorisme haurà de trobar l'assignació que maximitzi aquesta funció.

El problema és que les possibilitats d'assignacions en una sola categoria continuen essent inabordables. Per fer-nos-en una idea, podem pensar que cada tiquet pot ser assignat a tots els seients disponibles. Per tant el primer tiquet tindrà tantes possibles assignacions com seients hi hagi a la categoria. Les possibilitats del següent tiquet seran tots els seients excepte el que ja està ocupat per la primera assignació. I així successivament.

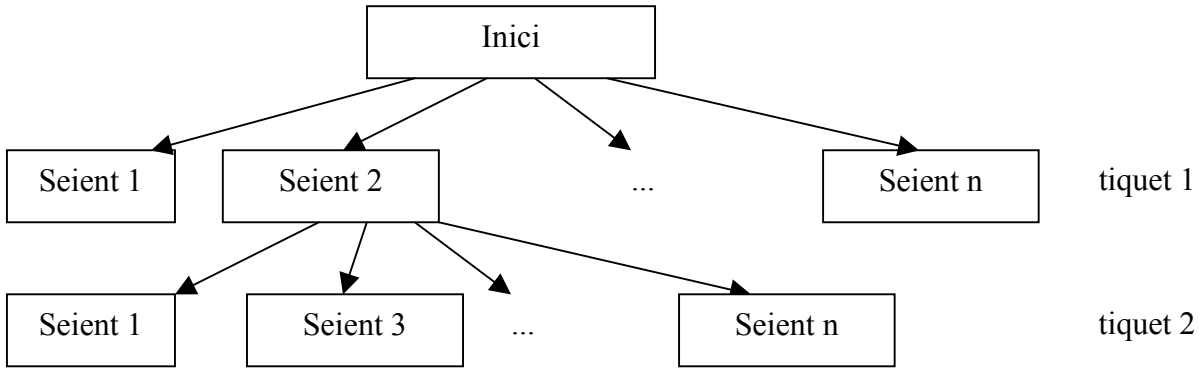


Figura 3.8 Espai de cerca per a una categoria

L'espai de cerca per a aquest arbre es pot expressar amb la fórmula:

$$\frac{n!}{(n-m)!}$$

on n és el nombre de seients de la categoria i m és el nombre de tiquets per assignar ($m \leq n$).

Si tenim una categoria amb uns quants milers de seients on s'hi han d'assignar uns altres milers de tiquets veiem que l'espai de cerca creix a un ritme desmesurat. Per exemple una categoria de mida mitjana amb 30000 seients on volem encabir-hi 20000 espectadors tenim:

$$\frac{30000!}{(30000-20000)!} = \frac{30000!}{10000!}$$

L'espai de cerca és prou monstruós com per descartar immediatament la possibilitat d'un algorisme exhaustiu que provi totes les possibilitats i esculli d'aquestes la que tingui un valor més alt en la funció de fitness. De fet el nombre de possibilitats d'aquest simple cas és tan gran que no podríem explorar-les encara que tinguéssim tant temps com la vida de l'univers.

3.3.4.1 Region growing

Per mirar de simplificar el problema podem pensar en un algorisme que es fa servir en visió per computador per realitzar segmentacions i que, encara que no té res a veure amb l'assignació de seients, el concepte servirà per entendre l'aplicació que en podem fer en el nostre cas.

En visió per computador per segmentar les regions d'una imatge una de les tècniques que es fa servir és l'algorisme del *region growing* [Zucker 1976]. Aquest mètode consisteix bàsicament en 'plantar' una llavor a la imatge, de manera que quan aquesta llavor vagi creixent ocupi tots els píxels d'una regió determinada.

Podem aplicar aquest mateix concepte per realitzar les assignacions. Podem pensar que les ordres que volem assignar són les regions i que els blocs de seients són la imatge. Sota aquest punt de vista per cada ordre hem de triar una llavor que serà un seient en un bloc i farem créixer aquesta llavor fins que ocupi tots els tiquets que pertanyen a la ordre. Aquesta temptativa és del tot escaient ja que per definició les ordres han d'estar juntes, per tant les possibilitats d'assignació de cada tiquet no són com es pressuposava tots els seients disponibles sinó un conjunt petit proper a la llavor. Aquesta aproximació també ens facilitarà molt la tasca de fer complir les restriccions ja que al mateix temps que fem créixer la regió podem anar acotant la zona on pot expandir-se per tal de satisfer les regles.

Amb aquest nou enfocament reduïm considerablement l'espai de cerca. Ara ja no tenim tots els seients del circuit com a candidats per cada tiquet, sinó que només els tenim per l'elecció de la llavor. Un cop s'hagi escollit la llavor la resta de tiquets de l'ordre s'assignaran fent créixer aquesta llavor. Tenint en compte aquesta solució podem definir l'espai de cerca de forma aproximada amb:

$$\frac{n!}{(n-p)!}$$

on p és el nombre d'ordres que hem d'assignar

En l'exemple d'abans si tenim una mida d'ordre de 40 tiquets, la p podria ser 500 (ja que 500·40 = 20000) i l'espai de cerca seria aproximadament:

$$\frac{30000!}{(30000-500)!} = \frac{30000!}{29500!}$$

Aquesta fórmula redueix molt l'espai de cerca en comparació amb l'anterior ja que p << m. Tot i així l'espai de cerca continua essent massa immens com per continuar descartant la cerca exhaustiva.

De fet podem arribar a la conclusió que qualsevol intent de reduir o podar l'espai de cerca (backtracking, backjumping, simulated annealing, tabú search, etc) seguirà donant un univers de possibilitats massa ampli com per contemplar la possibilitat d'explorar-lo tot sencer buscant la solució òptima.

3.3.4.2 Algorisme proposat

En aquests casos el que se sol fer en la Intel·ligència Artificial són algorismes no exhaustius basats en heurístiques que trobin una solució prou bona en un temps determinat [Reeves 1993]. Són algorismes progressius que com més temps d'execució se'ls doni millor serà la solució que trobaran.

La solució que proposem és un algorisme que trobi ràpidament una primera solució que sigui prou propera a la òptima i tot seguit un procés de "backtracking especial" que modifiqui aquesta solució per millorar el seu fitness.

Una primera temptativa per definir una funció heurística seria intentar assignar primer les ordres de mida més gran i després les més petites; d'aquesta manera s'afavoriria que s'acabessin assignant totes les ordres. Però això no és del tot cert si tenim en compte dos aspectes especials del nostre problema com són les regles i la funció de fitness. En primer lloc les regles obliguen a assignar les ordres agrupadament, fet que si utilitzem aquesta heurística generarà molt possiblement forats ens els blocs que seran difícilment emplenables més endavant. I en segon lloc la funció de fitness valora que s'assignin les ordres de rànquing més alt als seients de rànquing també més alt, cosa que no es seguiria de cap manera amb aquest mètode, amb la conseqüent davallada del seu fitness.

Aprofitant la característica del rànquing en la funció de fitness, podem definir una nova funció heurística que consisteixi en assignar les ordres per ordre de rànquing.

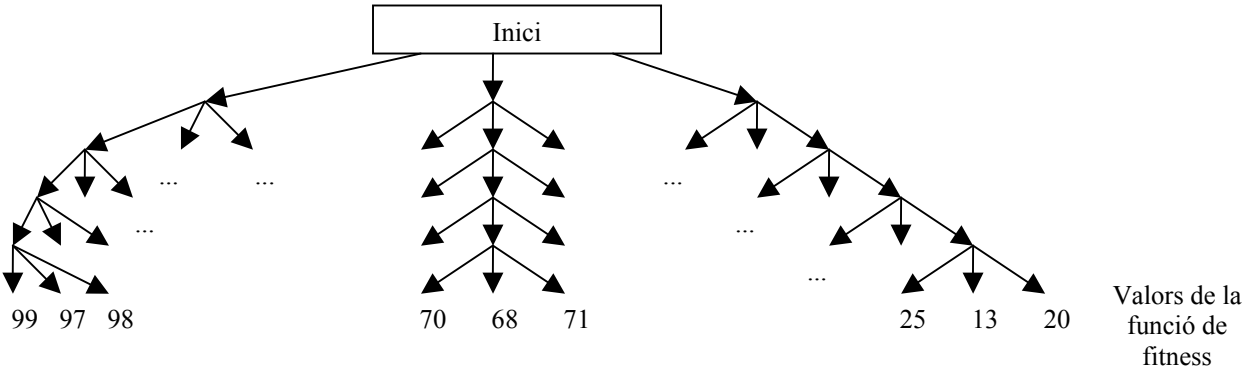


Figura 3.9 Arbre generat amb una heurística basada en el rànquing de les ordres

Si ens mirem l'arbre de cerca creat amb aquesta heurística (figura 3.9) ens adonarem que aquí sí que l'arbre està molt balancejat, és a dir, les solucions amb valors més alts de fitness es troben bastant agrupats a les primeres branques de l'arbre. Per tant una heurística força fiable per trobar una bona primera solució serà començar assignant les ordres de rànquing més alt als millors seients. En una situació ideal seguint aquesta estratègia mai es deixarien seients buits ni tiquets sense assignar i no hi haurien conflictes amb les regles, per tant la branca resultat d'utilitzar aquesta heurística seria efectivament la solució òptima. S'hauria trobat la solució òptima sense fer cap tipus de cerca en l'espai de solucions.

A la pràctica, la solució ideal no es donarà gairebé mai. Per aquesta raó l'heurística d'assignar sempre la millor ordre al millor seient pot no donar la solució òptima. Tanmateix, aquesta estratègia sí que ens assegura que en situacions normals la solució trobada és bastant propera a la òptima.

Un cop s'hagi trobat aquesta primera solució s'iniciarà el procés de "bactracking". Aquest procés tindrà la missió de millorar el fitness de la primera solució trobada amb la tècnica anterior. Tot seguit s'explicarà amb una mica més de detall els dos processos que conformaran l'algorisme d'assignació d'una categoria.

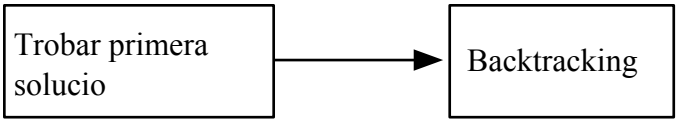


Figura 3.10 Esquema general de l'algorisme d'assignació d'una categoria

3.3.5 Trobar la primera solució

Com s'ha dit, l'estratègia és assignar les ordres de rànquing més alt als seients amb el rànquing també més elevat. Aquesta estratègia farà servir el concepte de region growing per realitzar l'assignació dels tiquets de cada ordre als seients.

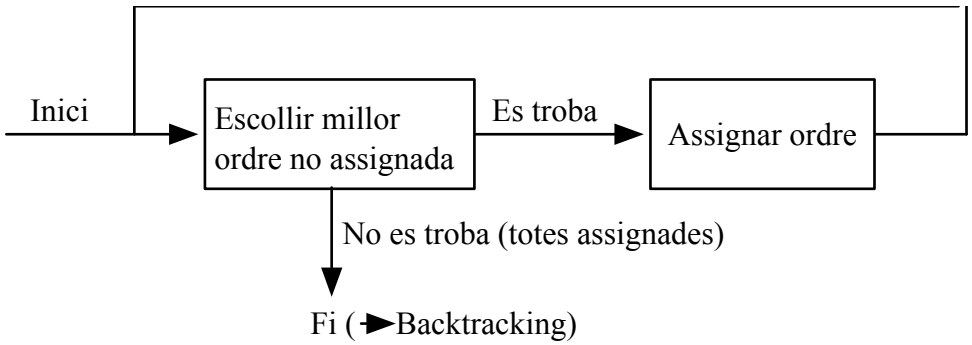


Figura 3.11 Esquema general del bloc per trobar la primera solució

El bloc d'escollir millor ordre no assignada simplement seleccionarà l'ordre de rànquing més alt que no estigui assignada. Aquesta ordre es passarà al procés d' "assignar ordre". Aquest és un procés iteratiu que podem dividir en dues fases que s'hauran de fer per assignar cada ordre seguint la filosofia del region growing.

1. Escollir una llavor per la ordre
2. Fer créixer la llavor fins que s'hagin assignat tots els tiquets de l'ordre

En la primera fase on s'escull una llavor per l'ordre s'ha de buscar primer un bloc amb suficients seients lliures i seguidament retornar d'aquest bloc el millor seient que s'utilitzarà com a llavor de l'assignació.

Un problema important és que no es pot assegurar que la llavor escollida, quan es faci créixer acabi produint una assignació correcta. Per una banda això significa que s'haurà de buscar una altra llavor per a assignar aquesta ordre Però per l'altra i molt més greu això vol dir que les següents ordres que s'escullin per assignar-se, si són semblants a aquesta també triaran aquesta llavor, s'intentaran assignar i tornaran a produir el mateix error. Considerant el gran nombre d'ordres que s'han d'assignar aquestes recaigudes insistents i innecessàries són molt costoses. Per tant es fa necessari algun tipus de control per tal que això no passi.

El control ens ha d'evitar que s'escullin repetidament llavors no vàlides per assignar les ordres. Però en general saber si una llavor produirà una bona assignació o no és inclús més costós que realitzar l'assignació i veure si un cop feta aquesta és correcta o no.

Per resoldre aquest contratemps sense haver d'esbrinar si les llavors escollides són o no vàlides es pot enfocar el procés d'assignació des d'un altre punt de vista. Fins ara dèiem que quan l'assignació no era correcta per a una ordre en una llavor determinada, es buscava una altra llavor per intentar d'assignar-la-hi. En comptes d'això ara el que farem serà mantenir la llavor i buscar una altra ordre que es pugui assignar en aquesta mateixa llavor.

L'avantatja d'aquest nou enfocament és que hi ha moltes menys ordres que llavors (cada seient de la categoria pot ser una llavor). En el cas que per a una llavor no es trobi cap ordre que s'hi pugui assignar podem prohibir la llavor per sempre, d'aquesta manera mai s'escolliran llavors incorrectes una i altra vegada per les ordres, com passava amb el plantejament inicial.

Una manera de visualitzar aquest procés és pensar que no es van distribuint les ordres als blocs sinó que es van omplint els blocs amb les ordres.

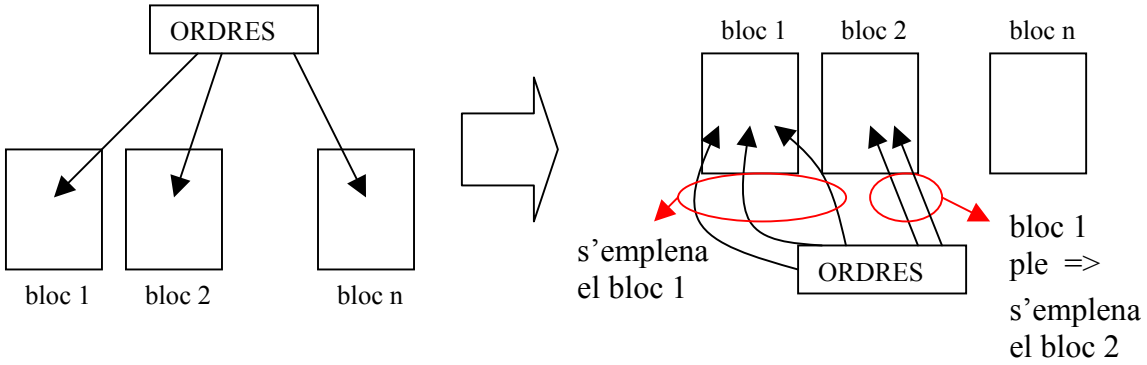


Figura 3.12 Canvi d'enfocament d'assignar ordres a emplenar blocs

S'escull el millor bloc i s'intenta emplenar amb les millors ordres. En el moment que s'hagi emplenat o quedin seients buits en els quals no hi càpiga cap de les ordres que queden es prohibiran aquests seients lliures i es procedirà a assignar el següent bloc. Això no canvia en absolut l'estratègia escollida inicialment ja que és el mateix recórrer les ordres assignant-les als millors seients que recórrer els seients de millor a pitjor assignant-los-hi les millors ordres.

Ara que hem introduït el concepte de prohibir llavors, introduirem un altre concepte que és el de marcar ordres. Això també es farà per accelerar el procés. Quan en la llavor triada no s'hi pugui assignar una ordre determinada, en comptes d'escollir senzillament la següent ordre no assignada abans el que farem serà marcar les ordres semblants a aquesta i escollir la següent ordre no marcada. Per exemple si l'assignació no s'ha pogut efectuar per què la mida de l'ordre era més gran que el conjunt de seients lliures podrem marcar les ordres d'igual o major mida que aquesta. O si no s'ha pogut assignar perquè no es complia la distància amb ordres del mateix client es poden marcar totes les ordres d'aquest mateix client, ja que a totes els passarà el mateix. Totes les marques que es posin a les ordres s'hauran de desfer un cop s'assigni una ordre o es prohibeixi la llavor.

Un cop introduïts i justificats els conceptes de prohibir llavors i marcar ordres, podem veure quin és l'esquema sencer del procés de trobar la primera solució. Tot seguit s'explicarà més detalladament l'esquema.

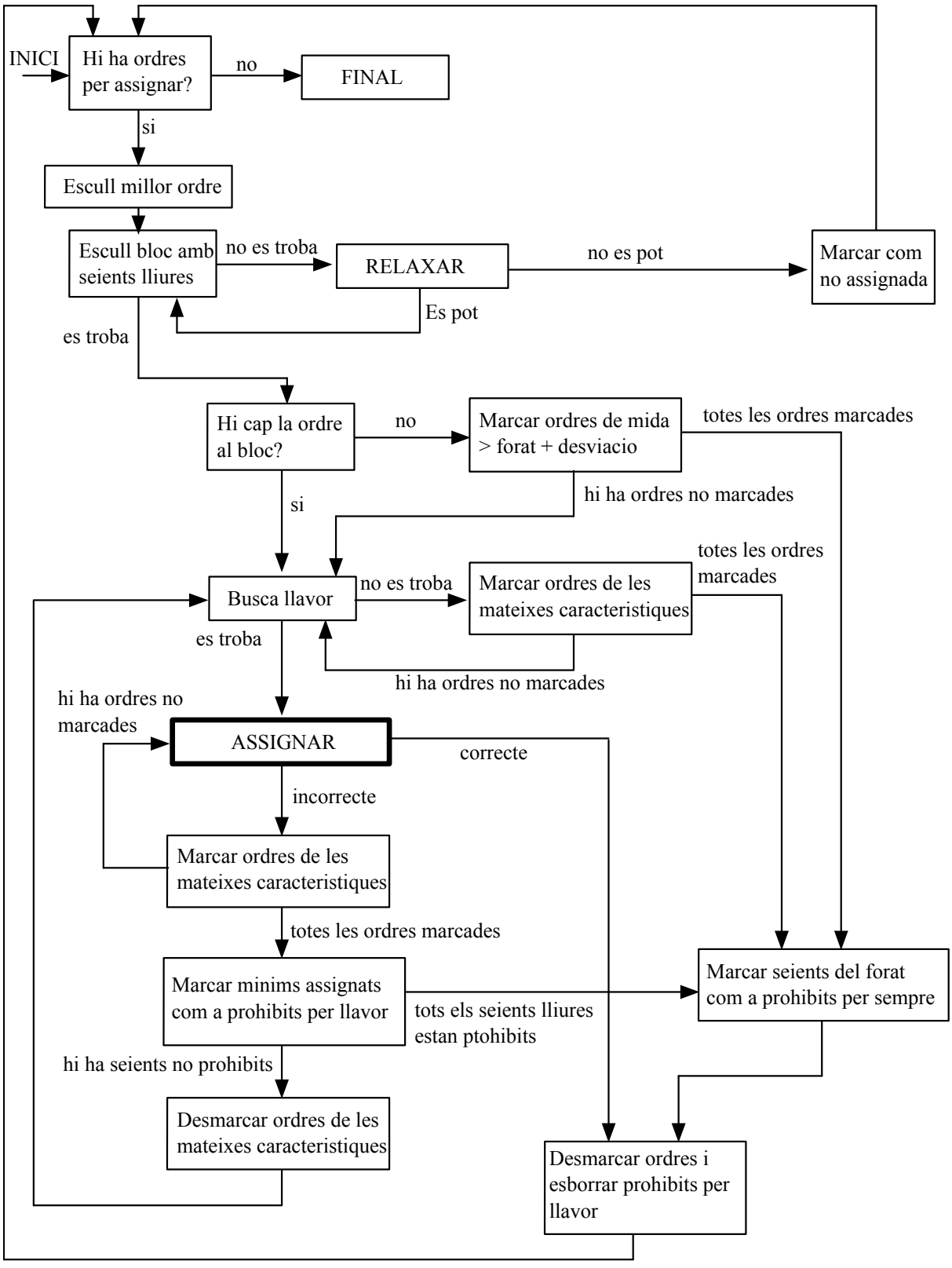


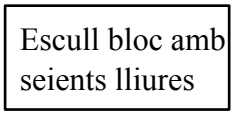
Figura 3.13 Esquema detallat de cerca de la primera solució

Com es pot veure l'esquema és iteratiu. Però no es pot dir que en cada iteració global s'assigni una ordre, que seria el més lògic. No obstant, el que assegura aquest mètode és que en cada iteració es progressarà en un d'aquests dos aspectes:

- s'assignarà una ordre
- es prohibirà almenys una llavor

Això ens assegura que no s'entrarà en bucles infinits perquè les llavors prohibides no es poden rehabilitar i les ordres assignades no es poden desassignar

3.3.5.1 Escull bloc amb seients lliures



Aquest mòdul ha de buscar i retornar el primer bloc (per ordre de rànquing) amb seients lliures no prohibits. No importa que al bloc no s'hi pugui acabar assignant l'ordre triada perquè si és així ja s'intercanviarà l'ordre, i si això no és possible es prohibiran els seients i a la següent iteració aquest bloc no es triarà.

Una altra funció que tindrà aquest mòdul serà la de dispersar les ordres per tots els blocs disponibles en els casos que la regla 13 estigui activada. Aquesta dispersió només tindrà sentit quan el nombre de tiquets venuts sigui notablement inferior als seients disponibles. En aquests casos aquest bloc haurà de repartir equitativament les ordres pels blocs de manera que tots els blocs tinguin aproximadament el mateix percentatge d'ocupació.

Per fer això a l'inici es calcularà el percentatge d'ocupació global de tota la categoria, que serà igual al total de tiquets venuts dividit pel total de seients. Aquest percentatge s'aplicarà en tots els blocs, de manera que quan un bloc determinat superi o iguali el percentatge precalculat deixarà de ser escollit per a l'assignació. D'aquesta manera s'aconseguirà que al final del procés tots els blocs tinguin un percentatge d'ocupació real aproximadament igual al fixat a l'inici.

3.3.5.2 Hi cap la ordre al bloc

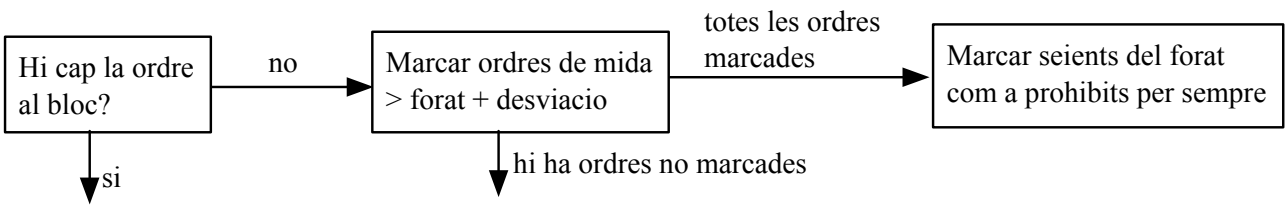


Figura 3.14 Esquema del mòdul Hi cap la ordre al bloc

Aquest bloc esbrina si la ordre hi cap al bloc escollit. Aquesta comprovació no es fa gaire meticulosament perquè sinó alentiria molt el procés, simplement mirarà si hi ha prous seients lliures i contigus com per encabir-hi la ordre en qüestió. Això es fa ràpidament perquè aquesta informació es guardarà internament als blocs justament per alleugerir aquesta verificació.

En cas que els seients lliures siguin menys que la mida de l'ordre es marcaran totes les ordres que són impossibles d'assignar en aquest bloc, que seran les de mida més gran que el nombre de seients lliures del bloc. A les ordres prou petites com per no estar marcades se'ls buscarà llavor i s'intentarà assignar-les. Però si no hi ha cap ordre prou petita (totes les ordres estan marcades) es prohibiran tots els seients lliures d'aquest bloc.

3.3.5.3 Busca llavor

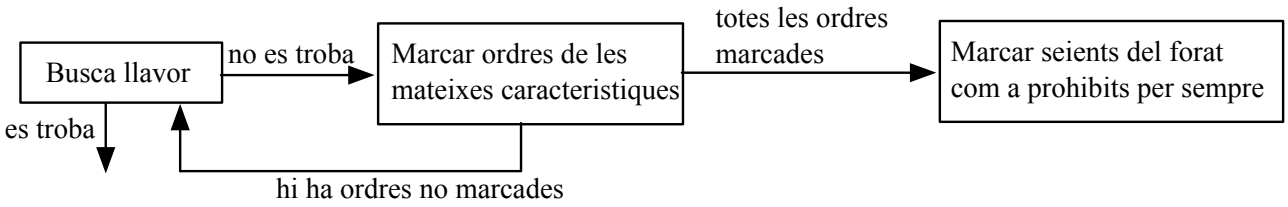


Figura 3.15 Esquema del mòdul Busca llavor

El mòdul de buscar llavor és una mica més complexe que els anteriors. La seva missió és trobar una llavor per a l'ordre. L'elecció de la llavor es farà normalment per rànquing. Com que les ordres van arribant per ordre de rànquing (de millor a pitjor), s'escollirà com a llavor per a l'ordre d'entrada el millor seient disponible de tots els del bloc que no estigui prohibit. Però si no en troba ha d'informar del motiu pel que no s'ha pogut trobat. Els motius poden ser aquests dos:

- Els seients lliures del bloc no compleixen amb la regla de la distància mínima entre ordres del mateix client. Dit d'una altra manera, en el bloc hi ha assignades una o més ordres del mateix client que la ordre que volem assignar i si la assignéssim a algun dels seients lliures del bloc es trobaria massa a prop (segons dicta la regla) d'aquestes ordres assignades.
- Els seients lliures del bloc no són del mateix tipus (atributs type, price type, sector i status) que els de la ordre.

Si no s'ha pogut trobar cap llavor el procediment serà anàleg al de bloc anterior: Es marcaran les ordres de les mateixes característiques que la d'entrada. Si l'error ha estat per distàncies amb ordres del mateix client es marcaran les ordres del mateix client, i si ha estat per error de tipus es marcaran totes les ordres del mateix tipus. Tot seguit, si hi ha ordres no marcades se'ls buscarà llavor i si no n'hi ha es prohibiran els seients.

Un aspecte interessant a tenir en compte és que les marques que s'hagin pogut posar en el bloc anterior es conserven en aquest. I en el cas que en aquest bloc s'hi produeixin vàries iteracions les marques no es reemplaçaran sinó que s'afegiran. Per tant el nombre d'ordres marcades sempre s'incrementarà. Això impedirà entrar en bucles infinits i evitarà escollir ordres que no són assignables.

3.3.5.3.1 Dispersió

L'elecció de la llavor s'ha dit que es farà *normalment* per rànquing. Un cas en que no es farà per rànquing serà quan estigui activada la regla de dispersió d'ordres (regla 13). Quan s'ha de

fer dispersió les ordres han d'estar repartides equitativament per tots els blocs (això ja ho fa el mòdul "escull bloc amb seients lliures") i uniformement dins de cada bloc.

Per distribuir uniformement les ordres dins el bloc, podem pensar en un mètode que distribueixi el més uniformement possible les llavors amb alguna fórmula matemàtica, per exemple triar la llavor més llunyana a tots els seients ocupats. Però això té varis inconvenients. Un és que calcular aquesta distància requereix un temps curt, però que s'ha de repetir molt sovint tenint en compte que aquesta funció es crida per cada llavor. L'altre inconvenient d'utilitzar fórmules és que els resultats són completament deterministes, produint formes molt similars en tots els blocs, que realment maximitzarien el fitness, però vist des de fora tots els blocs tindrien les ordres distribuïdes de la mateixa manera tal com es veu a la figura; i aquest no és l'objectiu autèntic:

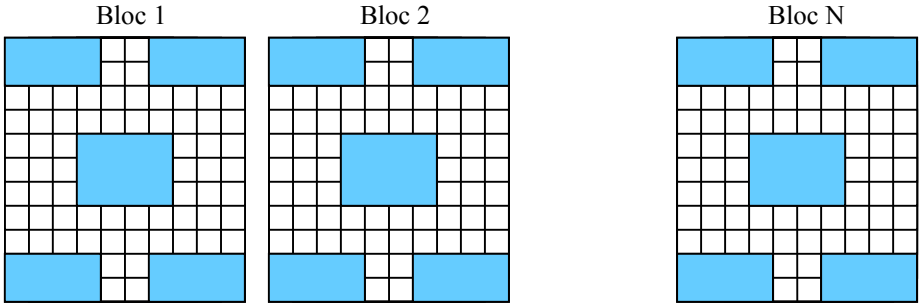


Figura 3.16 Exemple de dispersió d'ordres per fórmula matemàtica

El que proposem és repartir les llavors aleatòriament pel bloc. De fet, una funció aleatòria és perfectament dispersa i uniforme. D'aquesta manera s'obtindran resultats com els de la següent figura, que des de fora donen més la sensació de dispersió i uniformitat.

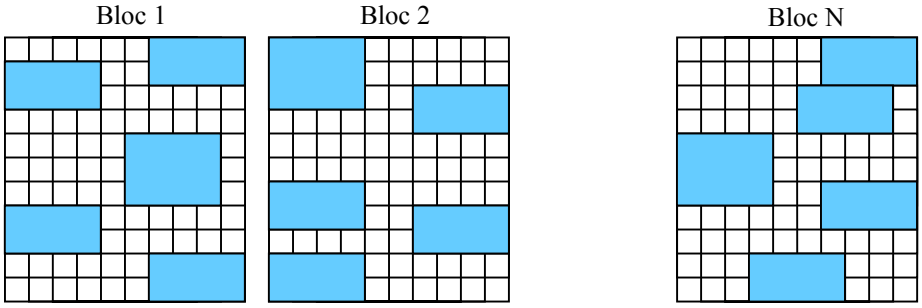


Figura 3.17 Exemple de dispersió triant la llavor aleatòriament

3.3.5.4 Assignar

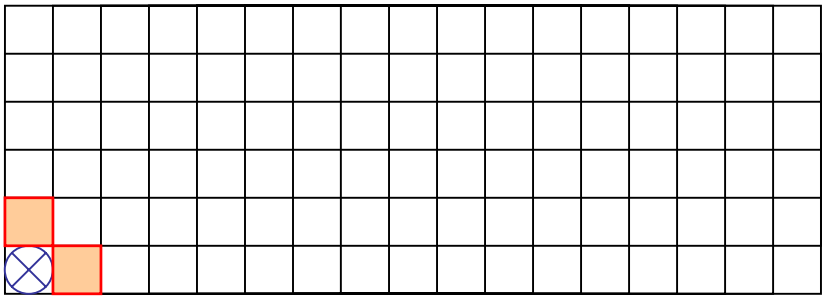
Ens centrarem primer només en el bloc d'assignar marcat amb un marc més gruixut. Posteriorment es comentaran els mòduls que formen el context d'aquest mòdul.

Com s'ha dit s'utilitzarà un algorisme de *region growing* per assignar l'ordre a la llavor. Aquest creixement haurà de complir amb les regles i restriccions. En cada pas del creixement

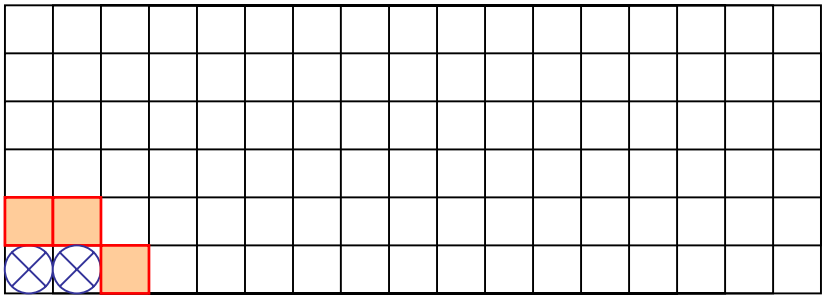
s'assignarà un seient. En el primer pas se n'assignarà un dels que són veïns de la llavor, en el següent se n'assignarà un dels que són veïns dels dos (la llavor i el primer assignat) i així successivament fins que s'hagin assignat tots els tiquets de l'ordre. Per tant a cada pas del creixement tindrem una llista de candidats dins la qual haurem d'escollir-ne el millor per ser assignat, i finalment s'actualitzarà la llista de candidats pel següent pas. Per escollir el millor es tindran en compte l'agrupació de l'ordre i el rànquing dels seients (tal com indiquen les regles).

En aquest exemple podem veure un creixement d'una llavor. Els seients assignats es marquen amb una creu  i els candidats es ressalten  .

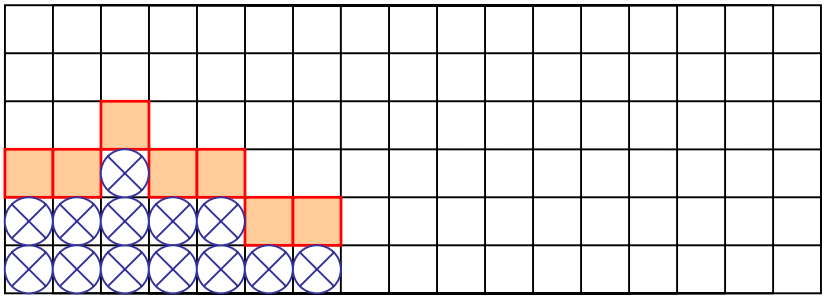
Inicialment només es considera assignada la llavor i els candidats a ser assignats són els veïns de la llavor:



En el segon pas s'assigna el millor dels candidats i s'actualitzen els candidats:



Pas n:



En el pas n podem veure que falta un candidat que és el que es troba a la dreta del seient assignat de més a la dreta. Això és perquè a cada pas s'eliminaran de la llista de candidats els que no compleixin amb les regles. En aquest exemple el màxim de seients en una fila és 7, per tant el seient que es troba a la posició 8 de la fila deixa de ser un candidat vàlid.

En cada pas s'eliminaran de la llista de candidats els que no compleixin amb les regles obligatòries 2, 3, 4 i 5 (la regla 1 es complirà implícitament perquè s'ha separat el procés en subprocessos per cada categoria), és a dir que s'eliminaran els candidats amb tipus diferents als de l'ordre (type, price type i sector). També s'eliminaran els que no compleixin amb la regla 8, que són els que es passen del màxim de seients contigus d'una ordre en una fila.

La resta de regles que s'han de fer complir en l'assignació i que fins ara no s'han tingut en compte són la 6 (el paràmetre desviació dels TOS), 8 (el paràmetre residu), 9 i 12.

La regla 6 conté un paràmetre desviació que fins ara no s'ha utilitzat. Durant l'assignació podem aprofitar-lo per modificar la mida de l'ordre actual en els casos que calgui. Per exemple si el procés de creixement de la llavor arriba a un punt on no pot avançar, podem intentar col·locar els tiquets de l'ordre que resten per assignar a una altra ordre provinent del mateix TOG, sempre que ambdues compleixin les mides establertes per aquesta regla. Un altre cas on podríem aprofitar la desviació seria quan un cop assignats tots els tiquets de l'ordre, quedessin en el bloc pocs seients lliures. Inversament al cas anterior, aquí agafaríem tiquets provinents d'una altra ordre del mateix TOG per ocupar aquests seients.

La regla 8 té dues parts. La primera és el màxim de seients per fila que ja s'ha introduït. La segona part fa referència al residu d'una fila. Aquest residu es defineix amb el sentit de que una fila pot tenir tants seients com marqui el màxim de seients per fila, però si és necessari a aquest màxim se li pot sumar el residu. Farem servir el residu quan en una fila quedin un nombre de seients lliures inferior o igual al residu. Per exemple en aquest cas en què el màxim de seients per fila és 10 i el residu és 2 l'assignació d'una ordre de 46 tiquets serà aquesta:

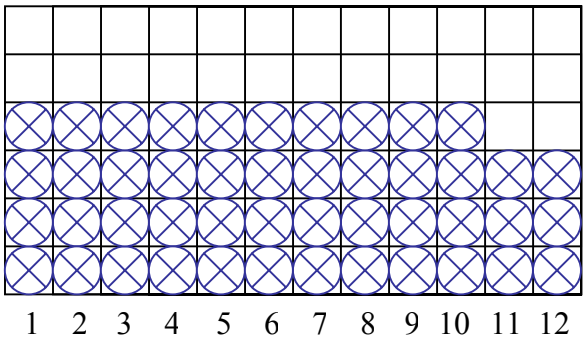


Figura 3.18 Exemple d'assignació d'una ordre utilitzant el residu

Com es veu es fa servir el residu per 'completar' la fila. La fila de dalt de tot no s'ha completat perquè la ordre és de 46 tiquets.

La regla 12 realment són dues regles. La primera si està activada no permet que dues ordres del mateix client s'assignin en un mateix bloc; aquesta la tindrà en compte el mòdul de buscar llavor. En el mòdul d'assignar només hi afecta la segona regla que si està activada, fixa una distància mínima (en seients) entre les ordres del mateix client. Aquesta regla es podria aplicar en cada pas del region growing eliminant els candidats massa propers. Però calcular aquestes distàncies per cada nou candidat seria força lent. Una manera d'accelerar-ho és marcar abans de començar el region growing, tots els seients del bloc que no compleixen aquesta regla. Els candidats del region growing només es triaran dels seients que no estiguin marcats.

3.3.5.4.1 No deixar mai un tiquet en una fila sol

La regla 9 es digna d'un apartat propi. Aquesta regla obliga a no deixar mai un tiquet sol en una fila (per a ordres de més d'un tiquet). Això realment implica dues situacions. Una és pròpiament no deixar un tiquet d'una ordre en una fila sol, i l'altra menys intuïtiva és no deixar un sol seient lliure en una fila; ja que deixant-lo s'obliga o bé a que una altra ordre l'ocupi deixant un dels seus tiquets sols, o bé que aquest tiquet no s'ocupi mai.

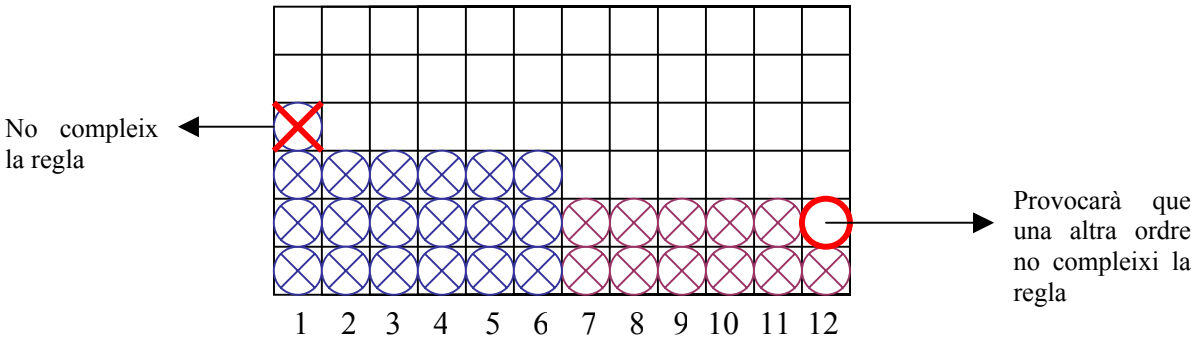


Figura 3.19 Implicacions de la regla de no deixar tiquets sols en una fila

Aquesta regla no es pot fer complir en cada pas del region growing perquè en molts passos del creixement hi haurà tiquets sols i seients buits en una fila. Per tant aquesta regla s'ha d'aplicar al finalitzar aquest mètode.

Els casos en què quedin tiquets sols en una fila es poden resoldre de diferents maneres. Per exemple si algunes de les files assignades amb tiquets de l'ordre tenen seients lliures a les bandes es pot moure el tiquet sol a una d'aquestes files sempre que el total de seients ocupats a la fila sigui inferior al màxim més el residu (tornem a aprofitar aquí el residu de la regla 8).

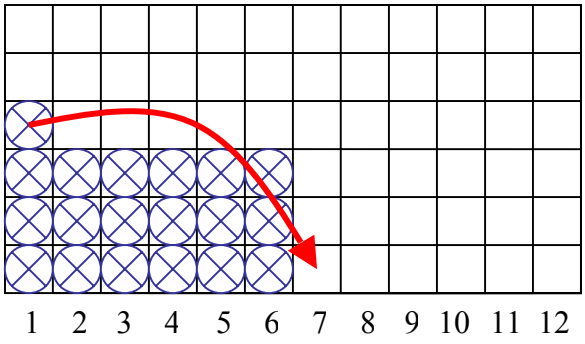


Figura 3.20 Exemple de resolució d'un tiquet sol movent-lo a una altra fila

En els casos que això no sigui possible el que es pot fer és moure algun dels tiquets assignats al costat del que es troba sol. De fet se n'hauran de moure dos al seu costat perquè si només en movem un aquest deixarà inevitablement un seient buit que com ja s'ha explicat no és desitjable. Diem inevitablement perquè si no fos així i per exemple el tiquet tingués un seient lliure al seu costat, ens trobaríem realment en el cas de dalt i la solució seria moure el tiquet sol al seu costat.

Per tant s'hauran de moure dos tiquets al costat del que ha quedat sol. Els dos que s'escullin hauran de ser contigus per no deixar seients buits sols.

Figura 3.21 Exemple de resolució d'un tiquet sol movent-ne dos al seu costat

Tot i així encara hi haurà casos on no es podrà aplicar cap de les solucions anteriors. En aquestes situacions es considerarà que l'assignació de l'ordre és incorrecta, ja que aquesta regla quan està activada s'ha de complir irrevocablement.

3.3.5.4.2 Context de l'algorisme assignar

Un cop vist l'algorisme intern d'assignació anem a veure el bloc d'assignar més globalment amb el seu context:

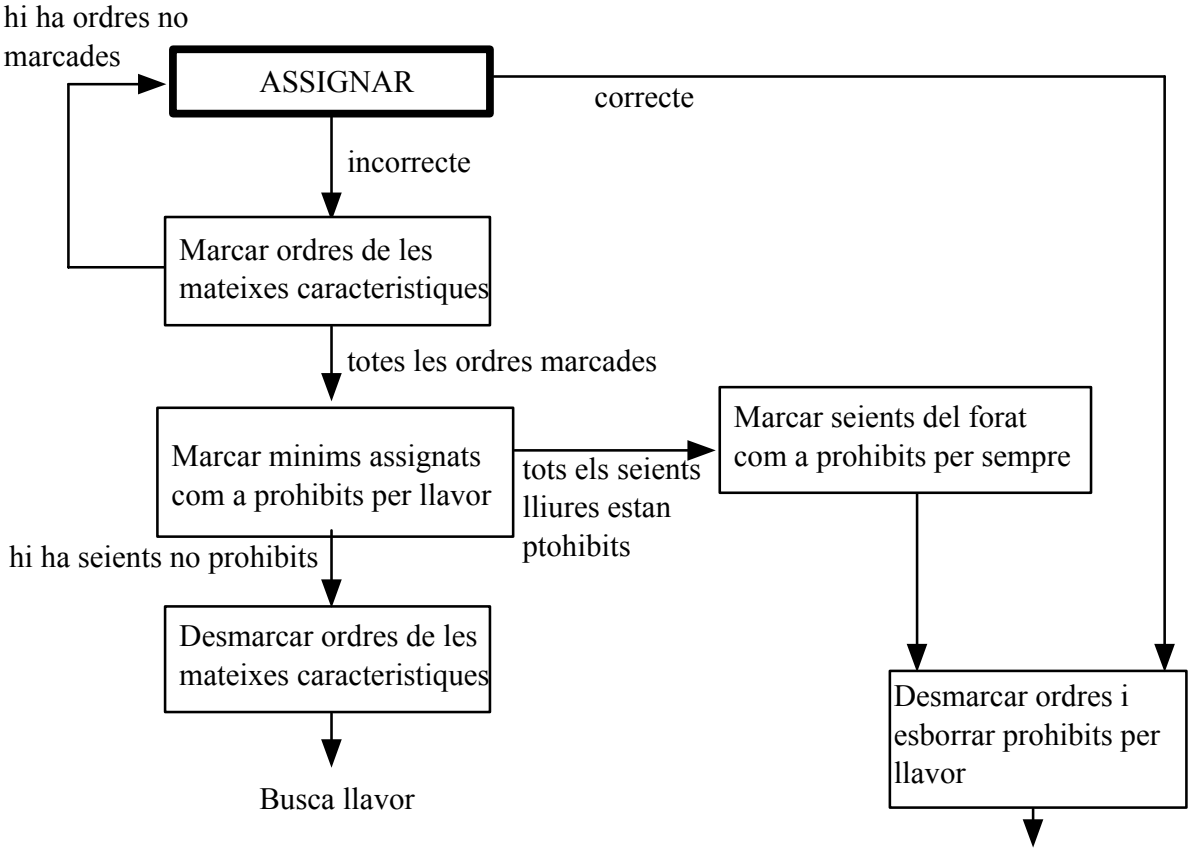


Figura 3.22 Context del mòdul assignar

El procediment per quan el resultat de l'assignació no és correcte és similar al de "buscar llavor" i també al de "hi cap la ordre al bloc". Es marcaran totes les ordres de les mateixes característiques que la que ha provocat la fallada i se n'escollirà una de les no marcades per intentar assignar-la amb la mateixa llavor. Si s'arriba a una situació en què totes estiguin marcades, en els casos d'abans es prohibien per sempre tots els seients del forat, però fer això també aquí no seria del tot correcte. Que l'assignació de l'ordre en aquesta llavor no hagi sortit correcte no vol dir que en una altra llavor del mateix bloc sí qui surti bé, per exemple una ordre de 10 tiquets assignada en una llavor tal com es veu a la figura, produeix una assignació incorrecte degut a que queda un tiquet sol en una fila que no es pot resoldre amb cap dels mètodes anteriorment exposats, en canvi si assignem la mateixa ordre en una altra llavor del mateix bloc l'assignació es pot realitzar obtenint un resultat correcte:

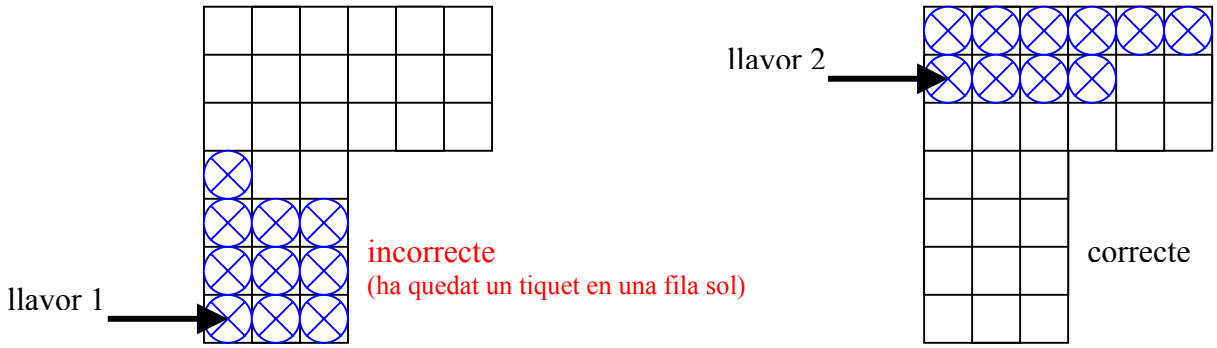


Figura 3.23 Exemple de resultats d'assignació diferents en llavors diferents del mateix bloc

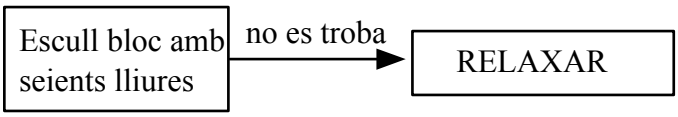
Per tant queda demostrat que no podem prohibir per sempre tots els seients del bloc com fèiem en els altres mòduls. Per resoldre aquest fet podem aprofitar una propietat de l'algorisme del region growing. Una assignació d'una ordre en una llavor, anirà creixent fins a ocupar un conjunt de seients (tants com tiquets tingui l'ordre). La peculiaritat és que el conjunt de seients ocupats seria el mateix si la llavor s'hagués plantat en qualsevol d'aquests seients ocupats.

Podem fer servir aquesta propietat per crear un nou tipus de seients prohibits, que anomenarem prohibits per llavor. Els prohibits per llavor com el seu nom indica, són seients que no podrem triar com a llavor perquè sabem a priori que provocaran una assignació incorrecte.

Quan s'assigni una ordre en una llavor i el resultat sigui incorrecte i no es pugui intercanviar l'ordre per cap altra, es marcaran tots els seients ocupats en aquesta assignació com a prohibits per llavor. El bloc "busca llavor" haurà de tenir en compte aquests seients per obviar-los. D'aquesta manera es triarà una llavor que es trobarà fora de la zona que sabem que resultarà en una assignació incorrecte. En el moment que tots els seients del bloc estiguin prohibits per llavor, tal com es veu a l'esquema, es marcaran aquests ara sí com a prohibits per sempre.

L'últim cas que falta per explicar és quan l'assignació surt incorrecte, es realitza un o més intercanvis d'ordres però no s'aconsegueix cap assignació correcte. En aquest cas els seients a prohibir per llavor han de ser els que hagin estat ocupats conjuntament per totes les ordres provades. Els seients d'aquesta zona comuna són els que s'anomenen "mínims assignats" en l'esquema.

3.3.5.5 Relaxar



Com es pot veure en l'esquema general ens podem trobar amb un cas excepcional on no es trobi cap bloc amb suficients seients lliures per a l'ordre d'entrada. Això podria ser per exemple quan volguéssim assignar una ordre de 40 tiquets i tots els blocs estiguessin plens o tinguessin menys seients lliures que la mida de l'ordre. Una altre situació que podria provocar aquest fet seria que hi hagués ordres del mateix client en tots els blocs i en cap d'ells es pogués trobar un seient que complís la distància mínima amb ells.

El bloc de relaxar haurà d'intentar solucionar aquest fet. Un intent de solucionar la primera situació esmentada al paràgraf anterior pot ser per exemple fer servir la regla de partir TOS (regla 10) en cas que estigui activada. Així reduint la mida de l'ordre a com a mínim la mida mínima del TOS especificada és més fàcil que es pugui trobar alguna llavor. Si la regla no està activada o no és possible trobar cap llavor ni per la mida mínima de l'ordre, una altra possibilitat que tenim és realitzar relaxacions de regles en cas que s'hagi activat aquesta opció. Per la segona situació la solució també podria ser per exemple realitzar una relaxació de la regla de la distància mínima.

L'esquema del bloc relaxar quedaria així:

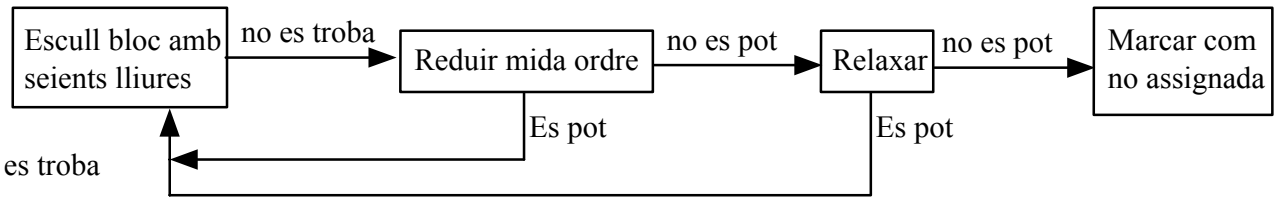


Figura 3.24 Esquema del bloc relaxar

Podem veure com quan no es troba llavor s'inicia una mena de bucle fins que s'acaba trobant la llavor. Aquest bucle comença reduint la mida de l'ordre en una certa quantitat, si segueix sense trobar-se llavor es redueix més la mida fins que s'arriba al mínim estipulat per la regla i en aquest cas s'ha de procedir a relaxar regles. Es començarà relaxant la primera, i s'aniran realitzant les relaxacions en l'ordre en què s'han entrat fins que es trobi una llavor per l'ordre.

Però com podem veure en l'esquema això no assegura de cap manera que arribem a la mateixa situació d'abans on no era possible trobar un bloc amb suficients seients lliures per l'ordre (encara que així es redueix molt la possibilitat de que es doni el cas). L'únic que podrem fer en aquest cas extrem és no assignar la ordre, marcar-la com a tal i provar d'assignar la següent. Això és el que no volen que passi els organitzadors de la FIA, ja que se suposa que mai es venen més tiquets que seients hi ha en el circuit. Per tant una assignació amb ordres no assignades serà una solució molt dolenta, i així ho reflectirà el seu valor de la funció de fitness.

Però en el procés de buscar la primera solució no hi ha més remei que deixar-ho així. Un cop acabat s'iniciarà el procés de backtracking que ja s'encarregarà de fer tot el possible per assignar aquestes ordres no assignades, ja que el seu objectiu és millorar el valor de la funció de fitness.

3.3.6 Resum de regles

En aquest apartat es fa un repàs de totes les regles i es mostra en quin/s bloc/s de l'esquema general s'apliquen.

- Regla 1

Aquesta regla obliga a assignar les ordres d'una categoria als seients de la mateixa categoria. Aquesta regla es complirà implícitament ja que com s'ha justificat abans d'iniciar el procés es filtraran i separaran les dades de cada categoria per poder executar-les en processos independents.

- Regles 2,3,4 i 5

Les regles 2, 3, 4 i 5 obliguen a les ordres que tinguin algun tipus especificat (type, price type, sector, status) a ser assignades en seients del mateix tipus. Aquest fet es té en compte en els següents mòduls:

  - Mòdul hi cap la ordre (3.3.5.2): Un dels criteris serà comptar el nombre de seients lliures del mateix tipus que l'ordre per determinar si l'ordre hi cap o no.
  - Mòdul buscar llavor (3.3.5.3): Només retornarà llavors que siguin dels mateixos tipus que l'ordre.
  - Mòdul assignar (3.3.5.4): L'algorisme del region growing eliminarà de la llista de candidats els que no siguin del tipus de l'ordre.

Un altre aspecte a tenir en compte és que pot ser que el nombre de seients d'un tipus determinat sigui major al total de tiquets venuts d'aquest tipus (no obstant, el cas invers no es pot donar). La regla contempla aquest cas, dient que els seients sobrants d'un tipus determinat podran ser assignats a tiquets que no tinguin cap tipus associat.

Per fer això l'algorisme internament emmagatzemarà una llista de tots els tipus possibles amb un comptador dels tiquets que falten per assignar de cada tipus. En el moment que no quedin més tiquets per assignar d'un tipus (comptador a zero) els seients marcats amb aquest tipus podran ser assignats. O dit d'una altra manera els mòduls esmentats a dalt només filtraran els seients quan el seu comptador respectiu sigui superior a zero.

- Regla 6

La partició dels TOG en TOS s'efectua abans de començar el procés. Durant el procés la mida de les ordres podrà modificar-se en el mòdul d'assignar fent ús del paràmetre desviació; sempre dins dels límits estipulats per aquesta regla.

- Regla 7

El rànquing es fa complir tant en el mòdul de triar llavor, que escollirà per llavor el seient de rànquing més alt, com en el mòdul d'assignar que un dels criteris per escollir el proper seient a assignar de la llista de candidats és el rànquing.

- Regla 8

La regla del màxim de seients per fila també s'aplica en el mòdul d'assignar que eliminarà de la llista de candidats els que sobrepassin aquest màxim.

- Regla 9

Al finalitzar el procés d'assignar l'ordre s'executarà el mecanisme explicat a l'apartat 3.3.5.4.1 per evitar deixar tiquets sols en una fila. En cas que aquest mecanisme no elimini tots els casos de tiquets sols, es considerarà que l'assignació és incorrecta. Per tant quan aquesta regla estigui activada es farà complir sempre.

- Regla 10

La regla de partir TOS només s'utilitza quan no es pot assignar una determinada ordre a cap bloc. Això es troba en el mòdul de relaxar (3.3.5.5).

- Regla 11

La distància entre ordres del mateix client es tindrà en compte alhora de triar la llavor i també en l'assignació, marcant prèviament els seients massa propers per no ser assignats.

- Regla 12

Aquesta regla es posa amb l'objectiu de no assignar les ordres al mig dels blocs deixant seients buits al marges. Segons el plantejament que s'ha fet de l'algorisme les ordres s'assignen seguint el rànquing.

S'assignarien les ordres al mig dels blocs si els millors seients del bloc fossin al mig del bloc, però aquest fet no es dona mai. Habitualment el rànquing o comença a la primera fila i recorre totes les files fins a la última fent esses o en zig-zag, o a la inversa. Per tant poques vegades ens trobarem amb que queden seients buits als marges de les files. Però inclús aquests pocs casos també els contempla el mòdul d'assignar, utilitzant el residu r de la regla 8 per completar sempre les files on quedin fins a r seients lliures als marges.

- Regla 13

La dispersió de les ordres es fa en tant al mòdul "buscar bloc amb seients lliures" com al "buscar llavor". El primer produirà una dispersió global per tots els blocs, intentant tenir un percentatge d'ocupació similar a tots els blocs. El segon farà la dispersió internament en cada bloc escollint les llavors aleatòriament.

3.3.7 Backtracking

L'objectiu del backtracking és modificar la solució obtinguda en el procés anterior per tal d'incrementar el valor de la funció de fitness. Podem distingir dos casos suficientment diferenciats com per aplicar-los-hi tècniques de backtracking diferents.

Un cas és quan el procés anterior de cercar la primera solució no ha pogut assignar totes les ordres. En aquest cas la prioritat serà assignar el màxim nombre d'ordres no assignades possible. El segon cas és per quan s'han pogut assignar totes o quan no es vol tenir en compte que resten ordres sense assignar. En aquest cas l'objectiu és millorar el fitness de les ordres ja assignades.

3.3.7.1 Backtracking per les ordres no assignades

La funció de fitness penalitza greument el fet que quedin ordres sense assignar. Per aquesta raó si el procés de cercar una primera solució ha deixat ordres sense assignar, la prioritat per tal de fer pujar el fitness és assignar el màxim nombre d'ordres possibles.

Com que el procés anterior no les ha assignat això vol dir que no hi ha cap bloc amb suficients seients per allotjar cap de les ordres no assignades que siguin compatibles en termes de característiques dels tiquets i en compliment de regles. Per tant la única manera d'assignar aquestes ordres és desassignant algunes de les assignades i provant combinacions d'assignacions amb totes les no assignades on s'aconsegueixi disminuir el total de tiquets no assignats.

Per exemple:

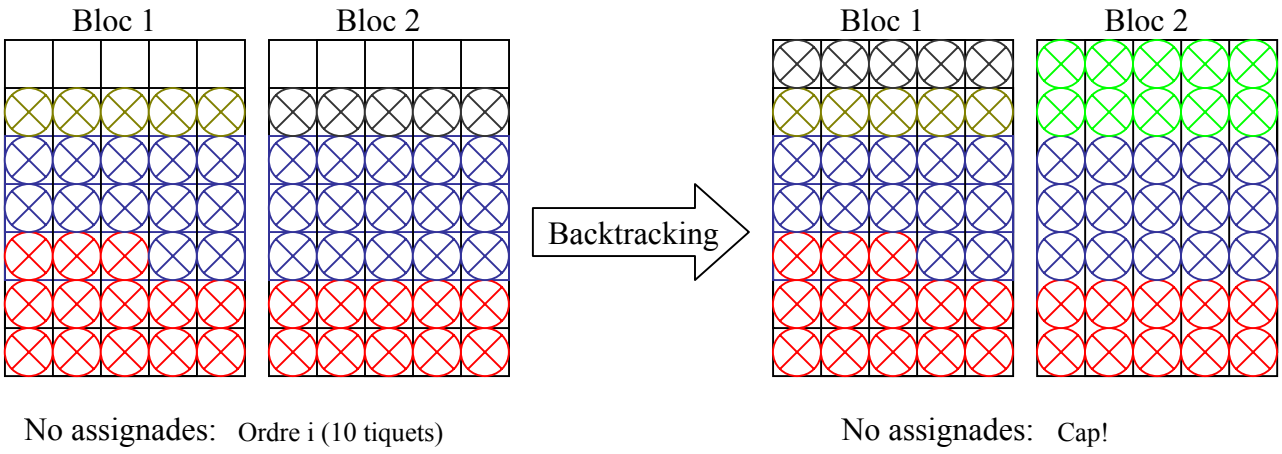


Figura 3.25 Exemple de resultat del backtracking per les ordres no assignades

No obstant, la majoria de vegades ens trobarem que si desassignem una ordre, el 'forat' (a partir d'ara anomenarem forat a un conjunt de tiquets veïns que estan lliures) que ens deixa és de la mida de l'ordre desassignada i la solució més òptima sol ser omplir aquest forat de nou amb l'ordre que s'acaba de desassignar. Per tant un mecanisme més intel·ligent pot ser

desassignar només ordres que tinguin seients lliures als voltants de manera que el forat resultant sigui d'una mida més gran al de l'ordre desassignada.

Amb aquesta estratègia de desassignació guiada serà més fàcil obtenir forats adients per alguna de les ordres no assignades i hi ha la possibilitat de que l'ordre que s'ha desassignat pugui ser assignada en un altre lloc, provocant un increment substancial del fitness.

Una altra manera de veure el backtracking per les ordres no assignades és pensar que l'objectiu és emplenar blocs. Com més blocs quedin complerts (a partir d'ara s'anomenarà bloc complert a un bloc sense cap seient lliure), més grans seran els forats que quedaran als altres blocs, augmentant així la possibilitat de poder assignar-hi les ordres no assignades.

La solució que es proposa consisteix en desassignar les ordres properes a forats; intentar assignar les ordres no assignades en aquest nou forat i la recentment desassignada als altres forats d'aquest o dels altres blocs.

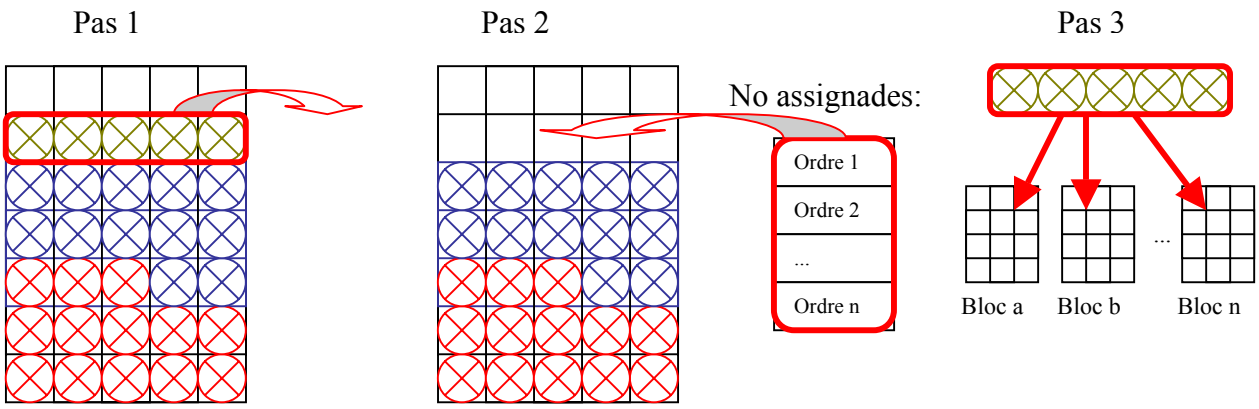


Figura 3.26 Solució proposada pel backtracking per assignar les ordres no assignades

Serà un procés iteratiu que anirà desassignant progressivament les ordres properes a cadascun dels forats de tots els blocs fins que la mida del forat sigui més gran que la mida màxima d'una ordre (en aquest cas ja no té sentit continuar desassignant). En cada desassignació s'intentarà assignar al nou forat les ordres no assignades i es provarà d'assignar l'ordre desassignada als forats dels altres blocs. Però només es considerarà que l'assignació d'una ordre és vàlida quan aquesta tapi el forat completament, ja que altrament no té sentit perquè més endavant es tornaria a desassignar aquesta ordre degut a que és al costat d'un forat, i es repetirien accions innecessariament assignant i desassignant sempre les mateixes ordres en diferents forats. Així doncs podem entendre que l'objectiu inicial d'assignar les ordres no assignades equival a aquest nou objectiu de completar blocs.

Al final d'aquest procés s'haurà aconseguit completar alguns blocs, tot i que possiblement tindrem més ordres no assignades que al principi degut a que només acceptem assignacions d'ordres que tapin els forats. Per això al final s'assignaran les ordres no assignades 'normalment' (sense haver de tapar obligatòriament els forats), deixant nous forats als blocs per a la següent iteració.

La idea d'aquest backtracking és que en cada iteració s'incrementi el número de blocs complerts, augmentant les mides dels forats dels altres blocs amb el conseqüent augment de probabilitats que als forats s'hi puguin assignar les ordres no assignades.

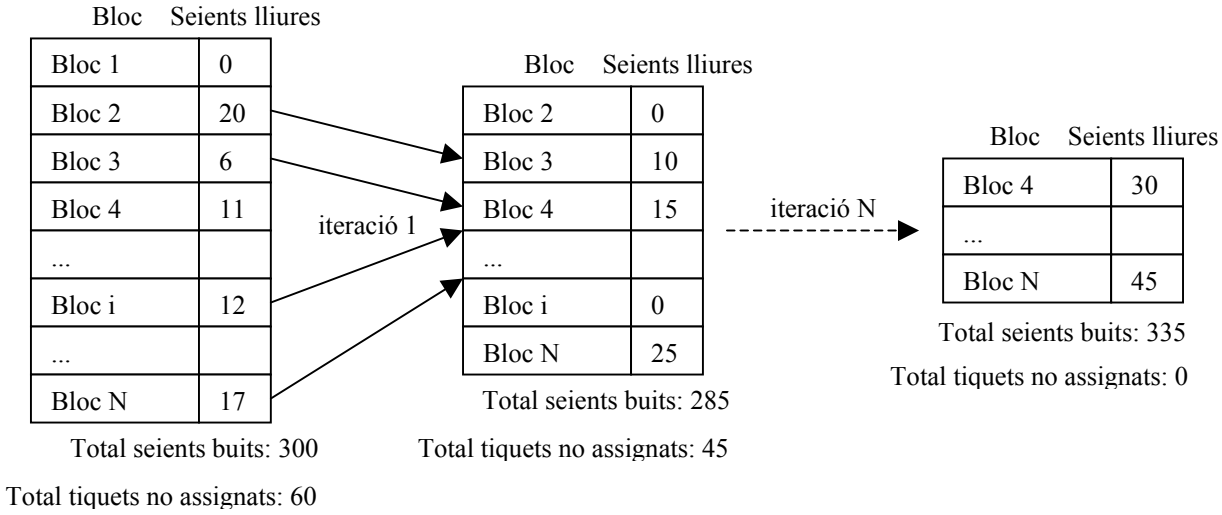


Figura 3.27 Exemple de l'evolució dels blocs en el backtracking per les ordres no assignades

Com es veu en la figura en cada iteració només es modifiquen els blocs no complerts (amb alguns seients lliures) i la idea és que a cada iteració hi hagi més blocs complerts i per tants menys blocs a tractar. Tot i que també es posarà un control perquè si en un número determinat d'iteracions no s'aconsegueix completar cap bloc s'aturi el procés.

3.3.7.2 Backtracking per millorar les ordres assignades

Aquest backtracking es realitza bloc a bloc amb l'objectiu de millorar el fitness del bloc, definit com BF a la funció de fitness. Com es pot veure en la funció de fitness, el BF depèn dels fitness de les ordres assignades en el bloc, i de la dispersió d'aquestes ordres pel bloc en el cas que la regla de dispersar es trobi activada.

Podem pensar en dues maneres diferents de millorar el fitness del bloc depenent de si s'ha de fer dispersió o no.

3.3.7.2.1 Sense dispersió

En cas que no s'hagi de dispersar, el fitness del bloc només depèn dels rànquings i de l'agrupació de les ordres. En un sol bloc el tema dels rànquings no afectarà gaire a la funció ja que com que aquesta mesura es fa a partir dels rànquings globals, en un bloc concret els diferents rànquings globals dels seus seients seran molt propers, per tant intentar millorar els fitness dels rànquings de les ordres (TOSR) produirà un increment inapreciable en el valor total del fitness.

Per tant la millora del fitness la causarà únicament l'agrupació de les ordres. L'objectiu serà que totes les ordres estiguin el màxim d'agrupades possible en el bloc tot complint les regles.

Però no és gens fàcil ni tan sols per un sol bloc i tenint en compte un únic criteri (l'agrupació) aconseguir maximitzar la funció de fitness.

El problema és que el fet d'assignar una ordre (la primera per exemple) en un bloc amb la millor agrupació, pot no donar el millor valor de fitness total per al bloc, ja que aquesta millor agrupació per aquesta ordre pot entorpir les assignacions de les altres ordres empitjorant les seves agrupacions, i fins i tot es pot donar el cas que al final no hi càpiguen al bloc totes les ordres que hi havien originalment quan les assignem amb un altre ordre, en unes altres posicions i amb unes altres formes.

Pot semblar que per un sol bloc amb poques ordres es podria intentar fer una cerca exhaustiva de totes les possibilitats d'assignació però ni tan sols aquest cas és tractable. L'espai de cerca per a un bloc de mida estàndard, amb 10 ordres de mida 40 seria 10! = 3.628.800. Aquest número és molt gran, si triguéssim només una dècima de segon per crear i avaluar cada possibilitat trigaríem unes 2 hores per explorar-lo tot; però si en comptes de 10 ordres fossin 11 o 12 les possibilitats es disparen (multiplicant les 2 hores per 10 o per 100). Sembla clar que tampoc podem pensar en un mètode que trobi la millor assignació possible en un bloc i menys encara sabent que s'ha de repetir aquest procés per tots els blocs del circuit.

La solució proposada és provar amb un número petit i fixe de combinacions diferents i quedar-nos amb la que doni el valor més alt de fitness. Per exemple que el nombre de provatures sigui 5 vegades el nombre d'ordres del bloc. En el cas d'abans faríem només 50 provatures (5 * 10). Per aconseguir que les provatures siguin diferents entre elles ho farem assignant en cada provatura les ordres en un ordre aleatori. D'aquesta manera òbviament no es pot assegurar que es trobarà la millor distribució en el bloc, però és una solució prou bona perquè realitza una cerca àmplia en l'arbre de cerca (àmplia en el sentit de que es fa per qualsevol zona de l'arbre sense parcialitat). Com que les branques que es trien són a l'atzar, trobarem casos substancialment diferents entre ells que és el que més interessa en aquesta situació.

3.3.7.2.2 Amb dispersió

En el cas que s'hagi de dispersar, la situació és ben diferent. En aquest cas els blocs no estan plens i les ordres es reparteixen en llavors aleatòries com s'ha dit. Per incrementar el valor de la funció de fitness aquí, hi entra un altre paràmetre de la funció que juga un paper molt important que és la dispersió. Si volem millorar el fitness del bloc el millor que podem fer és dispersar millor aquestes ordres, ja que el rànquing igual com abans gairebé no afectarà i l'agrupació com que el bloc no s'emplena en principi serà màxima com més ben dispersades estiguin les ordres.

El que proposem anàlogament a la solució escollida al mòdul "busca llavor", és distribuir les ordres en posicions aleatòries del bloc, però igual com en el cas d'abans fer un cert nombre de provatures i quedar-nos amb la que doni el valor més alt en la funció de fitness.

3.4 Funció de fitness

3.4.1 Introducció

La funció de fitness ens serveix per mesurar el grau de bondat d'una solució. També es fa servir per decidir quan un resultat d'una assignació és millor que un altre. Aquesta funció serà molt útil per millorar en el backtracking el primer resultat d'assignació.

3.4.2 Criteris

La funció de fitness té en compte els següents criteris:

- Rànquing del seient i de l'ordre
- Relaxacions/desactivacions produides durant l'assignació
- Fragmentacions de les ordres (partir TOS)
- Dispersió de les ordres (en cas d'assignació amb dispersió)
- Agrupació de les assignacions de les ordres
- Tiquets no assignats.

3.4.3 Definicions

Considerem que tenim els seients ordenats per un rànquing global de tot el circuit (SR):

Block	Block Rank	Row	Seat	Seat Rank	SR
1	1	1	1	1	1
1	1	1	2	2	2
1	1	2	1	3	3
1	1	2	2	4	4
2	2	1	1	1	5
2	2	1	2	2	6

Considerem que tenim els tiquets ordenats amb un rànquing global de tots els tiquets (TR). Aquest rànquing global es genera ordenant els tiquets de les ordres per l'ordre de rànquing de les ordres:

Order Id	Seats	Order Rank
1	3	1
2	5	1
3	3	2
4	3	3
5	2	7

Ordres originals

Order Id	Tiquet	Ticket Rank	TR
1	1	1	1
1	2	1	2
1	3	1	3
2	1	1	4
2	2	1	5
2	3	1	6
2	4	1	7
2	5	1	8
3	1	2	9
3	2	2	10
3	3	2	11

3.4.4 Funcions

Funció 1: Valor màxim de rànquing global tant a nivell de tiquet com de seient

És el rànquing més alt assignat a un seient o a un tiquet. Per lògica, com que hi hauran igual o menys nombre de tiquets que de seients, acostumarà a ser el rànquing més alt associat a un seient.

$$MaxGR = Max(Max(SR), Max(TR))$$ **Maximum Global Ranking**

On:

Max(SR) = el rànquing global més alt assignat a un seient
Max(TR) = el rànquing global més alt assignat a un tiquet

Funció 2: Fitness d'un tiquet individual

Adequació del tiquet al seient assignat des del punt de vista del rànquing. Comparem els rànquings globals del tiquet i del seient normalitzat respecte el Maximum Global Ranquing.

$$ITF_i = 100 - \left| \frac{TR_i * 100}{MaxGR} - \frac{SR_j * 100}{MaxGR} \right| \qquad 0 \le ITF_i \le 100$$ **Ticket Fitness**

On:

TR_i = rànquing global del tiquet *i*
SR_j = rànquing global del seient *j*

Funció 3: Fitness del rànquing d'un TOS

Adequació dels tiquets d'un TOS als seients assignats des del punt de vista del rànquing. Calculem la mitjana dels ITF de tots els tiquets que pertanyen al TOS.

$$TOSR_o = \frac{\sum_{i=1}^{nto_o} ITF_i}{nto_o} \qquad 0 \le TOSR_o \le 100$$ **TOS Ranking Fitness**

On:

nto_o = nº de tiquets del TOS *o*

Funció 4: Fitness de l'agrupació d'un TOS

Calculem el fitness de l'agrupació d'un Ticket Order Subgroup comparant l'agrupació real del TOS amb l'agrupació ideal. L'agrupació real la calculem tinguent en compte el nombre de files i columnes ocupades pel TOS. L'agrupació ideal és la que minimitza la funció.

$$TOSJ_o = 100 \cdot \frac{N'_o \cdot Y + M'_o \cdot X}{N_o \cdot Y + M_o \cdot X} \qquad 0 \le TOSJ_o \le 100$$ **TOS Joint Fitness**

$$M'_o = \sqrt{\frac{nto_o \cdot Y}{X}} \qquad N'_o = \frac{nto_o}{M'_o}$$

NOTA: Si el TOS_o està partit i els trossos estan en blocs diferents, $TOSJ_o$ serà igual a 0.

On:

N_o = nombre de files diferents que ocupa l'ordre *o*
N'_o = nombre de files ideal per tenir una agrupació màxima en l'ordre *o*
M_o = nombre de columnes diferents que ocupa l'ordre *o*
M'_o = nombre de columnes ideal per tenir una agrupació màxima en l'ordre *o*
Y = constant que indica la distància entre dos seients sobreposats
X = constant que indica la distància entre dos seients de costat
nto_o = nº de tiquets del TOS *o*

Funció 5: Fitness d'un TOS

Calculem el fitness d'un Ticket Order Subgroup ponderant el fitness del rànquing amb el fitness de l'agrupació i dividint-ho pels pesos de les relaxacions/desactivacions. Així doncs, la ponderació de fitness es veurà reduïda quan s'hagin hagut de relaxar/desactivar regles per poder assignar.

$$TOSF_o = \frac{p_{TOSR} \cdot TOSR_o + p_{TOSJ} \cdot TOSJ_o}{(p_{TOSR} + p_{TOSJ}) \cdot \left(1 + \sum_{r=1}^{nr_o} q_r\right)} \qquad 0 \le TOSF_o \le 100$$ **TOS Fitness**

On:

p_{TOSR} = pes del TOS Ranking Fitness
p_{TOSJ} = pes del TOS Joint Fitness
nr_o = nombre de regles relaxades/desactivades per assignar el TOS *o*
q_r = importància de la regla relaxada/desactivada *r*

Funció 6: Fitness del rànquing d'un fragment de TOS

Adequació dels tiquets d'un fragment de TOS als seients assignats des del punt de vista del rànquing. Calculem la mitjana dels ITF de tots els tiquets que pertanyen al fragment de TOS.

$$FragR_f = \frac{\sum_{i=1}^{ntf_f} ITF_i}{ntf_f} \qquad 0 \le FragR_f \le 100$$ **TOS Fragment Ranking**

On:

ntf_f = nº de tiquets del Fragment de TOS *f*

Funció 7: Fitness de l'agrupació d'un fragment de TOS

Calculem el fitness de l'agrupació d'un fragment d'un Ticket Order Subgroup comparant l'agrupació real amb l'agrupació ideal. L'agrupació real la calculem tinguent en compte el nombre de files i columnes ocupades pel fragment de TOS. L'agrupació ideal és la que minimitza la funció.

$$FragJ_f = 100 \cdot \frac{N'_f \cdot Y + M'_f \cdot X}{N_f \cdot Y + M_f \cdot X} \qquad 0 \le FragJ_f \le 100$$ **TOS Fragment Joint Fitness**

$$M'_f = \sqrt{\frac{ntf_f \cdot Y}{X}} \qquad N'_f = \frac{ntf_f}{M'_f}$$

On:

N_f = nombre de files diferents que ocupa el fragment *f*
N'_f = nombre de files ideal per tenir una agrupació màxima en el fragment *f*
M_f = nombre de columnes diferents que ocupa el fragment *f*
M'_f = nombre de columnes ideal per tenir una agrupació màxima en el fragment *f*
Y = constants que indica la distància entre dos seients sobreposats
X = constant que indica la distància entre dos seients de costat
ntf_f = nº de tiquets del Fragment de TOS *f*

Funció 8: Fitness d'un Fragment de TOS

Calculem el fitness d'un fragment de Ticket Order Subgroup ponderant el fitness del rànquing amb el fitness de l'agrupació i dividint-ho pels pesos de les relaxacions/desactivacions. Així doncs, la ponderació de fitness es veurà reduïda quan s'hagin hagut de relaxar/desactivar regles per poder assignar.

$$FragF_f = \frac{p_{TOSR} \cdot FragR_f + p_{TOSJ} \cdot FragJ_f}{(p_{TOSR} + p_{TOSJ}) \cdot \left(1 + \sum_{r=1}^{nr_f} q_r\right)} \qquad 0 \le FragF_f \le 100$$ **TOS Fragment Fitness**

On:

p_{TOSR} = pes del TOS Ranking Fitness
p_{TOSJ} = pes del TOS Joint Fitness
nr_f = nombre de regles relaxades/desactivades per assignar el fragment TOS *f*
q_r = importància de la regla relaxada/desactivada *r*

Funció 9: Fitness de tots els TOS d'un bloc

Ponderem el fitness de tots el TOS i fragments de TOS assignats a un bloc respecte el pes del TOS que li dóna el rànquing.

$$BTOS = \frac{\sum_{o=1}^{nob_k} TOSF_o \cdot w_o + \sum_{f=1}^{nfb_k} FragF_f \cdot w_f}{\sum_{o=1}^{nob_k} w_o + \sum_{f=1}^{nfb_k} w_f} \qquad 0 \le BTOS \le 100$$ **Block TOS Fitness**

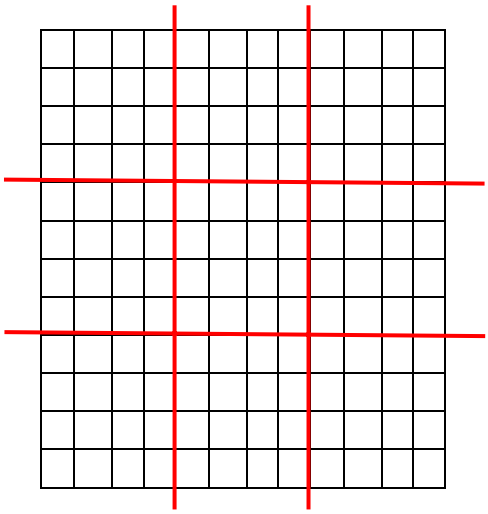
$$w_o = Max(rank) - rank_o$$
$$w_f = Max(rank) - rank_f$$

On:

nob_k = nº de TOS assignats al block *k*
w_o = pes del TOS *o*
$rank_o$ = rànquing del TOS *o*
w_f = pes del fragment de TOS f
$rank_f$ = rànquing del fragment de TOS *f* (el mateix del TOS al que pertany)
Max(rank) = el rànquing més alt assignat a un TOS

Funció 10: Dispersió dels tiquets assignats a un bloc

Partim el bloc en 9 parts iguals i calculem la desviació estàndard del percentatge de seients assignats a cada tros de bloc:



$$BD_k = 100 - \frac{2}{\sqrt{2}} \sqrt{\frac{\sum_{t=1}^{9} \left((P_t - Mean(Pt))^2 \cdot nst_t \right)}{\sum_{t=1}^{9} nst_t}} \qquad 0 \le BD_k \le 100 \qquad \textbf{Block Dispersion}$$

$$P_t = \frac{100 \cdot nat_t}{nst_t} \qquad Mean(Pt) = \frac{\sum_{t=1}^{9} P_t}{9}$$

On:

P_t = percentatge de seients assignats en el tros de bloc *t*
Mean(Pt) = mitjana dels percentatges de seients assignats a cada tros de bloc
Nst_t = nº de seients disponibles del tros de bloc *t*
Nsa_t = nº de seients assignats del tros de bloc *t*

Funció 11: Fitness d'un bloc

Calculem el fitness d'un bloc ponderant el fitness dels TOS assignats al bloc i de la dispersió del bloc.

$$BF_k = \frac{p_{BTOS} \cdot BTOS_k + p_{BD} \cdot BD_k}{\left(p_{BTOS} + p_{BD} \right)} \qquad 0 \le BF_k \le 100 \qquad \textbf{Block Fitness}$$

On:

p_{BTOS} = pes del Block TOS Fitness
p_{BD} = pes del Block Dispersion Fitness

Funció 12: Fitness global de tots els TOS del circuit

Ponderem el fitness de tots el TOS assignats respecte el pes del TOS que li dóna el rànquing.

$$GTOS = \frac{\sum_{o=1}^{no} TOSF_o \cdot w_o}{\sum_{o=1}^{no} w_o} \qquad 0 \le GTOS \le 100 \qquad \textbf{Global TOS Fitness}$$

$$w_o = Max(rank) - rank_o$$

On:

no = nº de TOS assignats
w_o = pes del TOS *o*
$rank_o$ = rànquing del TOS *o*
Max(rank) = el rànquing més alt assignat a un TOS

Funció 13: Dispersió global de tots els blocs del circuit

Ponderem la dispersió de cada bloc respecte el nombre de seients que tenen.

$$GBD = \frac{\sum_{k=1}^{nb} BD_k \cdot nsb_k}{\sum_{k=1}^{nb} nsb_k} \qquad 0 \le GBD \le 100 \qquad \textbf{Global Block}$$

On:

nsb_k = nº de seients disponibles del bloc *k*
nb = nº de blocs del circuit

Funció 14: Desviació global dels tiquets assignats a cada bloc

Calculem la desviació estàndard dels percentatges de seients assignats a cada bloc.

$$GDesv = 100 - \frac{2}{\sqrt{2}}\sqrt{\frac{\sum_{k=1}^{nb}\left(\left(P_k - Mean(Pk)\right)^2 \cdot nsb_k\right)}{\sum_{k=1}^{nb} nsb_k}} \qquad 0 \le GDesv \le 100$$ **Global Desviation**

$$P_k = \frac{100 \cdot nab_k}{nsb_k} \qquad Mean(Pk) = \frac{\sum_{k=1}^{nb} P_k}{nb}$$

On:

P_k = percentatge de seients assignats en el bloc *k*
Mean(Pk) = mitjana dels percentatges de seients assignats a cada bloc
Nsb_k = nº de seients disponibles del bloc *k*
Nb = nº de blocs del circuit
Nab_k = nº de seients assignats del bloc *k*

Funció 15: Dispersió global de tots els blocs del circuit

Ponderem la dispersió global de tots els blocs amb la desviació global de tiquets assignats a cada bloc.

$$GD = \frac{p_{GBD} \cdot GBD + p_{GDesv} \cdot GDesv}{p_{GBD} + p_{GDesv}} \qquad 0 \le GD \le 100$$ **Global Dispersion**

On:

p_{GBD} = pes del Global Block Dispersion
p_{GDesv} = pes del Global Desviation

Funció 16: Fitness global de tota l'assignació

Ponderem els fitness global dels TOS amb la dispersió de tot el circuit penalitzant en el cas de deixar tiquets no assignats.

$$GF = \frac{p_{GTOS} \cdot GTOS + p_{GD} \cdot GD}{p_{GTOS} + p_{GD} + nu^{\,p_u}} \qquad 0 \le GF \le 100$$ **Global Fitness**

On:

p_{GTOS} = pes del Global TOS Fitness
p_{GD} = pes del Global Dispersion Fitness
nu = nº de tiquets individuals no assignats
p_u = pes per la penalització dels tiquets individuals no assignats

3.4.5 Glossari

VARIABLE	DESCRIPCIÓ	ÀMBIT
TR_i	rànquing global del tiquet *i*	Tiquet Individual
Max(SR)	el rànquing global més alt assignat a un seient	
SR_j	rànquing global del seient *j*	
Max(TR)	el rànquing global més alt assignat a un tiquet	
Max(rank)	el rànquing més alt assignat a un TOS	
w_i	pes del tiquet individual *i*	
nto_o	nº de tiquets del TOS *o*	TOS
N_o	nº de files diferents que ocupa l'ordre *o*	
N'_o	nº de files ideal per tenir una agrupació màxima en l'ordre *o*	
M_o	nº de columnes diferents que ocupa l'ordre *o*	
M'_o	nº de columnes ideal per tenir una agrupació màxima en l'ordre *o*	
nr_o	nº de regles relaxades/desactivades per assignar el TOS *o*	
w_o	pes del TOS *o*	
$rank_o$	rànquing del TOS *o*	
ntf_f	nº de tiquets del fragment de TOS *f*	Fragment de TOS
N_f	nº de files diferents que ocupa el fragment *f*	
N'_f	nº de files ideal per tenir una agrupació màxima en el fragment *f*	
M_f	nº de columnes diferents que ocupa el fragment *f*	
M'_f	nº de columnes ideal per tenir una agrupació màxima en el fragment *f*	
nr_f	nº de regles relaxades/desactivades per assignar el fragment de TOS *f*	
w_f	pes del fragment de TOS *f*	
$rank_f$	rànquing del fragment de TOS *f*	
q_r	importància de la regla relaxada/desactivada *r*	Regla
nsb_k	nº de seients disponibles del bloc *k*	Bloc
nab_k	nº de seients assignats del bloc *k*	
ntb_k	nº de tiquets assignats del bloc *k*	
nob_k	nº de TOS continguts en el bloc *k*	
nfb_k	nº de Fragments de TOS continguts en el bloc *k*	
P_k	percentatge de seients assignats en el bloc *k*	
Mean(Pk)	mitjana dels percentatges de seients assignats a cada bloc	
nst_t	nº de seients disponibles del tros de bloc *t*	Tros de Bloc
nsa_t	nº de seients assignats del tros de bloc *t*	
P_t	percentatge de seients assignats en el tros de bloc *t*	
Mean(Pt)	mitjana dels percentatges de seients assignats a cada tros de bloc	
no	nº de TOS assignats en tot el circuit	Estadi
nu	nº de tiquets individuals no assignats	
nb	nº de blocs del circuit	

CONSTANT	DESCRIPCIÓ	VALOR INICIAL
Y	constant que indica la distància entre dos seients sobreposats	2.5
X	constant que indica la distància entre dos seients de costat	1
p_{TOSR}	pes del TOS Ranking Fitness	1
p_{TOSJ}	pes del TOS Joint Fitness	1
p_{BTOS}	pes del Block TOS Fitness	1
p_{BD}	pes del Block Dispersion Fitness	0 (si NO dispersió) 1 (si dispersió)
p_{GBD}	pes del Global Block Dispersion	1
p_{GDesv}	pes del Global Desviation	2
p_{GTOS}	pes del Global TOS Fitness	1
p_{GD}	pes del Global Dispersion Fitness	0 (si NO dispersió) 1 (si dispersió)
p_u	pes per a la penalització dels tiquets individuals no assignats	0.5

4 Implementació

4.1 Introducció

El propòsit d'aquest capítol és mostrar la implementació que s'ha fet del sistema de distribució d'espectadors. Primer es justificarà l'elecció del sistema de desenvolupament triat. Tot seguit s'explicarà l'arquitectura general del sistema i es mostraran les interfícies, les estructures de dades i classes que s'han utilitzat així com els mètodes i atributs principals de cada classe. Al final d'aquest document també s'indica com es pot integrar l'algorisme en una altra aplicació.

4.2 Sistema de desenvolupament utilitzat

Els requeriments especifiquen que l'aplicació s'ha de desenvolupar en Microsoft Visual Studio .NET, perquè la integració amb els sistemes existents sigui immediata.

La plataforma de desenvolupament Microsoft Visual Studio .NET permet programar amb llenguatge Visual Basic, Visual C++, CS, Java, PHP, etc. indistintament ja que tots tenen el mateix accés a les llibreries ADO.NET.

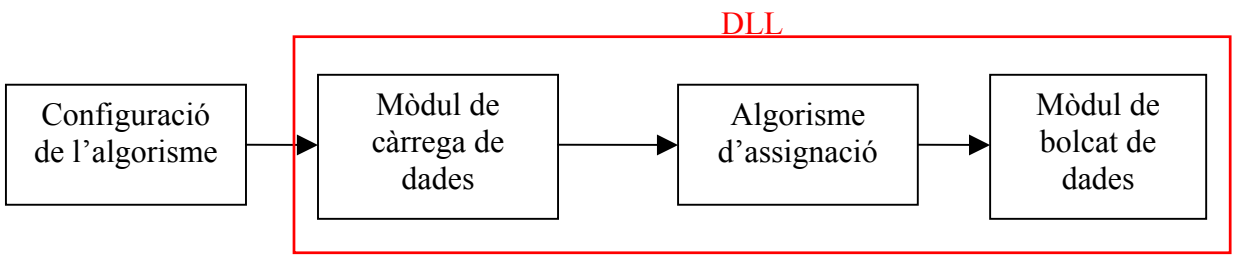
Com que el temps d'execució és un requisit important en l'aplicació no serà escaient la utilització de llenguatges interpretats (com per exemple Java). D'entre els llenguatges compilats restants s'ha escollit el Visual C++ perquè és el que dona més llibertat al programador i el més optimitzat i eficient en temps d'execució.

4.3 Arquitectura del sistema

Implementarem el sistema amb una arquitectura tipus client. L'usuari interactuarà amb l'aplicació donant-li la informació que aquest li demana i l'aplicació li mostrarà la solució corresponent a aquestes entrades.

4.3.1 Esquema general

L'aplicació estarà formada per dos blocs bàsics: una interfície de configuració de l'algorisme i una DLL que serà l'encarregada de realitzar l'assignació.



Dins de la DLL tindrem tres blocs: el mòdul de càrrega de dades, l'algorisme d'assignació i el mòdul de bolcat de resultats.

Mitjançant el bloc de configuració, configurarem els paràmetres de l'algorisme i cridarem a una funció de la DLL que serà l'encarregada de realitzar l'assignació.

Mitjançant aquesta estructura, l'algorisme serà fàcilment integrable en una altra aplicació, ja que només haurà d'integrar la DLL i cridar el mètode de l'algorisme amb tots els paràmetres necessaris.

4.4 Interfícies de configuració de l'algorisme

La configuració comprèn les dades d'entrada, les regles, relaxacions, paràmetres de la funció de fitness i condicions de finalització de l'execució.

4.4.1 Interfície d'entrada de dades

Aquest és un exemple de la interfície d'entrada de dades que tindrà el sistema. En aquesta interfície ha de ser fàcil entrar la localització de tots els fitxers d'entrada necessaris: blocs, seients, ordres, segments del mercat, customer groups, groups sales customers, categories i les matrius de disponibilitat, restricció i compatibilitat.

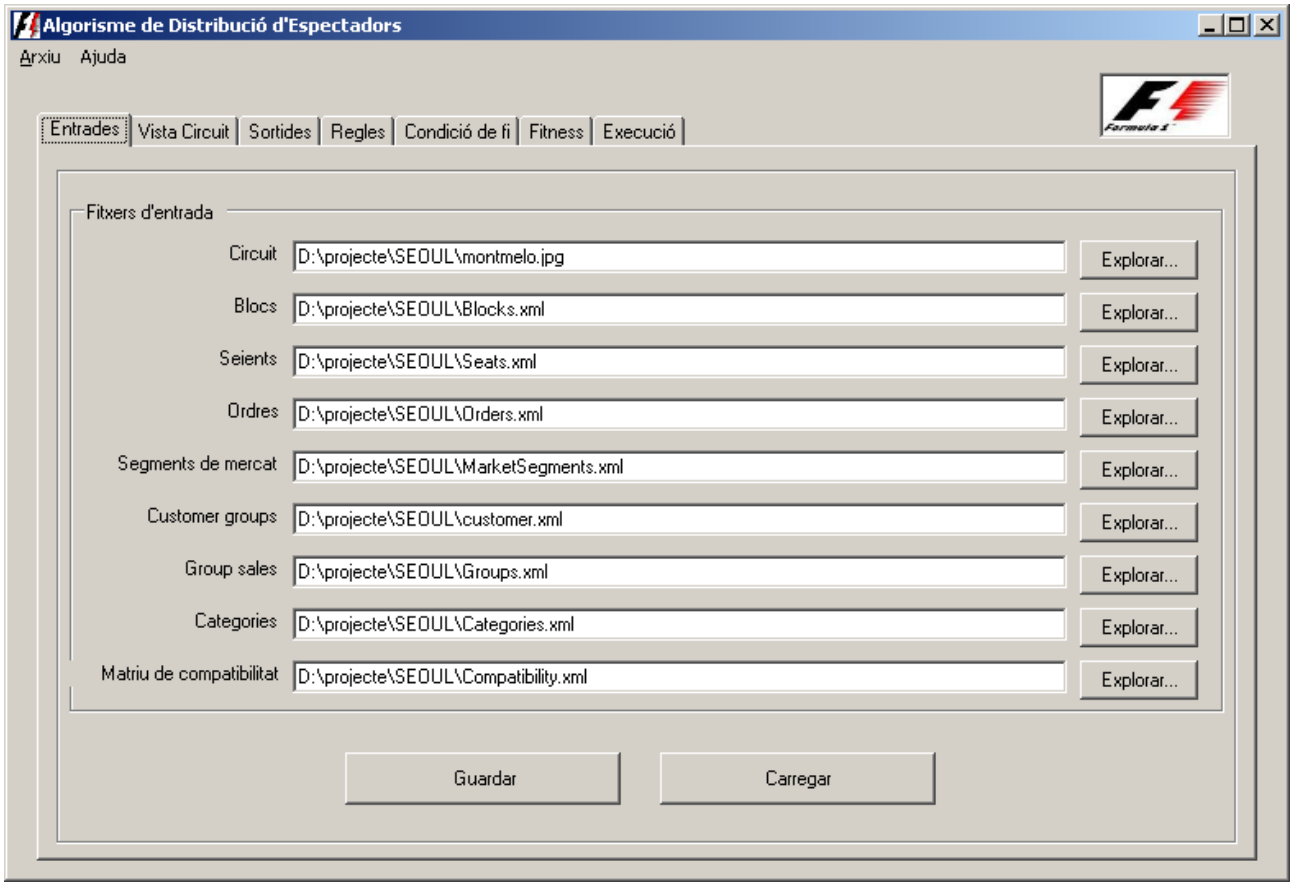


Figura 4.1 Interfície d'entrada de dades

Per no haver d'entrar tots els fitxers cada vegada que s'executi l'aplicació hi hauran dos botons per carregar i guardar els fitxers entrats.

4.4.1.1 Interfície de visualització del circuit

Les dades entrades referents al circuit entrades en la pantalla anterior seran representades gràficament en una interfície que mostrarà una vista esquemàtica del circuit, amb els seus blocs. Clicant sobre algun dels blocs apareixerà una interfície de visualització dels blocs.

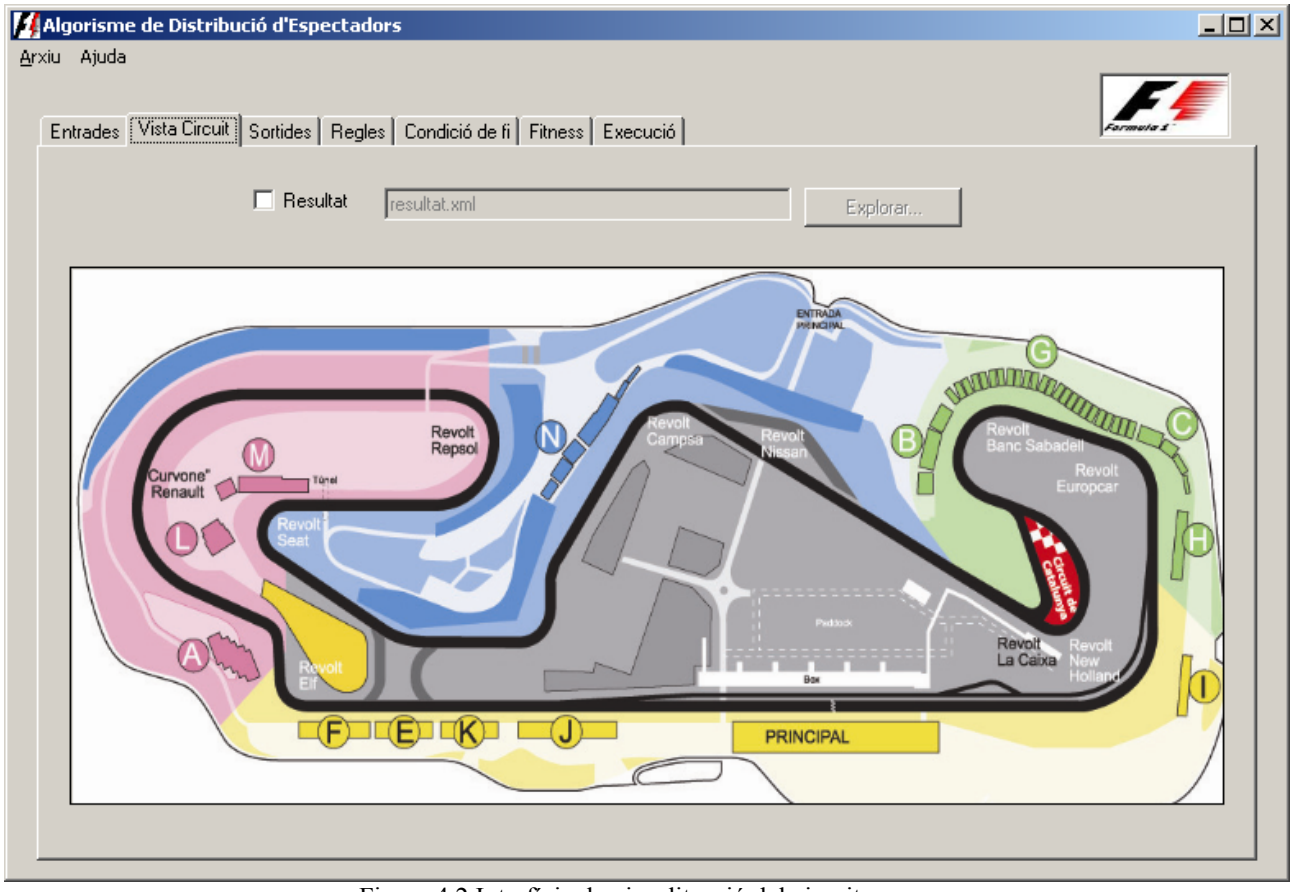


Figura 4.2 Interfície de visualització del circuit

4.4.1.2 Interfície de visualització dels blocs

Aquesta interfície permetrà visualitzar tots els blocs que formin part de la zona del circuit seleccionada en la pantalla anterior. A la representació gràfica hi apareixeran tots els seients del bloc representats en una matriu.

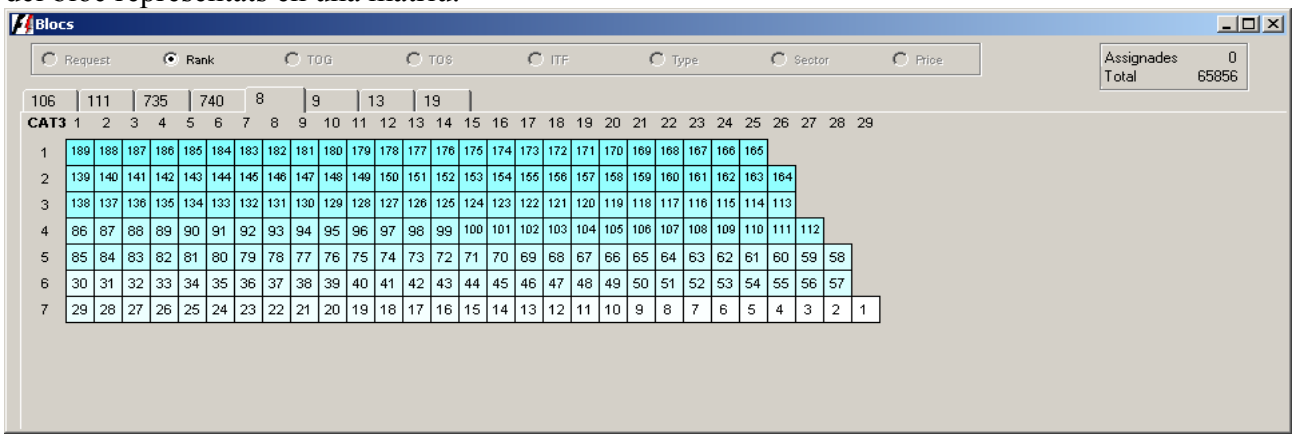


Figura 4.3 Interfície de visualització dels blocs

4.4.2 Interfície d'entrada de regles

En aquest panell hi apareixen totes les regles que poden desactivar-se o ser relaxades (3 - 13) amb els seus paràmetres corresponents. Les regles obligatòries no hi apareixen ja que són fixes i no configurables.

Aquestes seran les regles que inicialment s'aplicaran. Només es desactivaran o relaxaran si així s'indica explícitament en la configuració de les relaxacions.

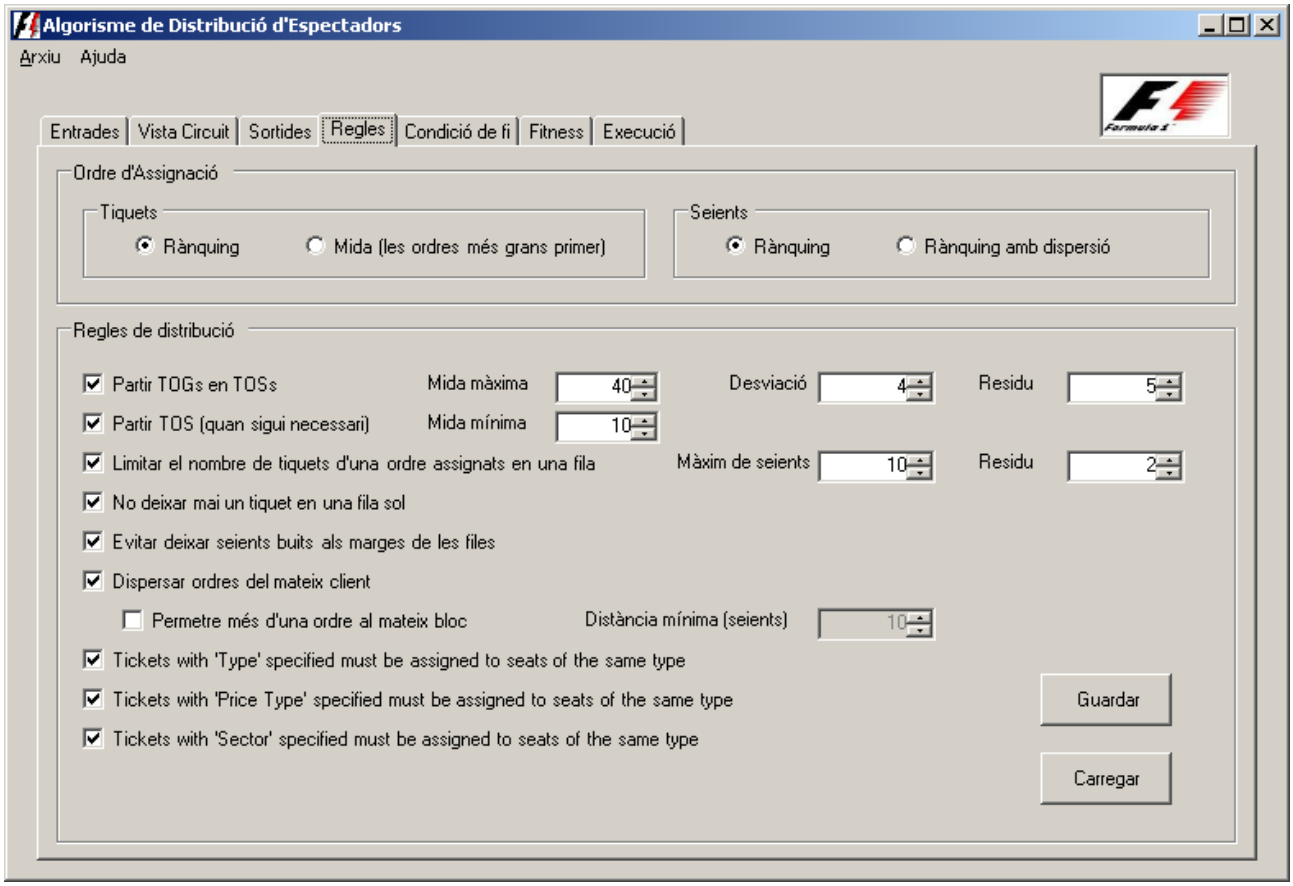


Figura 4.4 Interfície d'entrada de regles

Com es pot veure finalment s'ha incorporat l'opció citada a l'apartat 3.3.4.2 del disseny que consistia en assignar les ordres per mides en comptes de per rànquing.

4.4.3 Interfície de selecció de condició de finalització

En aquest panell es podrà seleccionar una de les condicions de finalització que s'han explicat a la secció de disseny:

- Assignació total de les ordres
- Assignació parcial de les ordres
- Assignació total o parcial de les ordres amb desactivació i relaxació automàtica de regles

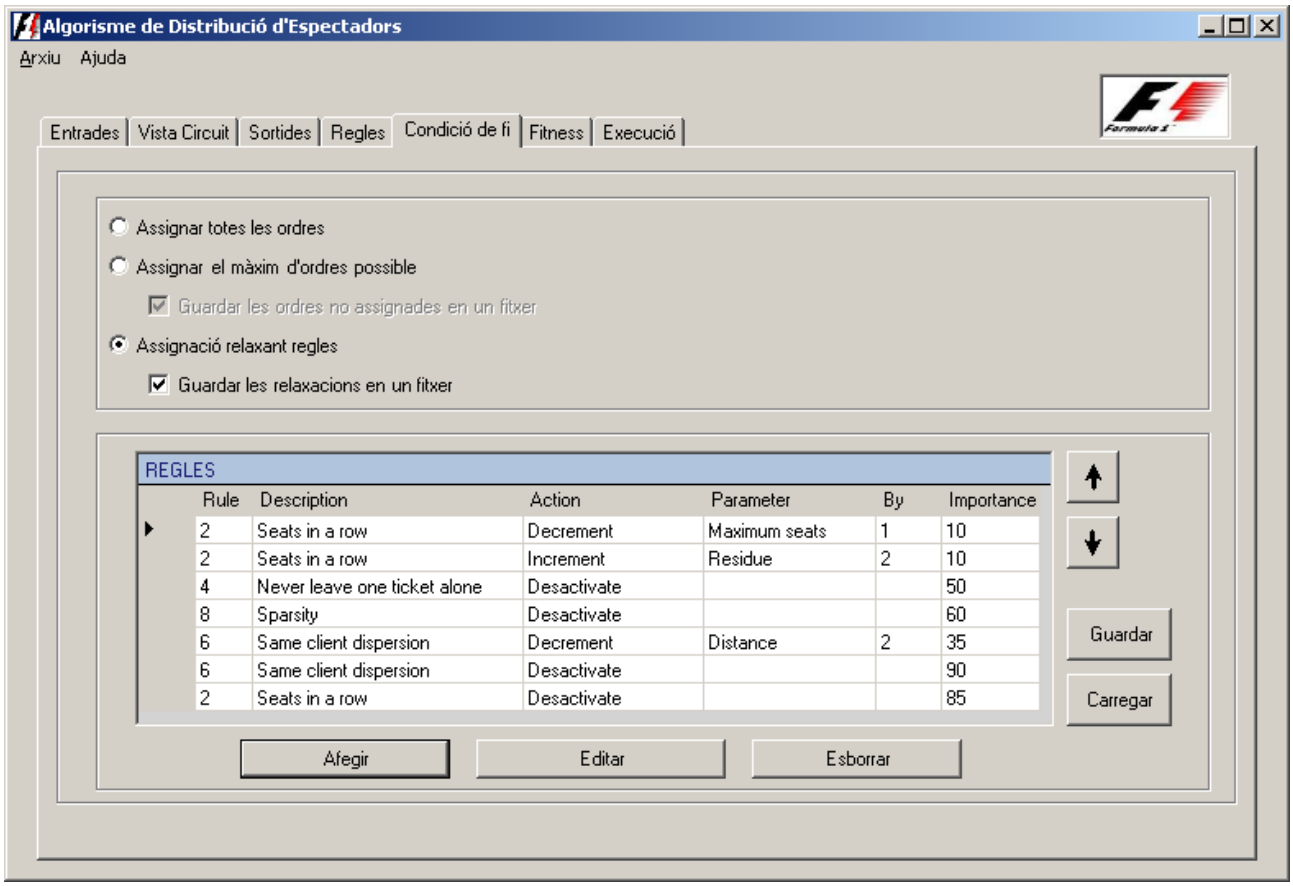


Figura 4.5 Interfície de selecció de la condició de finalització

En el cas d'assignació amb desactivació i relaxació de regles automàtica, l'ordre amb què es desactiven o es relaxen les regles s'ha de definir. Aquestes accions a realitzar es guarden en una taula i l'algorisme quan es trobi en una situació on no pot assignar més ordres (seguint les regles inicials) començarà a realitzar les accions en l'ordre que haguem decidit.

El camp importància defineix la gravetat de realitzar una desactivació/relaxació. Aquest fet es veurà reflectit en la funció de fitness.

REGLES

Rule	Description	Action	Parameter	By	Importance
2	Seats in a row	Decrement	Maximum seats	1	10
2	Seats in a row	Increment	Residue	2	10
4	Never leave one ticket alone	Desactivate			50
8	Sparsity	Desactivate			60
6	Same client dispersion	Decrement	Distance	2	35
6	Same client dispersion	Desactivate			90
2	Seats in a row	Desactivate			85

Figura 4.6 Exemple de taula de relaxacions

En aquesta llista d'accions a realitzar amb regles se n'hi poden afegir de noves, modificar les existents (augmentant o disminuint la seva prioritat, modificant paràmetres, etc) i eliminar les que no interessin.

4.4.4 Interfície de selecció de les dades de sortida

En el cas dels fitxers de sortida s'especificarà només el directori on es guardaran tots els fitxers generats.

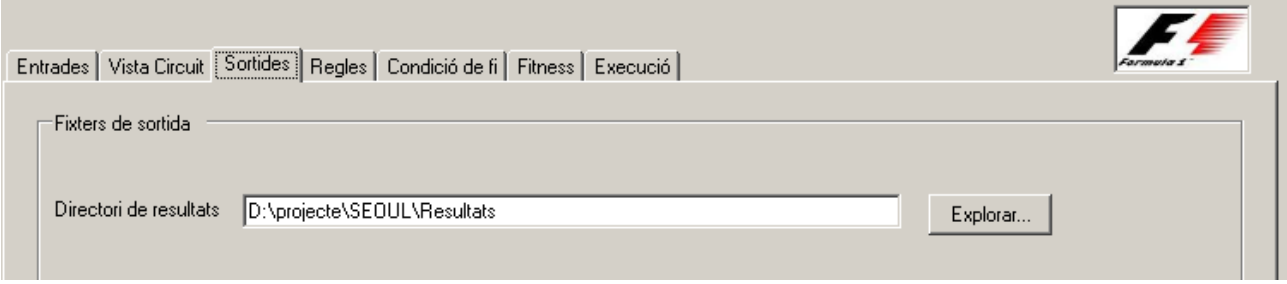


Figura 4.7 Interfície de selecció de les dades de sortida

El programa generarà diferents fitxers en funció de les opcions d'execució que s'hagin triat. La relació entre les condicions de finalització i els fitxers que obtindrem de sortida és aquesta:

1. Sigui quina sigui l'opció que triem el fitxer d'ordres assignades es generarà amb el nom "assignades.xml".

2. Si triem l'opció de no assignar totes les ordres i activem la possibilitat de crear el fitxer d'ordres no assignades també es generarà el fitxer d'ordres no assignades amb el nom "no_assignades.xml.

3. Si seleccionem l'opció d'assignació desactivant i relaxant automàticament les regles també es generarà el fitxer d'ordres no assignades. I si a més a més activem l'opció de generar el registre d'incidències, es generarà també aquest fitxer amb el nom "log.xml".

4.4.5 Interfície de paràmetres de la funció de fitness

Tal com s'ha definit la funció de fitness al capítol de disseny, aquesta està formada per una sèrie de paràmetres i pesos que poden ser configurables. El aquesta interfície es mostraran tots els paràmetres i podran ser modificats.

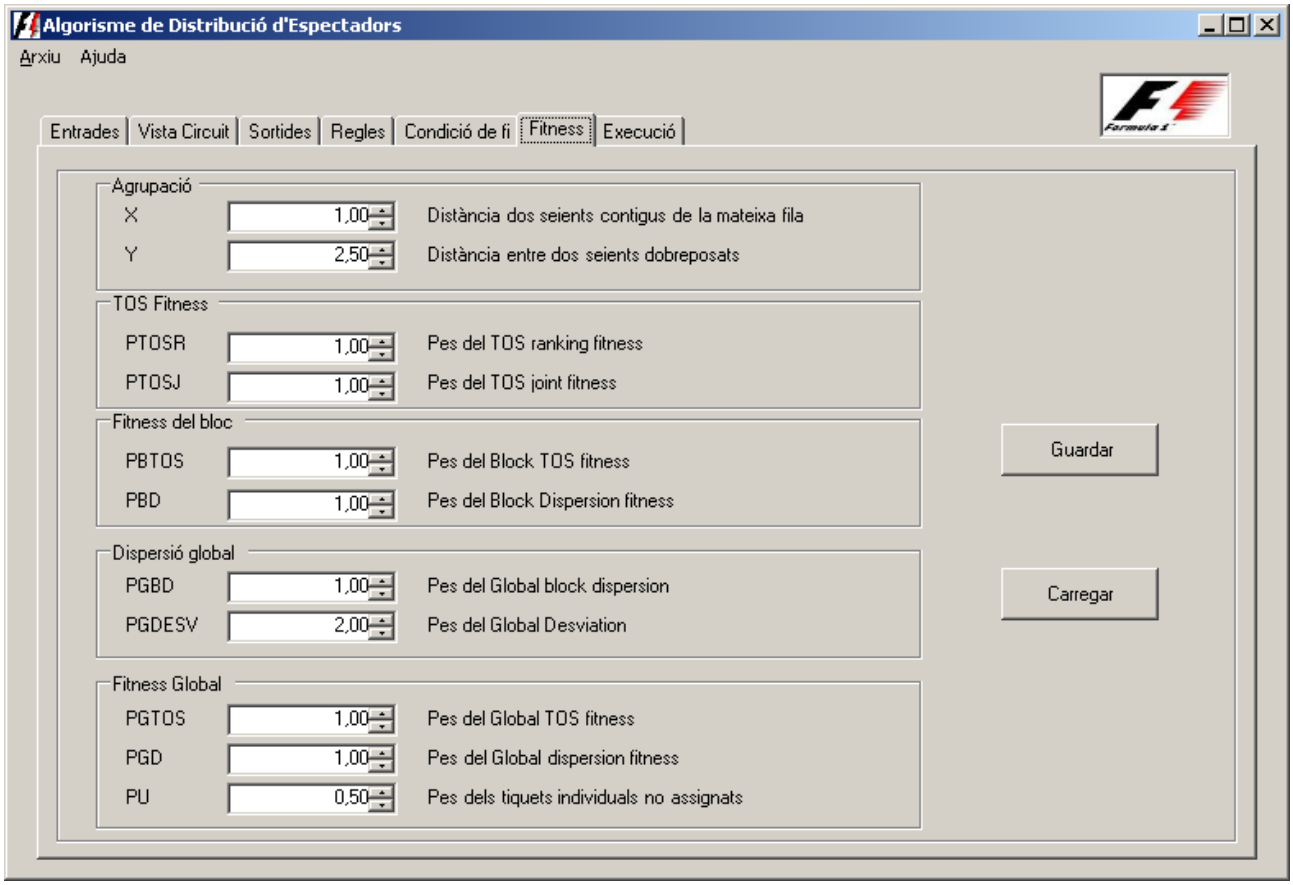


Figura 4.8 Interfície de paràmetres de la funció de fitness

4.4.6 Interfície d'execució

Un cop s'han configurat tots els paràmetres, podem passar ja a executar l'algorisme i trobar les assignacions que satisfacin les restriccions que haguem fixat.

Hi ha diferents modes d'execució de l'algorisme. Pot ser una execució interactiva, on l'usuari decideix el moment en que comença l'execució i pot aturar l'execució quan li interessi. També pot ser una execució amb un temps de durada. I finalment té la opció d'executar-se fins trobar una solució amb un fitness concret o fins que es trobin les primeres *n* solucions (amb *n* definit per l'usuari).

La interfície té les opcions d'execució esmentades a l'anterior paràgraf per especificar quan volem que s'aturi l'execució. A més podem seleccionar quantes solucions diferents volem guardar. També apareix una llista de les categories on podem seleccionar les categories en les quals volem realitzar el procés d'assignació.

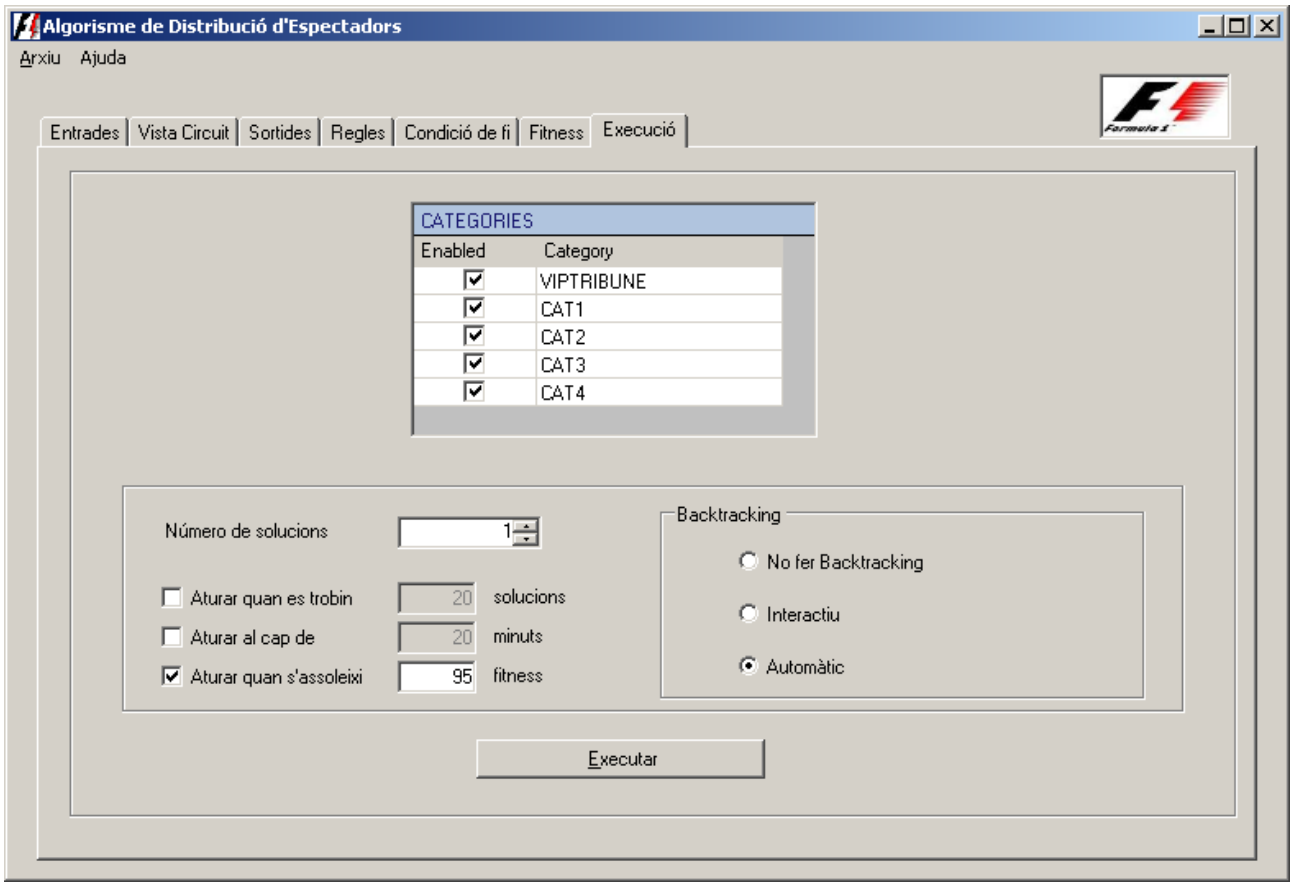


Figura 4.9 Interfície d'execució

4.4.7 Interfície d'execució de la llibreria

Quan es clica sobre el botó Executar apareix la interfície d'execució de la llibreria dinàmica (ade.dll). És en aquesta on veiem l'evolució de l'execució, amb un camp per cada categoria que indica el percentatge que es porta de la primera solució, un semàfor que es posa vermell quan es troba la primera solució, un camp d'estat que informa de si la cerca en la categoria s'està executant, o si està parada o si ja ha finalitzat. A més a més surten columnes de fitness de les solucions trobades.

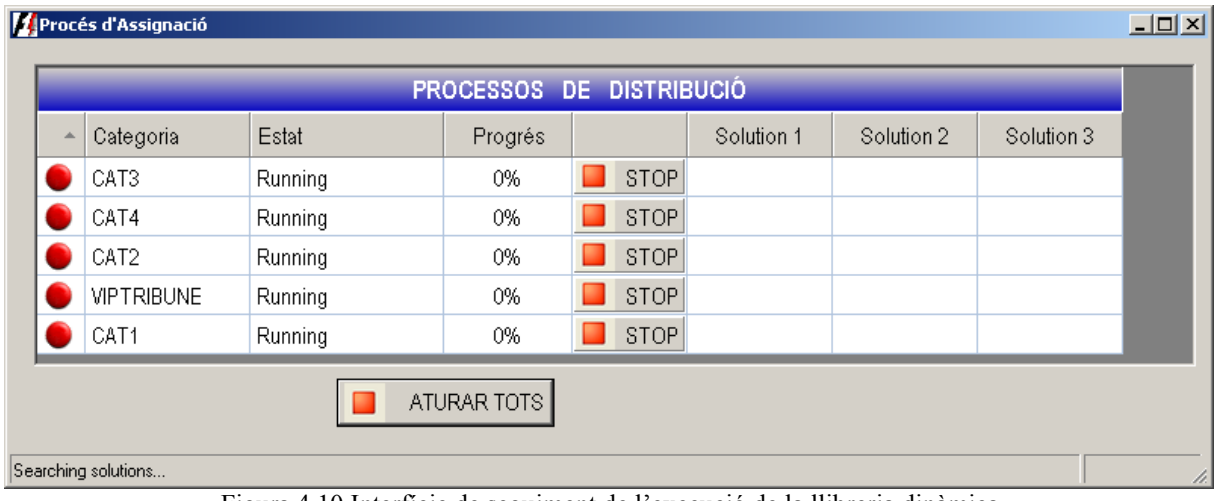


Figura 4.10 Interfície de seguiment de l'execució de la llibreria dinàmica

Quan finalitzi l'exeució es poden veure els resultats amb la interfície de visualització del circuit i de visualització dels blocs definides anteriorment.

4.4.7.1 Backtracking Interactiu

En el cas que hàgim seleccionat realitzar un backtracking interactiu, ens aparaixerà una interfície de configuració del backtracking (veure figura 4.11).

Block	Rank	Total seats	Unassigned	Assigned	BTOS	BD	Total fitness	Backtrack
361	2	112	28	84	96,57705	39	96,57705	☑
362	2	183	22	161	78,83591	58	78,83591	☐
363	2	98	14	84	86,57705	73	86,57705	☑
364	2	98	14	84	86,57705	68	86,57705	☑
365	2	183	76	107	83,04562	46	83,04562	☐
366	2	112	70	42	68,57705	32	68,57705	☑

Figura 4.11 Interfície de configuració del backtracking interactiu

Aquesta interfície mostra, un cop trobada la primera solució de l'assignació, informació sobre cadascun dels blocs de la categoria corresponent. La informació mostrada és la següent:

- Block: número del bloc.
- Rank: rànquing del bloc.
- Total seats: nombre total de seients de que disposa el bloc per assignar.
- Unassigned: nombre de seients del bloc que no s'han pogut assignar.
- Assigned: nombre de seients del bloc que s'ha pogut assignar.
- BTOS: fitness agregat de les ordres assignades al bloc.
- BD: dispersió de les ordres dins del bloc (només tindrà sentit quan l'usuari hagi activat l'assignació amb 'Ranking with Sparsity')
- Total fitness: fitness de bloc.

Llavors, la interfície permet a l'usuari seleccionar al seu criteri sobre quins blocs s'ha de realitzar el backtracking marcant el 'Checkbox' de la columna 'Backtrack'.

4.5 Estructures de dades

Les dades que utilitzarà l'algorisme són les dades del circuit, dels seients, de les entrades venudes, etc. El mòdul de càrrega de dades s'encarrega de proporcionar a l'algorisme les dades necessàries per a realitzar l'assignació mitjançant fitxers.

El format dels fitxers serà XML amb l'estructura que s'especifica a l'Annex "Format dels fitxers d'entrada / sortida", i tindran una estructura equivalent a la base de dades original. Totes les dades es guardaran a memòria principal per accelerar l'execució. Les estructures de dades utilitzades internament seran DataSets i DataTables del .NET. Tot i que també es crearan classes especials per algunes de les dades.

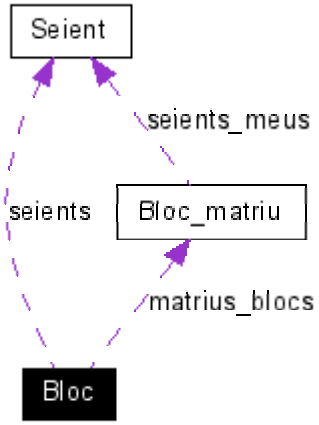
Les classes utilitzades són:

- Blocs
- BlocMatriu
- Seients
- Ordres
- Solucio
- Regles

4.5.1 Classe Blocs

Aquesta classe contindrà la informació sobre els blocs del circuit, els quals tindran les dades dels seients.

4.5.1.1 Diagrama de col·laboració



4.5.1.2 Atributs principals

DataTable *	**dtBlocs**
Seient *	**seients**
Bloc_matriu *	**matrius_blocs** []

- dtBlocs

En aquest DataTable s'hi guarda la informació llegida al fitxer de blocs.

- seients

Aquest atribut és un punter que fa referència a l'objecte de seients, que conté tots els seients.

- matrius_blocs

Aquesta és una llista d'objectes Bloc_matriu. Tindrem un objecte Bloc_matriu per cada bloc.

4.5.1.3 Mètodes principals

void	**Genera_solucio** (**Solucio** *&solucions, **Ordre** *ORDRES, String *categoria)
void	**marca_ordres_mes_grans_que_forat_mes_desviacio** (**Ordre** *&ordres, int max_forat, int prof)
void	**marcar_ordres_de_les_mateixes_caracteristiques** (int i_bloc, DataRow *ordre, int motiu, DataRow *llavor, **Ordre** *&ordres, int profunditat, **Regles** *regles)
void	**desmarcar_ordres** (**Ordre** *&ordres)
int	**Busca_bloc_amb_seients_lliures** (DataRow *&ordre, int percentatge_ocupacio, **Regles** *regles)
bool	**hi_cap_la_ordre** (DataRow *&ordre, int i_bloc, String *&filtre_seients_conjunt, int &max_forat, **Regles** *regles, String *&filtre_forat)
void	**marca_seients_del_forat_com_a_prohibits_per_sempre** (int i_bloc, String *filtre_seients_conjunt)
int	**busca_llavor** (DataRow *&llavor, int i, String *filtre_seients_conjunt, DataRow *&ordre, int &index, **Ordre** *&ordres, ArrayList *&lSeientsProhibits, int prof, **Regles** *regles)
void	**afegeix_a_prohibits_tots_els_massa_propers** (int i, DataRow *ordre, ArrayList *&lSeientsProhibits, **Regles** *regles)

- Genera_solucio

Crea l'objecte solució recorrent tots els blocs i ajuntant la informació sobre les assignacions realitzades a cadascun dels blocs que li proporciona la funció Afegeix_assignacions de la classe Bloc_matriu.

- marca_ordres_mes_grans_que_forat_mes_desviacio

Aquest és un dels blocs de l'esquema general de l'algorisme mostrat a l'apartat 3.3.5. La seva funció és posar una marca a les ordres no assignades que la seva mida (nombre de tiquets) sigui més gran o igual a una mida determinada passada per paràmetre.

- marcar_ordres_de_les_mateixes_caracteristiques

Semblant a l'anterior aquesta el que ha de fer és marcar totes les ordres no assignades de la llista que siguin semblants a una altra. El criteri de semblança el defineix el paràmetre motiu.

- desmarcar_ordres

Treu totes les marques posades pels dos mètodes anteriors.

- Busca_bloc_amb_seients_lliures

Aquesta funció ha de retornar el millor bloc de la categoria amb algun seient lliure. Per fer això haurà de mirar quants seients lliures no prohibits té a cada bloc. És una funció molt ràpida ja que totes aquestes dades es guarden a la classe Bloc_matriu que té cada bloc. També pot retornar error en el cas que no es trobi cap bloc amb seients lliures no prohibits.

- hi_cap_la_ordre

El bloc anterior només informa que el bloc té seients lliures. Amb aquesta funció s'esbrinarà si la ordre cap teòricament al bloc amb una sèrie de comprovacions ràpides (com s'explica al capítol de disseny aquesta funció el fet que aquesta funció retorni una llavor no assegurarà que la ordre es podrà assignar correctament en la llavor retornada). Retornarà error en cas que no sigui possible assignar l'ordre al bloc.

- marca_seients_del_forat_com_a_prohibit_per_sempre

Aquesta és la funció que marca com a prohibits els seients d'un bloc en els quals no és possible assignar-hi cap de les ordres que falten per assignar. S'anomena prohibits per sempre perquè els seients que es prohibeixin en aquest mètode romandran en aquest estat durant tota la assignació fins que es relaxi alguna regla o es comenci el backtracking.

- busca_llavor

Aquesta funció retornarà un seient del bloc que es farà servir de llavor per l'assignació. En cas que no es trobi cap llavor retornarà el perquè no s'ha trobat.

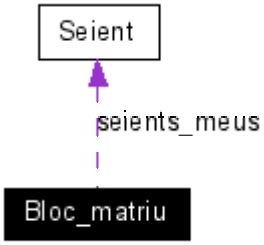
- afegeix_a_prohibits_tots_els_massa_propers

Funció que genera una llista amb els seients lliures del bloc que no compleixen la distància mínima entre ordres del mateix client amb la ordre que es vol assignar.

4.5.2 Classe BlocMatriu

Aquesta classe representa cadascun dels blocs amb una matriu on cada casella de la matriu és un seient. Aquesta manera de representar els blocs és molt útil per realitzar la distribució de les ordres. La matriu és de tres dimensions. Les dues primeres identificaran els seients segons la seva posició (fila, columna) i a la tercera dimensió que anomenarem capa s'hi posarà informació addicional per tal d'accelerar tot el procés.

4.5.2.1 Diagrama de col·laboració



4.5.2.2 Atributs principals

Seient *	**seients_meus**
int	**seients_lliures**
float	**bd**
float	**btos**
float	**total_fitness**
String *	**matriu** [,,]
int seients_contigus	**__gc** []
ArrayList *	**lMidesForats**
DataTable *	**clients_en_aquest_bloc**
ArrayList *	**lProhibitsPerLlavor**
ArrayList *	**lProhibitsPerSempre**

- seients_meus

Aquest és un objecte de tipus Seient on hi ha tots els seients que pertanyen al bloc respectiu. És molt útil tenir-lo i accelera molt el procés degut a què les operacions que es fan en un bloc només afecten als seients del propi bloc, i tenir-los d'aquesta manera en cada bloc tot i que duplica la informació és molt més ràpid.

- seients_lliures

Comptador dels seients lliures del bloc molt utilitzat en vàries funcions com per exemple "Busca_bloc_amb_seients_lliures", "hi_cap_la_ordre" de la classe Bloc i també en el backtracking. S'ha d'actualitzar cada cop que s'assignen o es desassignen tiquets al bloc.

- bd, bts, total_fitness

Aquests són els diferents valors de la funció de fitness referents als blocs.

- matriu

Matriu de 3 dimensions. Com s'ha dit a l'apartat introductori les dues primeres dimensions identificaran als seients i la tercera es farà servir per emmagatzemar diversa informació addicional que accelerarà el procés.

- seients_contigus

És un vector de tantes posicions com files hi hagi a la matriu que informa de quants seients contigus d'una mateixa ordre hi ha en una fila. Cada vegada que s'inicia l'assignació d'una ordre s'inicialitzen totes les posicions a zero i cada seient ocupat incrementarà en 1 la posició de la fila corresponent. Servirà per tenir un control ràpid dels seients ocupats en cada fila per una ordre.

- lMidesForats

Aquesta és una llista amb tantes posicions com forats hi hagi a la matriu i informa sobre la mida dels forats. Per realitzar el càlcul d'aquesta llista es fa servir l'algorisme CCL (Connected Component Labelling).

- clients_en_aquest_bloc

Llista inicialment buida de totes les ordres que hi han assignades en aquest bloc. Té informació la caixa envolvent (bounding box) de l'ordre, és a dir, el rang de files i columnes que ocupa. Serveix per accelerar el procés de distanciar ordres d'un mateix client que han de ser dispersades.

- lProhibitsPerLlavor

Llista de seients del bloc prohibits per llavor.

- lProhibitsPerSempre

Llista de seients del bloc prohibits per sempre.

4.5.2.3 Mètodes principals

int	**Assigna_ordre** (int x, int y, DataRow *ordre, **Ordre** *&ordres, ArrayList *&lSeientsAssignats, **Seient** *seients_tots, int node_i, ArrayList *lSeientsProhibits, **Regles** *regles)
void	**Afegeix_assignacions** (**Solucio** *&solucions, **Seient** *SEIENTS, **Ordre** *ORDRES, String *rank_bloc, int inici_rank_bloc)
void	**Buscar_contigus** (int i, int j, ArrayList *&l, int max_seients_fila)
int	**escollir_ne_un** (ArrayList *l, ArrayList *lAssignats, int &i, int &j, int &index)
void	**Eliminar_els_que_no_compleixen_les_regles** (ArrayList *l, int max_seients_fila)
String *	**Aplica_regla_one_ticket_alone** (ArrayList *&lSeientsAssignats, **Regles** *regles)
int	**Desassigna_propera_al_forat** (int i_forat, **Ordre** *&ordres, **Regles** *regles)
int	**Assigna_ordre_no_forats** (int x, int y, DataRow *ordre, **Ordre** *&ordres, ArrayList *&lSeientsAssignats, **Seient** *seients_tots, int node_i, ArrayList *lSeientsProhibits, **Regles** *regles)

- Assigna_ordre

Mètode basat en region growing per assignar una ordre de n tiquets a una llavor donada. Si l'assignació no es pot fer retorna un error indicant la causa de l'error.

- Afegeix_assignacions

Funció que recorre la matriu per generar la solució del bloc, o sigui, la relació de seients assignats a tiquets.

- Buscar_contigus, Eliminar_els_que_no_compleixen_les_regles, escollir_ne_un

Aquestes tres funcions les crida la funció assigna_ordre. Són les encarregades de generar la llista de candidats a ser assignats, eliminar d'aquests candidats els que no compleixen les regles (màxim de seients per fila, tipus, distàncies, etc) i escollir d'aquests candidats el millor (el que acabi donant un valor de fitness a l'ordre en conjunt més alt).

- Aplica_regla_one_ticket_alone

Efectua els mecanismes per resoldre aquesta situació especificats al capítol de disseny apartat ?. En cas que continuï quedant algun tiquet sol en una fila es retornarà un error i l'assignació es desfarà.

- desassigna_propera_al_forat

Funció feta servir a la fase de backtracking per assignar les ordres no assignades. Busca les ordres properes a forats i en desassigna la de menor mida.

- Assigna_ordre_no_forats

Idèntica a la d'assignar_ordre excepte que en aquest cas només es considera que la assignació és correcte si no queden forats, és a dir, l'ordre no té contigu cap seient lliure.

4.5.3 Classe Seients

Està formada per una taula amb tota la informació de tots els seients del circuit recollida en el fitxer de seients d'entrada.

4.5.3.1 Atributs principals

DataTable *	**dtSeients**

- dtSeients

Aquesta taula guardarà tots els seients del circuit o només el d'un bloc depenent de si es troba a la classe principal (en aquesta hi són tots els del circuit) o a la classe bloc_matriu (aquesta només requereix els seients del bloc al que fa referència).

4.5.4 Classe Ordres

Classe per emmagatzemar les ordres que han de ser assignades pel procés de distribució.

4.5.4.1 Atributs principals

DataTable *	**dtOrdres**

- dtOrdres

Cada arbre d'execució d'una categoria tindrà una sola taula d'ordres. Aquesta taula guardarà tant les ordres (TOS) assignades com les no assignades. Disposarà d'un camp que identificarà les ordres que no s'han assignat encara, les que ja estan assignades, les que no s'han pogut assignar i les que temporalment no volem assignar (ordres marcades).

4.5.4.2 Mètodes principals

void	**Split_ticket_order_group** (int mida_TOS, int residu_TOS)
void	**filtra** (**Ordre** *ordres, String *columna, String *valor)
DataRow *	**Escull_ordre** (int &index, int profunditat)
Ordre *	**Ordena_per_mida** ()
Ordre *	**Ordena_aleatoriament** ()

- Split_ticket_order_group

Aquesta funció és l'encarregada de partir les ordres (Requests) en TOS i dispersar-los utilitzant el mètode explicat a l'apartat 3.2.3.1.

- filtra

La funció filtra es crida just abans d'iniciar els threads per cada categoria. Rep com a entrada les ordres (TOS, ja que prèviament s'ha executat la funció Split_ticket_order_group) i d'aquestes tria les de la categoria corresponent.

- Escull_ordre

Aquest és el mètode fet servir a l'algorisme per retornar la millor ordre de la taula que no estigui assignada per ser tractada. El que fa realment es retornar la primera ordre de la llista que no estigui assignada, ja que les ordres a la taula estan ordenades per rànquing o per mida depenent de la tècnica d'assignació que s'hagi triat.

- Ordena_per_mida

La tècnica d'assignació es pot fer tant per ordre de rànquing com per ordre de mides (les de mida més gran primer). Per tant, en el cas que s'hagi seleccionat aquesta tècnica s'ordenen les ordres per mida, d'aquesta manera no s'han de tenir dos mètodes Escull_ordre sinó que amb una ordenació feta a l'inici s'aconsegueix complir amb el mètode desitjat.

- Ordena aleatòriament

La ordenació aleatòria és útil per la fase de backtracking tal com es justifica a l'apartat 3.3.7.2.2.

4.5.5 Classe Solucio

Classe encarregada d'allotjar la solució. És a dir, l'associació de cada tiquet assignat amb el seu seient, així com la llista de tiquets o ordres no assignades i el fitness global de tota la solució.

4.5.5.1 Atributs principals

DataTable *	**dtSolucions**
DataTable *	**dtNoAssignades**
DataTable *	**dtTOS**
DataTable *	**dtBLOCS**
double	**fitness_total_tt**

- dtSolucions

Aquesta és la taula Assignacions de l'esquema entitat-relació vist a l'apartat 3.2.3. Aquí és on s'hi emmagatzemen les ordres assignades. És una taula que relaciona cadascun dels tiquets de les ordres assignades amb els seus seients corresponents.

- dtNoAssignades

Com el nom indica, en aquesta taula hi posarem les ordres que no ha estat possible assignar ni partint ni relaxant. No tindrà una fila per cada tiquet assignat com l'anterior sinó una per cada ordre assignada, ja que això voldrà dir que tots els tiquets de l'ordre han quedat sense assignar.

- dtTOS

En aquesta taula hi guardarem els valors de fitness de totes les ordres assignades, tal com s'ha definit en la funció de fitness a l'apartat 3.4.

- dtBLOCS

Aquesta taula semblant a l'anterior contindrà els valors de fitness dels blocs del circuit. El càlcul d'aquest valor s'especifica en l'apartat 3.4.

4.5.6 Classe Regles

Aquesta classe tindrà tota la informació sobre les regles o restriccions no obligatòries que l'usuari ha configurat. També tindrà la llista ordenada de relaxacions en cas que s'hagi habilitat aquesta opció.

4.5.6.1 Atributs principals

int	**MIDA_TOS**, **DESVIACIO_TOS**, **RESIDU_TOS**
int	**MAX_SEATS_IN_A_ROW**, **RESIDU_SEATS_IN_A_ROW**
int	**MIN_DISTANCE_SAME_CLIENT**, **MIN_TOS_PARTITION**
bool	**activada_max_seats_in_a_row**
bool	**never_leave_one_ticket_alone**
bool	**same_client_dispersion**, **same_client_dispersion_inblock**
bool	**split_tos**, **sparsity**
bool	**filtrar_tipus**, **filtra_type**, **filtra_price_type**, **filtra_sector**
DataTable *	**Relaxacions**
ArrayList *	**nTypeList**, **nPriceTypeList**, **nSectorList**

Com es veu conté un booleà per indicar si està activada o no cadascuna de les regles, així com tots els paràmetres de les regles que en necessiten.

La llista de relaxacions s'emmagatzema en la taula anomenada Relaxacions, aquesta taula és de la forma:

Regla	Acció	Paràmetre	Quantitat	Importància
Màxim seients per fila	Decrementar	màxim	2	40
Dispersió	Desactivar			

Les llistes de tipus (nTypeList, nPriceTypeList, nTicketProductTypeList, nSectorList i nTeamList), ens indiquen per cada classe de tipus quants tiquets falten per assignar. Per exemple la llista nTypeList podria ser:

Tipus	Quantitat total restant
Tipus 1	40
Tipus 2	20

Quan s'assigna una ordre d'un tipus es decrementa la quantitat total restant amb el nombre de tiquets de l'ordre. En el moment que el comptador d'algun dels tipus arribi a zero els seients marcats amb aquest tipus podran ser ocupats per tiquets d'ordres sense cap tipus especificat.

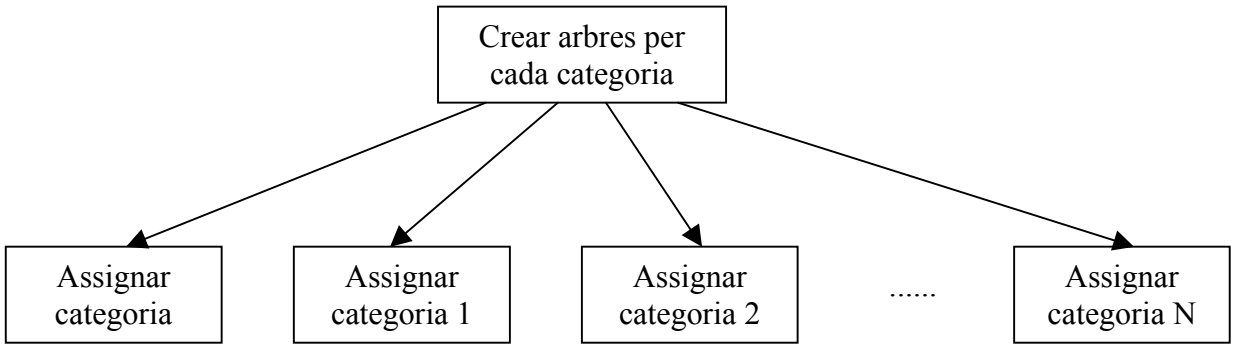
4.5.6.2 Mètodes principals

bool	**Relaxa_una_regla** (int n)

Aquest és el mètode més important de la classe. La seva funció és realitzar la relaxació *n*. Segons l'algorisme d'assignació *n* comença valent 1. En el moment que no és possible assignar una ordre es crida aquest mètode per efectuar la primera relaxació. El mètode cercarà a la taula de relaxacions quina és l'acció a emprendre per la primera relaxació i la portarà a terme. Un cop s'ha relaxat s'incrementarà *n* per si és necessari tornar a relaxar. En el moment que s'assigni l'ordre o es deixi sense assignar n tornarà a valer 1 per intentar assignar sempre les ordres amb les regles inicials.

4.6 Algorisme

L'algorisme d'execució seguirà els passos que s'han explicat a la secció de disseny. Com s'ha justificat en aquella secció podem dividir l'algorisme en tantes fases com categories haguem de considerar.

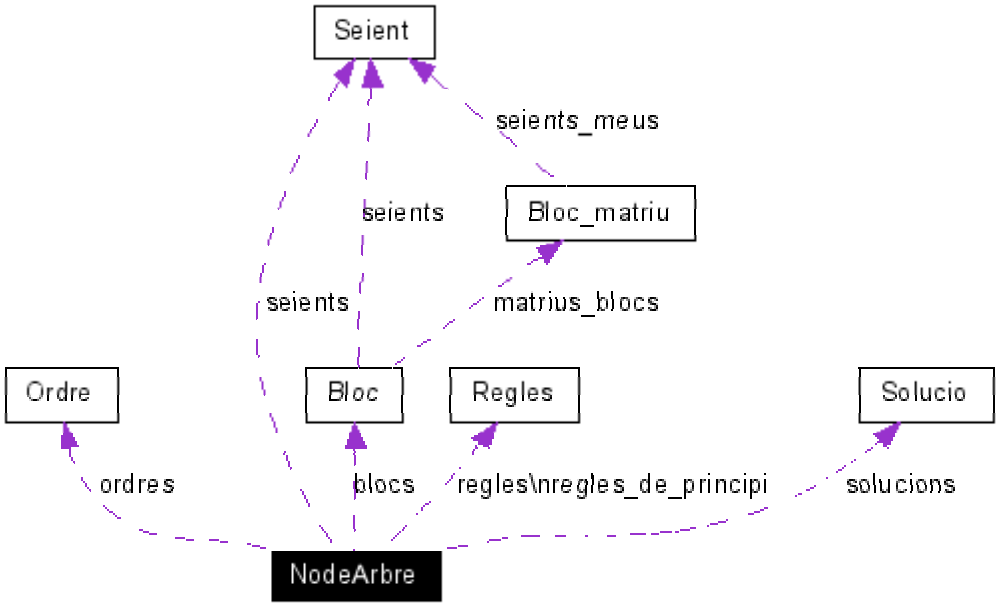


La tasca de crear arbres per cada categoria ha de crear tants arbres com categories i executar-los en paral·lel. Això es farà mitjançant threads. El procés principal crearà un thread per cada categoria i li passarà les seves dades per què aquest pugui executar-se. Cada thread assignarà els tiquets de la seva respectiva categoria. Quan els threads vagin finalitzant les seves execucions passaran el resultat de l'assignació al procés principal que serà l'encarregat de fusionar els resultats de tots.

4.6.1 Classe NodeArbre

S'ha creat una classe anomenada NodeArbre que és la que cada thread executarà. Aquesta classe és l'encarregada de dur a terme el procés d'assignació d'una categoria. Tindrà tota la informació referent a la categoria que està processant i en generarà la solució.

4.6.1.1 Diagrama de col·laboració



4.6.1.2 Atributs principals

Ordre *	**ordres**
Seient *	**seients**
Bloc *	**blocs**
Solucio *	**solucions**
Regles *	**regles**
double	**fitness_total_cat**

- Ordres, seients, blocs

Cada NodeArbre ha d'assignar una categoria. Aquestes taules contindran tota la informació referent a la categoria.

- Solucions

En aquesta taula s'hi introduirà la solució trobada pel procés d'assignació d'aquesta categoria.

- Regles

Objecte amb les regles que s'apliquen a la categoria. Totes les categories comencen amb regle idèntiques però cada categoria anirà realitzant relaxacions en funció de les seves necessitats. Per tana no hi pot haver un únic objecte compartit sinó que cada categoria tindrà el seu, tot i que inicialment tots seran còpies de les regles inicials fixades a la configuració.

- Fitness_total_cat

Fitness global de la categoria.

4.6.1.3 Mètodes principals

void	**proces** (int s_ha_relaxat)
void	**Genera_solucio** ()
void	**marcar_ordres** (DataRow *ordre, int motiu, DataRow *llavor)
void	**desmarcar_ordres** ()
int	**parteix_TOS** (DataRow *&ordre, int &index_real, **Ordre** *&**ordres**, bool escriu)
void	**BACKTRACKING** ()
double	**Calcula_Global_Fitness** (String ***categoria**, **Solucio** *sol, **Ordre** ***ordres**, **Bloc** ***blocs**)
void	**marquem_seients_assignats_com_a_prohibits_per_llavor** (int i_bloc, ArrayList *lMinimsAssignats)
void	**afegeix_a_prohibits_per_sempre** (int i_bloc, ArrayList *lSeientsProhibits)
void	**inicialitza_comptadors_tipus** ()

- procés

Acció que genera la primera solució d'una categoria seguint tots els passos de l'esquema general que es mostra i s'explica a l'apartat 3.3.5 del capítol de disseny.

- Genera_solucio

Funció que recorre tots els blocs creant la taula de solucions de la categoria.

- marcar_ordres

Marca ordres de les mateixes característiques que una donada. Les característiques s'examinaran les defineix el paràmetre motiu.

- desmarcar_ordres

Treu totes les marques posades per la funció anterior.

- parteix_TOS

Funció que aplica la regla de partir TOS.

- BACKTRACKING

Funció que realitza el backtracking. Inclou tant el backtracking per les ordres no assignades com el de millorar els fitness dels blocs

- Calcula_global_fitness

Calcula tots els valors definits a la funció de fitness (apartat 3.4) d'una solució.

- marcar_seients_assignats_com_a_prohibits_per_llavor,afegir_a_prohibits_per_sempre

Funcions per prohibir seients. Cada bloc té una llista de prohibits per llavor i una altra de prohibits per sempre.

- inicialitza_comptadors_tipus

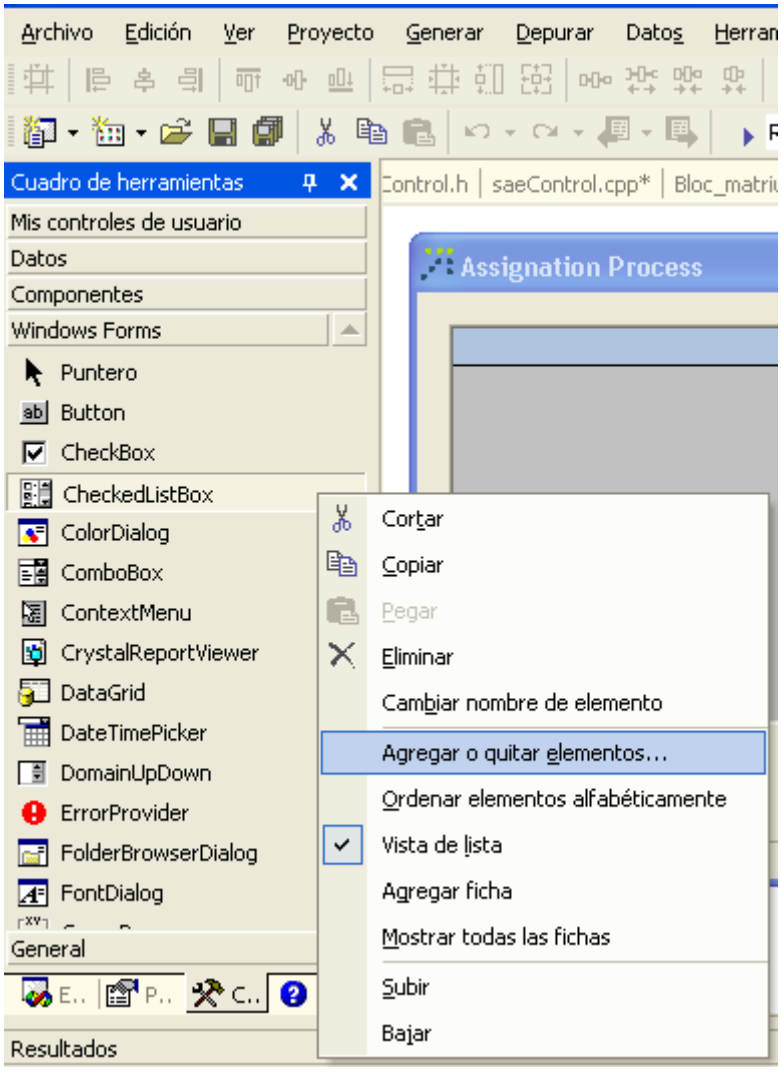
Crea les llistes de tipus. Aquesta llista es troba a l'objecte regles. Les llistes s'actualitzen cada vegada que s'assigna una ordre.

4.7 Integració

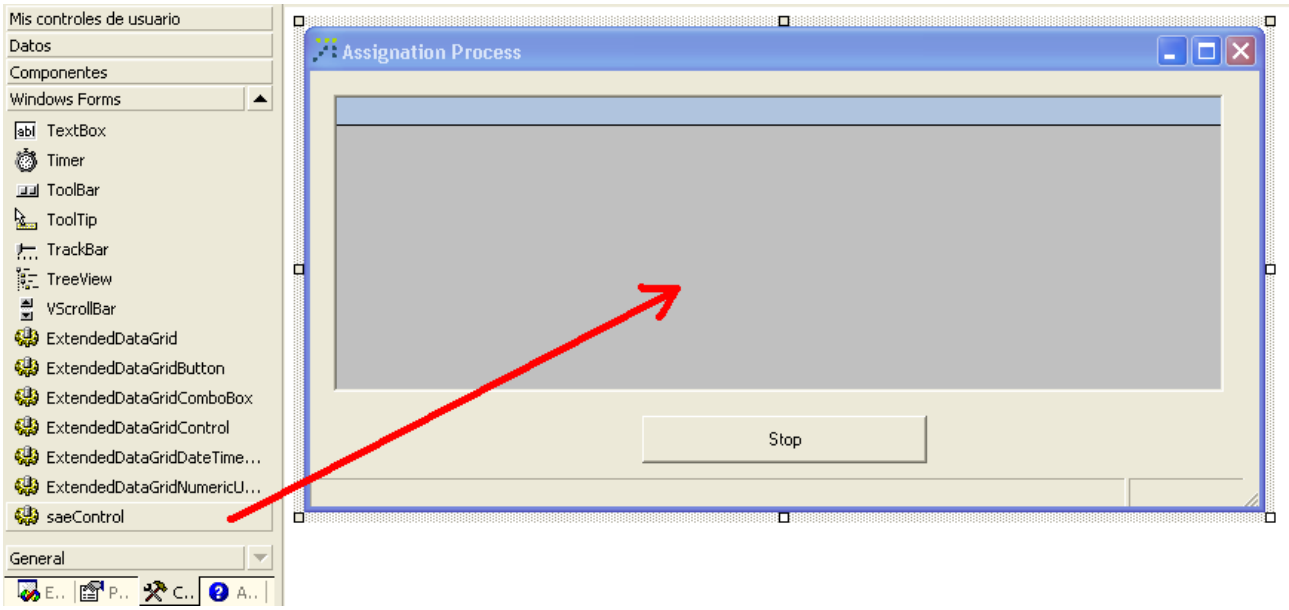
4.7.1 Llibreria ade.dll

Aquesta llibreria és un control que es pot afegir a qualsevol formulari d'una aplicació de Visual Studio .NET.

Per afegir-la cal anar al cuadro d'herramientas, clicar amb el botó dret i seleccionar la opció Agregar o quitar elementos..., seguidament apareix una finestra per seleccionar la llibreria a afegir. Amb el botó examinar... es localitza l'arxiu sae.dll i amb això s'afegirà el control a la llista.



Per col·locar el control al formulari s'ha de seleccionar el control saeControl i clicar en el formulari a la posició on s'hi vulgui inserir.



Això genera un codi similar a aquest:

```
private: sae::saeControl *  saeControl1;

void InitializeComponent(void)
{
        ...
        this->saeControl1 = new sae::saeControl();
        //
        // saeControl1
        //
        this->saeControl1->Location = System::Drawing::Point(0, 0);
        this->saeControl1->Name = S"saeControl1";
        this->saeControl1->Size = System::Drawing::Size(640, 296);
        this->saeControl1->TabIndex = 0;
        ...
}
```

4.7.2 Crida a la llibreria

La crida per iniciar el procés d'assignació es fa mitjançant un funció on els seus paràmetres són tot el que s'ha configurat amb l'aplicació.

```
saeControl1->Start_Assignation(...)
```

La funció té aquesta firma:

```
void Start_Assignation(
System::String *ftxBlocs,
System::String *ftxSeients,
System::String *ftxOrdres,
System::String *ftxMarket,
System::String *ftxCustomer,
System::String *ftxGroup,
System::String *ftxCategories,
System::String *ftxCompatibility,
System::String *outputDir,
DataTable *dtCategories,
DataTable *Regles,
DataTable *Relaxacions,
int numSolucions,
int StopSolucions,
int StopTemps,
int StopFitness,
int mode,
int backtracking,
double XX,
double YY,
double PTOSR,
double PTOSJ,
double PBTOS,
double PBD,
double PGBD,
double PGDESV,
double PGTOS,
double PGD,
double PU,
int assignation_order
);
```

- ftxBlocs: String amb el nom del fitxer xml de blocs (amb la ruta). Per exemple: "c:\blocs.xml".

- ftxSeients: String amb el nom del fitxer xml de seients (amb la ruta). Per exemple: "c:\seients.xml".

- ftxOrdres: String amb el nom del fitxer xml d'ordres (amb la ruta). Per exemple: "c:\ordres.xml".

- ftxMarket: String amb el nom del fitxer xml de Market segment (amb la ruta). Per exemple: "c:\market.xml".

- ftxCustomer: String amb el nom del fitxer xml de Customer groups (amb la ruta). Per exemple: "c:\customer.xml".

- ftxGroup: String amb el nom del fitxer xml dels Group Sales Customer (amb la ruta). Per exemple: "c:\group.xml".

- ftxCategories: String amb el nom del fitxer xml de categories (amb la ruta). Per exemple: "c:\categories.xml".

- ftxCompatibility: String amb el nom del fitxer xml de la matriu de compatibilitat (amb la ruta). Per exemple: "c:\compatibility.xml".

- outputDir: String amb el nom del directori a on es desaran els fitxers de sortida. Per exemple: "c:\sortida".

- dtCategories: DataTable de categories.

Enabled	Name

  Enabled: Booleà que indica si la categoria s'ha d'assignar o no. Pot valer "True" o "False".
  Categoria: Nom de la categoria (VIP, Category 1, etc)

- Regles: DataTable amb totes regles

Id	Descripció	Active	Param 1	Param 2	Param 3

  Id: Identificador de la regla. Enter corresponent al número de regla (veure caràleg de requeriments)
  Descripció: Nom de la regla. String amb una descripció de la regla (exemple: Split ticket order group)
  Active: Booleà per indicar si la regla s'aplicarà durant l'assignació. Pot valdre true o false.
  Param 1: Enter amb el valor del paràmetre 1 de la regla. Null en cas de no existir.
  Param 2: Enter amb el valor del paràmetre 2 de la regla. Null en cas de no existir.
  Param 3: Enter amb el valor del paràmetre 3 de la regla. Null en cas de no existir.

- Relaxacions: DataTable amb una llista d'accions sobre regles ordenades per prioritat.

Regla	Action	Parameter	By

  Regla: Identificador de la regla.
  Action: Acció a realitzar amb la regla. Pot valer Activate, Desactivate, Increment, Decrement, Set to.
  Parameter: Paràmetre de la regla que es vol relaxar.
  By: Valor de l'acció que es vol realitzar.

- numSolucions: enter amb el número de columnes de solucions que es vol veure a la taula de solucions. Ha de ser més gran o igual que 1.

- StopSolucions: enter amb el número de solucions màxim que es vol trobar. Un cop s'hagin trobat aquest nombre de solucions, l'algorisme s'aturarà. Si no es vol que l'algorisme tingui aquesta possibilitat d'aturada aquest paràmetre haurà de valer -1.

- StopTemps: enter amb el temps en minuts que es vol que l'algorisme s'estigui executant. Un cop hagi passat aquest temps l'algorisme s'aturarà. Si no es vol que l'algorisme tingui aquesta possibilitat d'aturada valdrà -1.

- StopFitness: enter amb el valor de fitness que es vol trobar. Un cop s'hagi assolit aquest valor de fitness amb alguna solució, l'algorisme s'aturarà. Si no es vol que l'algorisme tingui aquesta possibilitat d'aturada valdrà -1.

- Mode: Mode d'execució de l'algorisme. És un enter que pot valer:

  1 – Assignar totes les ordres
  2 – No assignar les ordres que no compleixin les regles sense crear fitxer d'ordres no assignades.
  3 – No assignar les ordres que no compleixin les regles creant fitxer d'ordres no assignades.
  4 – Relaxar automàticament regles per poder assignar les ordres sense crear fitxer de log amb les relaxacions
  5 – Relaxar automàticament regles per poder assignar les ordres creant fitxer de log amb les relaxacions

- Backtracking: Mode d'execució del backtracking. És un enter que pot tenir els valors:

  0 – No fer backtracking
  1 – Fer backtracking interactiu
  2 – Fer backtracking automàtic

- Variables per al càlcul del fitness:

  - XX: Constant que indica la distància entre dos seients de costat
  - YY: Constant que indica la distància entre dos seients sobreposats
  - PTOSR: Pes del TOS Ranking fitness
  - PTOSJ: Pes del TOS Joint fitness
  - PBTOS: Pes del bloc TOS fitness
  - PBD: Pes del block dispersion fitness
  - PGDESV: Pes del global desviation
  - PGTOS: Pes del global TOS fitness
  - PGD: Pes del global dispersion fitness
  - PU: Pes per la penalització dels tiquets individuals no assignats

- Assignation_order: Ordre d'assignació de les ordres. És un enter que pot valdre:

  1 – S'assignen primer les ordres de rank més alt
  2 – S'assignen primer les ordres de mida (Number_of_tickets) més alt

5 Resultats

5.1 Introducció

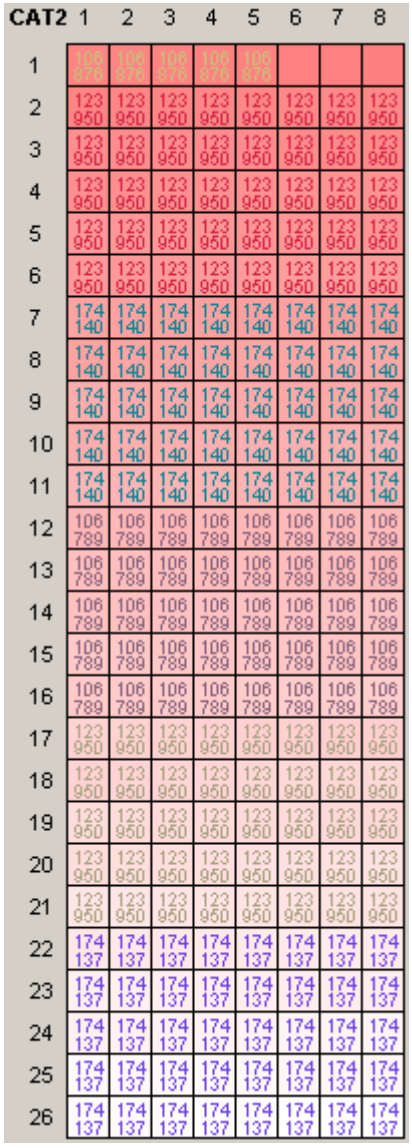
En aquest capítol s'analitzaran els resultats obtinguts amb l'algorisme implementat.

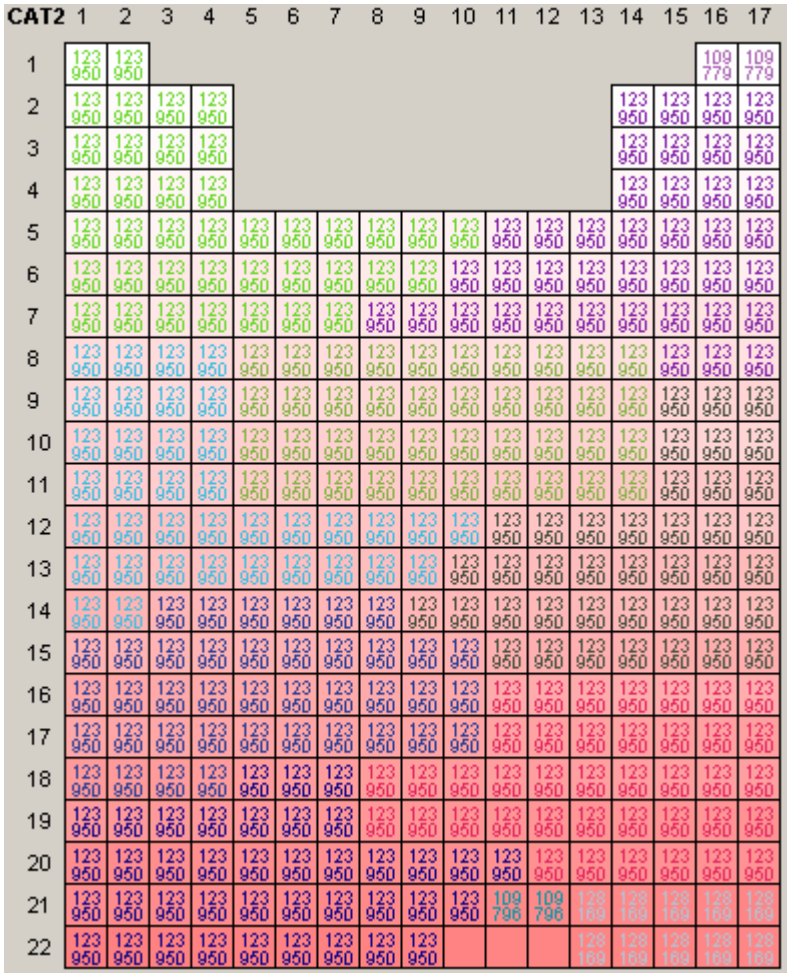
Les proves realitzades s'han fet amb el circuit de Catalunya de Montmeló i amb la informació sobre els seients, blocs i tiquets venuts a l'any 2003. La màquina utilitzada ha estat un Pentium 4 a 2.6 GHz, amb 768 Mb de RAM, 30 Gb de disc dur i S.O. Windows XP.

Com s'explica al capítol de disseny l'algorisme consta de dues fases: cerca de la primera solució i backtracking. Primer es mostraran els resultats de la cerca de primeres solucions amb diferents situacions (blocs, ordres, regles a complir) i seguidament es veurà com treballa el mecanisme de backtracking per millorar les solucions.

5.2 Primeres solucions

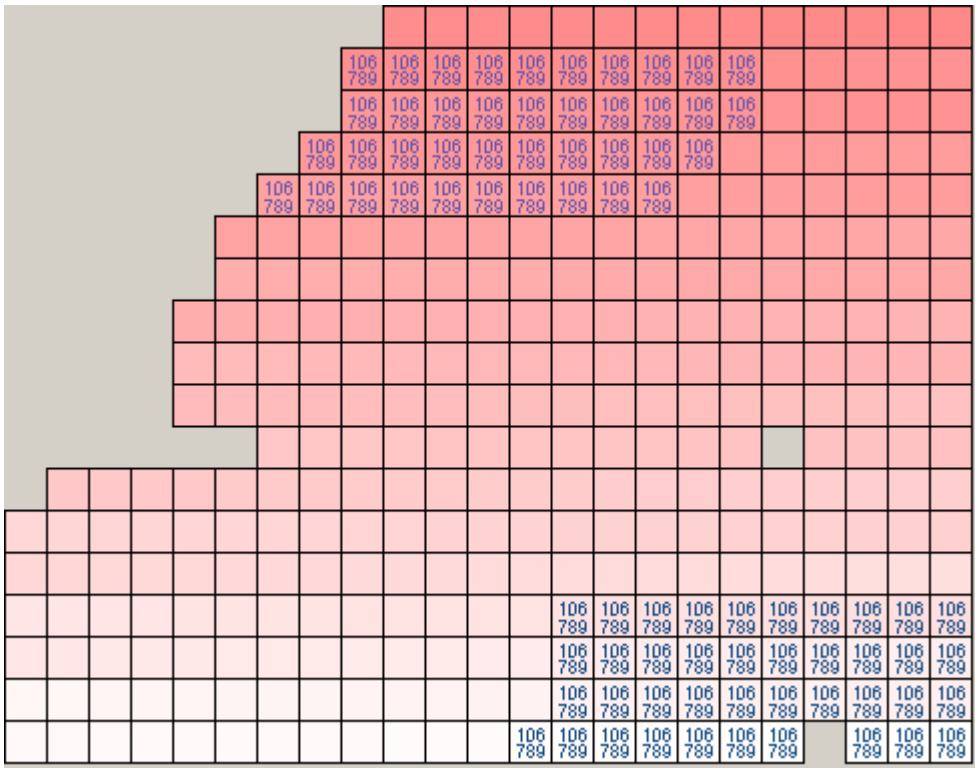
Aquí podem veure alguns exemples de blocs assignats. Pel que fa a les regles, s'ha seleccionat una mida d'ordres de 40, un màxim de seients per fila d'una ordre de 10, no es permet deixar tiquets d'una ordre en una fila sols i no cal fer dispersió de les ordres d'un mateix client.

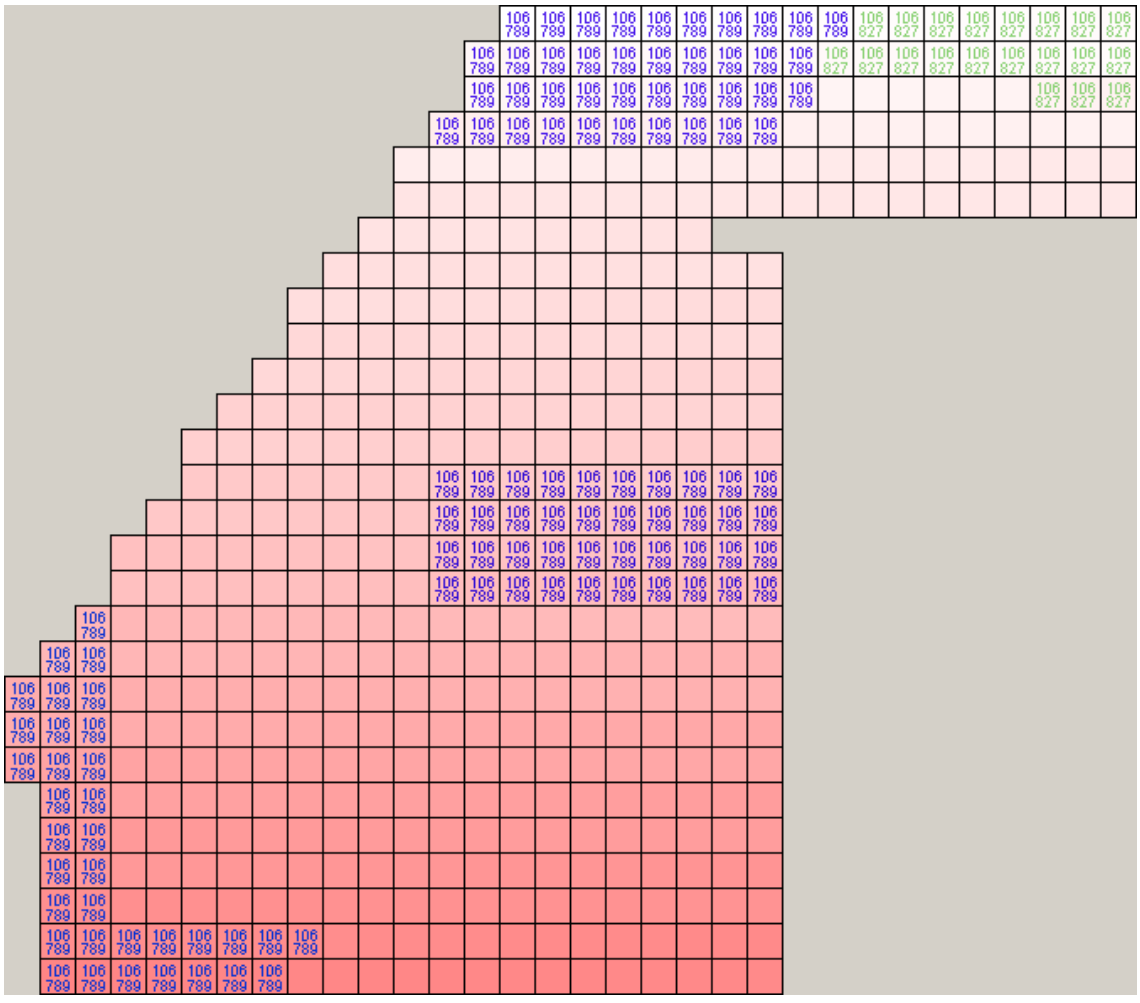




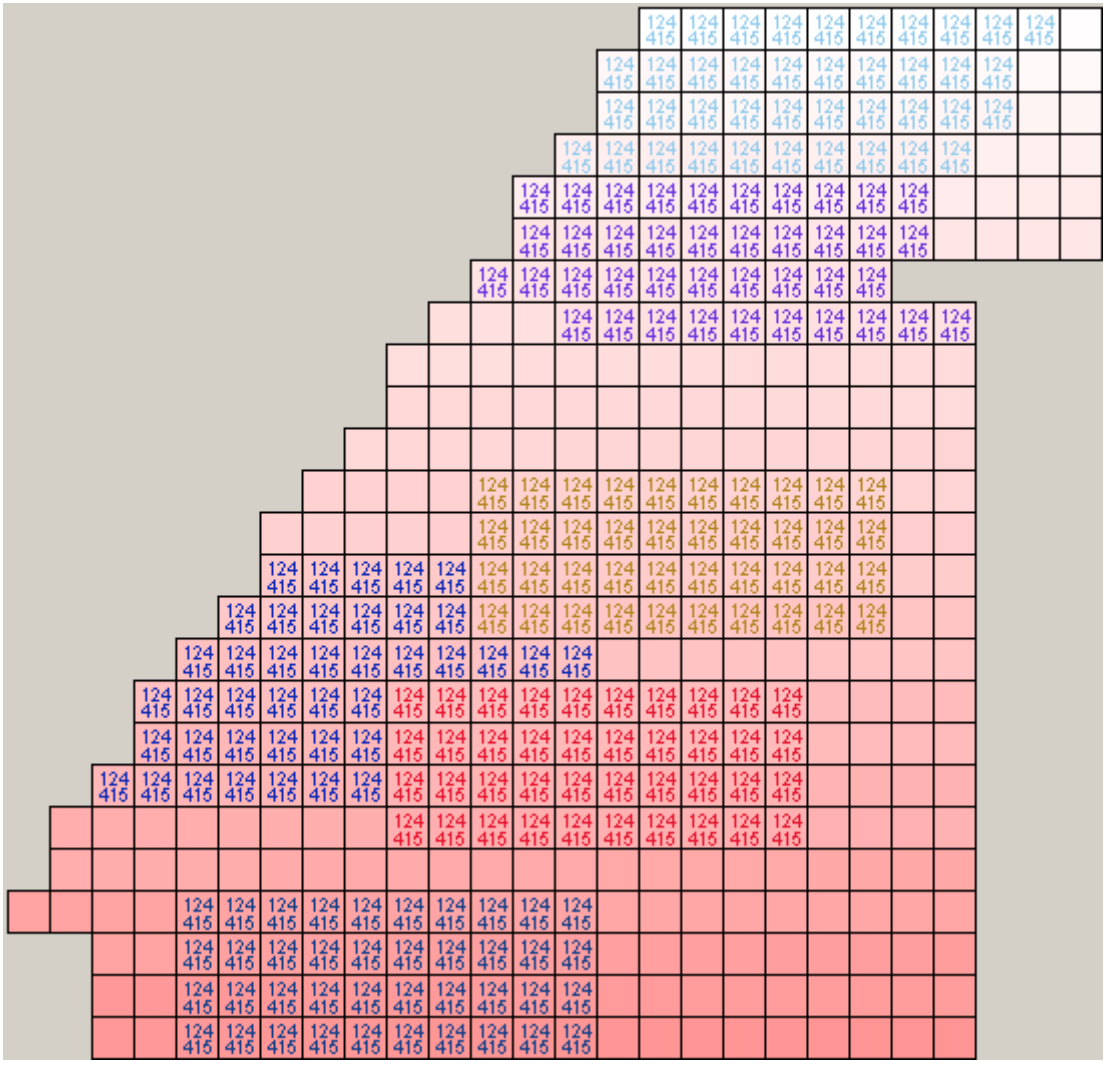
Nota: En el conjunt d'ordres d'entrada hi ha alguns requests de mida inferior a 40 que no s'han hagut de partir (requests 106876, 109779, 109796 i 128169).

Exemples amb la regla de dispersar ordres del mateix client dins dun bloc activada. S'ha introduït una distància mínima entre ordres de 10 seients.





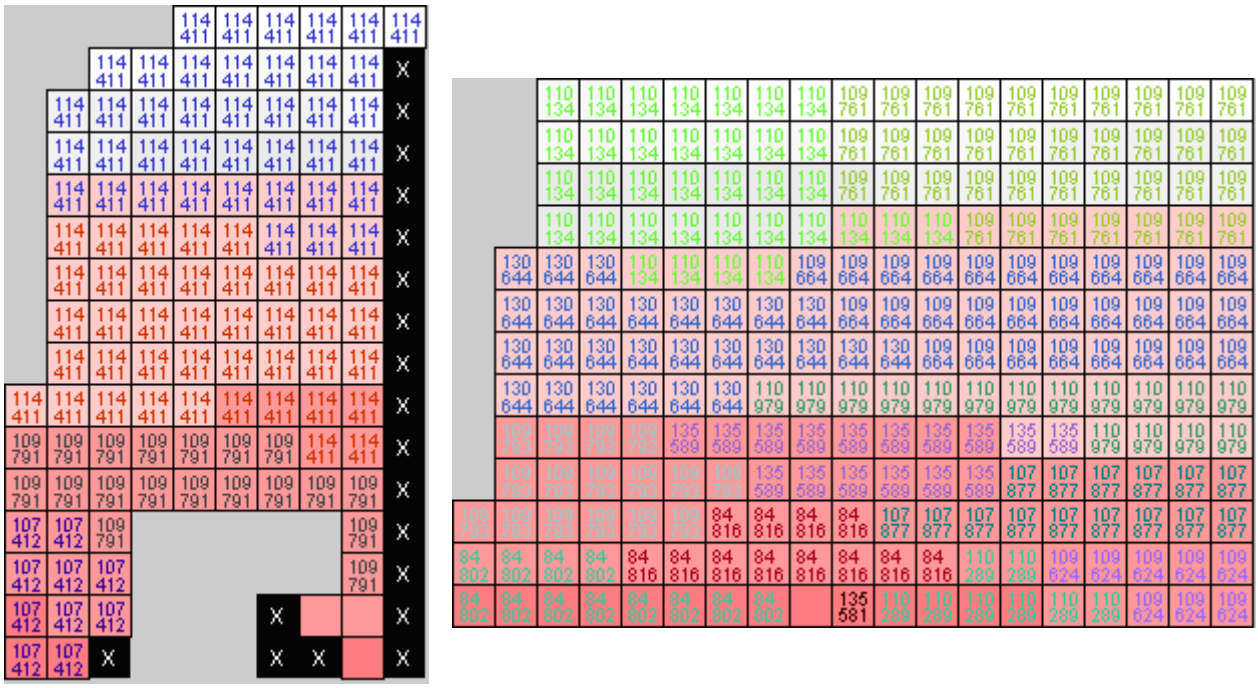
Exemples amb dispersió global de les ordres:



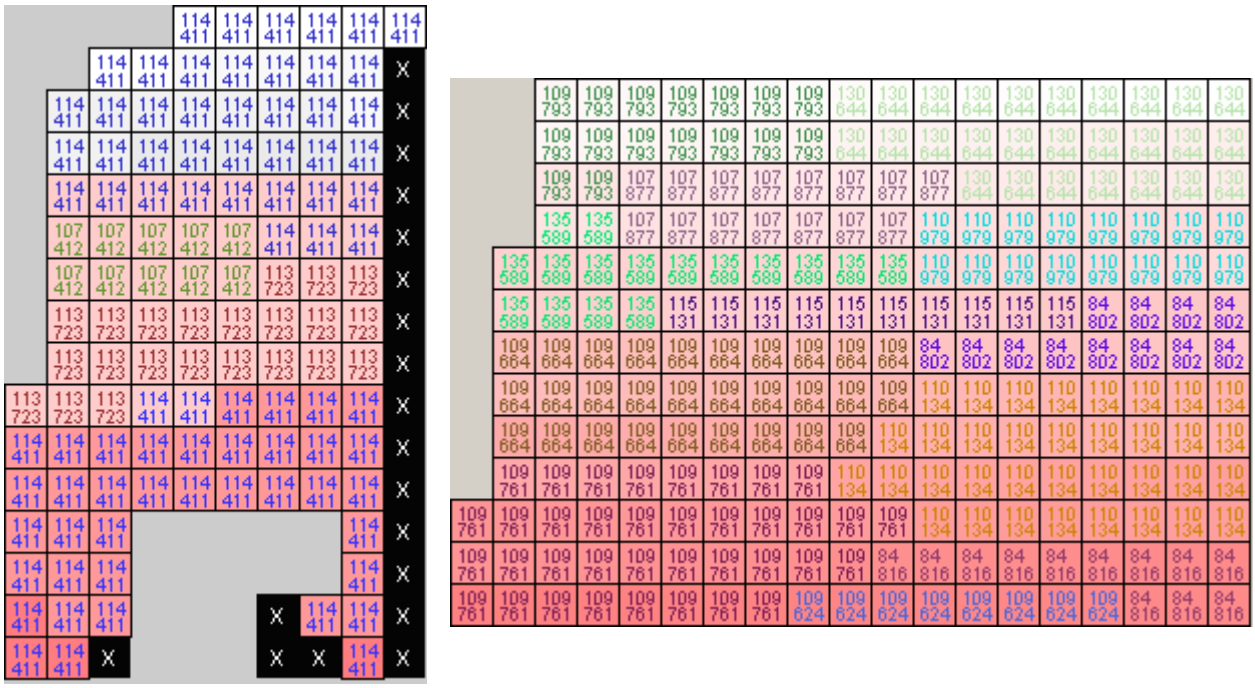
5.3 Backtracking

El backtracking és inclús més efectiu del que pensàvem a priori que seria. En la primera fase d'assignar les ordres no assignades vàries vegades ha estat capaç de trobar la manera d'assignar-les totes. En la segona fase de millorar els fitness dels blocs un a un, en pràcticament totes les execucions que hem fet s'ha aconseguit millorar el fitness d'un bon grapat de blocs.

Exemples:
Abans de fer backtracking: 4960 tiquets assignats. 40 tiquets no assignats. Fitness 12,28 %



Després del backtracking: 5000 tiquets assignats. 0 no assignats. Fitness 87,08 %



5.4 Temps d'execució

Diferenciarem dos temps d'execució. El temps de trobar la primera solució i el temps del backtracking. Els primers solen ser són molt inferiors en comparació amb els de backtracking; tots els circuits que s'han provat han finalitzat aquest primer procés en menys de 10 minuts. Cal esmentar que com més regles estiguin activades i més restrictives siguin les regles més lent és el procés, però sempre en magnituds similars.

En aquests exemples s'han activat la totalitat de les regles amb els paràmetres que habitualment s'utilitzen a la realitat.

Número total de tiquets a distribuir	Número total de seients disponibles	Temps transcorregut per trobar la primera solució	Fitness global de la solució
5000	5052	0 min. 23 seg.	12,28
13589	14202	3 min. 49 seg.	83,64 *
28599	32997	4 min. 8 seg.	89,05
33678	53673	3 min. 2 seg.	90,75 *
49804	56836	5 min. 39 seg.	87,35 *

* El circuit conté més d'una categoria. El fitness s'ha calculat fent la mitjana dels fitness de totes les categories

Podem observar un valor de fitness molt baix en el primer exemple. Això és degut a que no s'han pogut assignar totes les ordres. Com veiem aquests casos la funció de fitness els penalitza molt.

Una altre observació interessant és que en general els casos en que s'assignen totes les ordres assoleixen uns valors de fitness força alts. Aquest fet es deu a que com s'ha revelat al capítol de disseny, l'arbre de cerca està molt balancejat i amb el mètode de cerca de la primera solució utilitzat el resultat obtingut s'aproparà bastant a l'òptim.

El temps de càlcul del mecanisme de backtracing és mot més gran, però té la avantatja que es pot aturar en qualsevol moment donant com a resultat la millor solució trobada fins aquest moment.

En aquests exemples es mostra el temps total d'executar completament el procés de backtracking i la millora aconseguida en la solució.

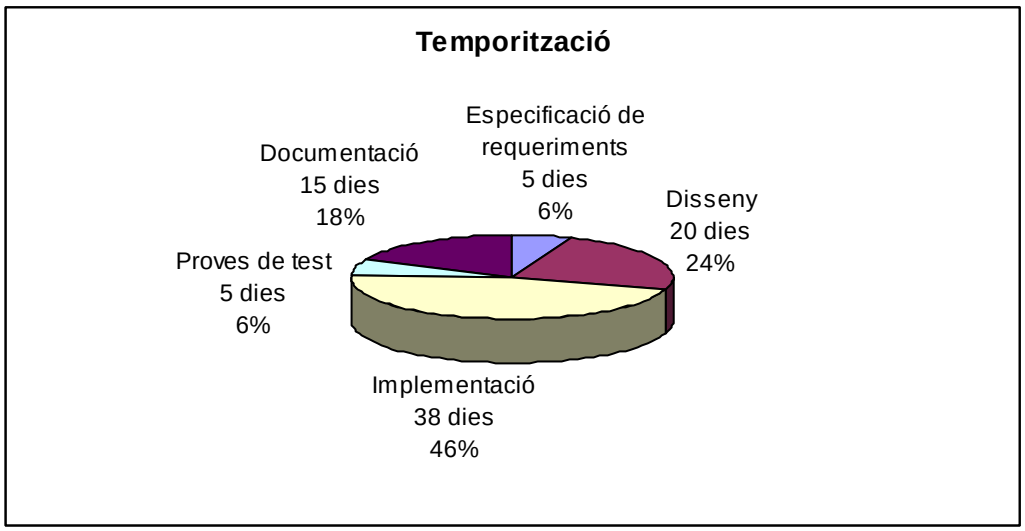
Número total de tiquets a distribuir	Número total de seients del circuit	Temps backtracking	Fitness anterior	Fitness final
5000	5052	29 min. 28 seg.	12,28	87,08
5403	14688	59 min. 27 seg.	91,03	91,52
10261	14864	1h. 10 min. 33 seg.	90,14	90,54

Aquí observem com el backtracking millora espectacularment el fitness quan aconsegueix assignar totes les ordres. En canvi, quan la primera solució trobada ja es prou bona li és molt més difícil de millorar-la substancialment.

5.5 Temporització

En aquest apartat es mostrarà el temps dedicat a cadascuna de les fases d'aquest projecte de fi de carrera.

La data d'inici del projecte pel que fa al desenvolupament de l'aplicació és el 03/01/2005 i la data de finalització el 17/06/2005. La dedicació ha estat d'unes 31 hores setmanals. Aquí podem veure un gràfic dels percentatges de temps invertits en cadascuna de les fases principals del projecte, comptant que cada dia equival a unes 6 hores de treball:



6 Conclusions i treballs futurs

6.1 Conclusions

La Fórmula 1 és un esdeveniment esportiu al que hi acudeixen grans multituds de persones. Els espectadors quan compren una entrada aquesta no té assignada un seient físic del circuit. Per tant, una de les tasques dels responsables de la organització d'aquests esdeveniments consisteix en la distribució de les persones per les diferents localitats disponibles en el circuit.

En aquest projecte s'ha creat un algorisme que resol el problema de la distribució dels espectadors. L'algorisme assigna un conjunt de tiquets als seients d'un circuit concret de forma efectiva, seguint i complint els criteris i normes de la FIA (Federació Internacional de l'Automòbil).

El problema de la distribució d'espectadors es representa com un problema de satisfacció de restriccions (CSP), en el qual cada assignació representa una o vàries decisions a prendre, essent el conjunt de decisions preses el resultat del procés. Donat que aquest és un problema de la classe NP, l'avaluació de cada resultat es podrà fer en temps polinòmic però per descobrir quin de tots els possibles resultats és el millor s'hauria de realitzar una cerca exhaustiva en l'espai de solucions, fet inviable degut al desmesurat nombre de possibles solucions que hi ha.

Per no haver d'explorar tot l'espai d'estats s'han utilitzat una sèrie d'heurístiques que suggereixen quina decisió prendre en cada pas del procés per tal d'obtenir una primera solució prou bona el més ràpid possible.

Una d'aquestes heurístiques es basa en realitzar la distribució de les ordres (conjunts de tiquets) mitjançant la tècnica del *region growing* consistent en la selecció de llavors i el posterior creixement d'aquestes fins a acumular tots els tiquets de l'ordre. Amb aquesta tècnica es construeix una branca que sinó arriba a la solució òptima, en troba una de molt propera.

Una altra heurística s'ha exposat per resoldre el problema de la dispersió de les ordres per tots els blocs. La funció escollida ha estat distribuir les ordres equitativament pels blocs i aleatòriament dins dels blocs, amb uns resultats molt satisfactoris i millors en termes d'uniformitat global que altres més matemàtics, però per contra més deterministes.

Per tal de millorar la primera solució obtinguda s'ha elaborat un mecanisme de backtracking amb l'objectiu de modificar la solució augmentant el seu valor de la funció de fitness. S'han creat dues estratègies de backtracking diferents en funció de si la primera solució havia deixat ordres sense assignar o no. En el primer cas s'enfoca el backtracking a intentar assignar el màxim nombre possible de tiquets. En canvi, pel segon cas l'objectiu és modificar la distribució de les ordres assignades a cada bloc aconseguint millorar els fitness individuals dels blocs.

Els resultats finals obtinguts amb l'algorisme implementat demostren que els mètodes escollits són adients per resoldre aquest problema ja que les assignacions sempre compleixen les restriccions obligatòries, es generen en un temps raonable i indiscutiblement es millora en tots els aspectes qualsevol solució assolida a mà.

6.2 Treballs futurs

6.2.1 Noves funcionalitats

- Recentment s'està parlant en l'àmbit de la Fórmula 1 de crear un nou tipus de tiquets de venta al públic amb la característica de que permetran seguir la trajectòria d'una escuderia concreta en totes o un subconjunt de les curses del calendari. Això vol dir que s'hauran de reservar en tots els circuits tants seients com tiquets d'aquest tipus s'hagin venut. Per incorporar aquesta nova funcionalitat a l'algorisme s'hauria de crear un nou atribut tant a les ordres com als seients i tractar-los d'una manera similar a com es fa amb els altres tipus existents (status, type, price type i sector).

- Una altra característica que volen afegir a l'algorisme és una nova regla que obligui a assignar les ordres en formes estrictament rectangulars. Per exemple si l'ordre és de 40 tiquets, que s'assignin en 4 files de 10 tiquets cadascuna, o 5 files de 8 tiquets. Tot amb uns màxims i minims de files i de tiquets per fila. Per realitzar aquesta funcionalitat s'hauria de modificar el mòdul d'assignació amb un mètode que assignés amb aquesta forma específica, i probablement també caldria modificar el mòdul de buscar llavor per retornar només llavors on s'hi pogués assignar d'aquesta manera.

- Als organitzadors els aniria bé el fet de que el procés de backtracking tingués l'opció de ser més interactiu en el sentit de que l'usuari pogués intervenir més activament en el procés. L'algorisme hauria de mostrar per pantalla les accions que va realitzant, i donar opcions de modificació manual a l'usuari.

6.2.2 Millores

- Quan s'assigna amb dispersió de les ordres per tots els blocs, es fa repartint equitativament les ordres per tots els blocs i assignant-les dins del bloc en llavors aleatòries. Potser es podria intentar crear un mètode no aleatori per assignar-les dins del bloc dispersament, però sense donar distribucions massa semblants en tots els blocs.

- El procés de backtracking millora la solució de les ordres assignades bloc a bloc, és a dir, desassignant i assignant un bloc un cert nombre de vegades i quedant-se amb aquella assignació amb valor més alt de fitness. Potser es pot millorar més la solució desassignant més d'un bloc a la vegada o determinades ordres de diferents blocs.

- El backtracking reassignatiu es fa ordenant aleatòriament les ordres perquè una cerca exhaustiva de totes les ordenacions possibles és intractable. Potser es pot trobar un mètode per ordenar les ordres amb algun criteri que doni certa confiança que el resultat obtingut sigui prou bo. O realitzar un conjunt de proves que puguem tenir la certesa de que conté la ordenació òptima.

7 Referències

[Aamodt & Plaza 94] "Case-based Reasoning : Foundational Issue s, Methodological Variations, and system approaches". Aamodt A. And Plaza, E. Artificial Intelligence Communications, IOS Press, Vol : 1, pp: 39-59, 1994.

[Kumar 1992] Algorithms for Constraint Satisfaction Problems: A Survey. Vipin Kumar. AI Magazione. 1992

[Louis 1993] Genetic Algorithms as a Computational Tool for Design, by Sushil J. Louis. August 1993.
URL: http://www.cs.unr.edu/~sushil/papers/thesis.ps

[Reeves 1993] Modern heuristic techniques for combinatorial problems. C.R. Reeves, Blackwell Scientific Publications, Oxford. 1993.

[Rich 1991] Expert Systems in Artificial Intelligence. Rich, Elaine, and Kevin Knight. 1991. El capítol 20 conté una introducció de 10 pàgines als sistemes experts.

[Tsang 1993] Foundations of Constraint Satisfaction Problems. Tsank, E.P.K. Academic Press, London and San Diego. 1993.

[Zucker 1976] Region Growing: Childhood And Adolescence. Zucker, S.W. pp 382-399. Journal CGIP. 1976