UAB
Universitat Autònoma de Barcelona

Sistema de subhastes cooperatives per a l'assignació dinàmica de tasques a robots mòbils

Jonatan Trullàs Ledesma

UAB
Universitat Autònoma
de Barcelona

Departament de Microelectrònica i Sistemes Electrònics

Sistema de subhastes cooperatives per a l'assignació dinàmica de tasques a robots mòbils

Tesi en el programa de Doctorat en Informàtica

Jonatan Trullàs Ledesma

Director:
Prof. Lluís Ribas Xirgo
Tutor:
Prof. Eduardo César Galobardes
Universitat Autònoma de Barcelona (UAB)
Escola d'Enginyeria

Abril 2024

Resum

Grups de robots més o menys nombrosos s'encarreguen del transport intern dins dels magatzems automatitzats. Un dels grans reptes en la gestió d'aquests tipus de magatzem és l'assignació de tasques als robots a l'hora que es minimitza el cost global de les operacions. En entorns amb pocs punts de càrrega és possible fer una assignació òptima mitjançant tècniques de programació lineal. Però, a mesura que el nombre de tasques i robots augmenta, i que les tasques entren al sistema al llarg del temps de forma dinàmica, ja no és possible fer l'assignació òptima mitjançant la programació lineal en un temps raonable. Les solucions basades en subhastes solucionen el problema de l'assignació de tasques aproximant-se a la solució òptima i aportant robustesa i escalabilitat al sistema. El protocol d'assignació de tasques a robots que es fa servir és el denominat "contract-net", en què els agents que representen les tasques son les que inicien subhastes tancades en les què els robots fan les seves ofertes basades en el seu cost d'operació. A causa del dinamisme del sistema, l'aparició de noves tasques, possibles accidents i altres esdeveniments inesperats fan que les tasques llancin noves subhastes quan es produeix un d'aquests esdeveniments. Això fa que aquestes noves subhastes provoquin el canvi de robot assignat a la tasca. Tot i que aquest mecanisme de "resubhasta"ens acosta encara més a la solució òptima (fins un 15% pitjor que la solució de la programació lineal) les tasques no tenen en compte el fet que que els robots poden estar participant a la vegada en altres subhastes. Tenim doncs que les tasques tenen un comportament "egoista"(*greedy*) perquè s'assignen el robot que aporta el valor mínim a la seva subhasta sense considerar les altres. Per tant, les primeres subhastes que avaluen les ofertes s'associen els robots. En aquesta tesi es modifica el protocol de les subhastes de manera que els agents que representen les tasques prenen les decisions basant-se no només en les ofertes que els robots fan per a la seva subhasta sinó també en les ofertes que fan a les altres. A més d'aquest canvi en el protocol, en aquesta tesi també es proposa un algorisme d'assignació de tasques que es denomina *Munkres**, que està basat en el mètode hongarès i que aprofita assignacions prèvies òptimes amb l'objectiu de minimitzar les comunicacions i d'evitar bucles en reassignacions. Amb aquesta solució, el cost d'execució de les tasques i el cost de transport es redueix, cosa que porta a millores dels costos globals d'operació que s'acosten fins a un 90% de l'òptim.

Resumen

Dentro de los almacenes automatizados grupos de robots más o menos numerosos se encargan del transporte interno. Uno de los grandes retos en la gestión de este tipo de almacenes es la asignación de tareas a los robots a la vez que se minimiza el coste global de las operaciones. En entornos con pocos puntos de carga es posible realizar una asignación óptima mediante técnicas de programación lineal. Sin embargo, a medida que el número de tareas y robots aumenta, y que las tareas entran en el sistema a lo largo del tiempo de forma dinámica, ya no es posible realizar la asignación óptima mediante la programación lineal en un tiempo razonable. Las soluciones basadas en subastas solucionan el problema de la asignación de tareas aproximándose a la solución óptima y aportando robustez y escalabilidad al sistema. El protocolo de asignación de tareas a robots que se utiliza es el denominado "contract-net", donde los agentes que representan las tareas son los que inician subastas cerradas en las que los robots realizan sus ofertas basadas en su coste de operación. Debido al dinamismo del sistema, la aparición de nuevas tareas, posibles accidentes y otros acontecimientos inesperados hacen que las tareas lancen nuevas subastas cuando se produce uno de estos eventos. Esto provoca que estas nuevas subastas puedan provocar el cambio de robot asignado a la tarea. Aunque este mecanismo de "resubasta"nos acerca aún más a la solución óptima (hasta un 15% peor que la solución de la programación lineal) las tareas no tienen en cuenta el hecho de que los robots pueden estar participando a la vez en otras subastas. Tenemos pues que las tareas tienen un comportamiento "codicioso"("greedy") porque se asignan el robot que aporta el valor mínimo a su subasta sin considerar las demás subastas. Por tanto, las primeras subastas que evalúan las ofertas se asocian a los robots. En esta tesis se modifica el protocolo de las subastas por lo que los agentes que representan las tareas toman las decisiones basándose no sólo en las ofertas que los robots hacen para su subasta sino también en las ofertas que hacen a las demás. Además de este cambio en el protocolo, en esta tesis también se propone un algoritmo de asignación de tareas que se denomina *Munkres**, basado en el método húngaro y que aprovecha asignaciones previas óptimas con el objetivo de minimizar las comunicaciones y de evitar bucles en reasignaciones. Con esta solución, el coste de ejecución de las tareas y el coste de transporte se reduce, lo que lleva a mejoras de los costes globales de operación que se avecinan hasta un 90% del óptimo.

Abstract

Automated warehouses are populated with robots that take charge of the internal transportation. One of the main challenges of the management systems is to allocate tasks to robots while minimizing overall operation costs. For environments with a few loading spots, it is possible to make optimum assignments with linear programming techniques. As the number of tasks and robots increases, and as tasks arrive over time dynamically, it is no longer possible to make the optimal allocation using linear programming in a reasonable amount of time. Dynamic approaches based upon auctions solve the allocation problem with near-optimal solutions and give the systems robustness and scalability. We use the so-called "contract-net"protocol for assigning tasks to robots, in which the agents representing the tasks are the ones that initiate closed auctions, and the robots make their bids based on their cost of operation. Due to the dynamism of the system, the appearance of new tasks, potential crashes, and other unexpected events cause tasks to launch new auctions when one of these events occurs. This may cause the robot assigned to the task to change. Although this "re-auction"mechanism brings us even closer to the optimal solution (up to 15% worse than the linear programming solution) the tasks do not take into account the fact that robots may be participating at the same time in other auctions. We therefore have that the tasks have a "greedy"behavior because they are assigned the robot that contributes the minimum value to its auction without considering other auctions. Therefore, the first auctions that evaluate bids are assigned to robots. In this thesis the auction protocol is modified so that the agents representing the tasks make decisions based not only on the bids the robots make for their auction but also on the bids they make to the others. In addition to this change in the protocol, this thesis also proposes a task assignment algorithm called $Munkres^*$, which is based on the Hungarian method and which takes advantage of optimal prior assignments with the aim of minimizing communications and to avoid loops in reassignments. With this solution, the task execution cost and transportation cost are reduced, leading to overall operating cost improvements approaching 90% of optimal.

Agraïments

A la Glòria, per ser-hi sempre i per encoratjar-me en tots els moments d'aquest camí. També a la Mar i en Biel, que van arribar durant els anys de recerca, i ja els dec unes quantes estones d'estar junts. I, especialment, al Lluís, amic i director, per la seva dedicació.

Índex

Índex de figures

Índex de taules

Capítol 1

Introducció

Els sistemes multirobot (*Multi-Robot Systems, MRS*) s'utilitzen en l'automatització de magatzems[5] per encarregar-se del transport intern, tot aportant una capacitat de treball més elevada i més continuada que la dels sistemes no automatitzats. L'ús de múltiples robots fa que els MRS puguin ser tolerants a fallades d'algunes unitats. De fet, aquesta robustesa s'aconsegueix mitjançant el treball coordinat dels robots [14]. Aquest treball coordinat dels robots també permet variar-ne el nombre en actiu i, consegüentment, incrementar-lo segons les necessitats, és a dir, fer-lo escalable. Ara bé, la robustesa i l'escalabilitat de les solucions en dificulta la implementació. Entre els problemes a resoldre per aquest tipus de sistemes hi ha el d'assignació de tasques o MRTA *(Multi-Robot Task Allocation)*. Els *MRTA Solvers* estimen el cost associat a cada robot per dur a terme cada tasca i busquen l'associació entre robots i tasques de manera que es maximitzi el guany global [23]. Una tasca consisteix en recollir items dels punts de recollida i transportar-los las punts de descàrrega designats a la ordre de transport per tal de ser processats [26]. Pel que fa al càlcul del cost, tot i que les mètriques més habituals són el temps i la distància, se'n poden utilitzar d'altres com el consum d'energia, o una combinació d'aquestes.

Entre les solucions que tracten el problema del MRTA [34], les basades en mètodes d'optimització intenten trobar assignacions òptimes explorant l'espai de solucions. I hi ha una varietat de mètodes per fer-ho, inclosos els mètodes de programació lineal, l'algorisme hongarès [20], tècniques de cerca en arbres [2] i, fins i tot, assignacions basades en tècniques de joc cooperatiu [21].

Tanmateix, aquests mètodes requereixen un coneixement previ de totes les tasques i robots i no tenen en compte el dinamisme de l'entorn, l'arribada de noves tasques [14], la inclusió de nous robots o els esdeveniments imprevistos. Normalment aquest aspecte dinàmic es tracta repetint el procés d'assignació sempre que es produeix un esdeveniment rellevant dins el sistema [23]. Adicionalment, a mesura que el nombre de robots i la complexitat de les restriccions del problema augmenten, l'espai de solucions creix. Això fa que aquests mètodes d'optimització no siguin pràctics per aplicacions a gran escala (amb un gran nombre de robots i tasques) dins d'un temps de computació raonable [26, 2]. A més, les solucions basades en l'optimització tendeixen a ser centralitzades, cosa que dificulta l'explotació dels principals avantatges dels sistemes multirobot (MRS): robustesa i escalabilitat. Tenir un agent central responsable de l'assignació de tasques es converteix en un coll d'ampolla en cas de fallada, i

l'escalabilitat es veu limitada, ja que tots els agents han d'estar connectats a un agent central [19]. Els mètodes centralitzats esmentats anteriorment són difícils de descentralitzar i requereixen informació de tots els agents, fins i tot si la computació està descentralitzada [23].

Per això la majoria de solucions utilitzen mètodes aproximats que no garanteixen un resultat òptim globalment, però proporcionen solucions acceptables en un temps d'execució raonable [2]. A més, aquests mètodes són robusts i fàcilment escalables [1], i molts d'ells descentralitzats. No obstant això, com que una bona assignació local no contribueix necessàriament a una bona solució del sistema global, les solucions totalment descentralitzades poden produir solucions molt subòptimes [19].

Entre tots aquests enfocaments descentralitzats, els mètodes basats en subhastes es troben entre els més populars [19, 1]. Les subhastes proporcionen un mecanisme senzill per coordinar agents i assignar tasques en un sistema multiagent (MAS). Implica el procés d'assignació d'un recurs o servei a un conjunt de licitadors en funció dels criteris de l'agent licitador i del contingut de les ofertes dels licitadors. El disseny de subhasta més conegut, senzill i àmpliament utilitzat és el protocol contract-net (CNP) [35]. Aquest protocol defineix l'estructura d'alt nivell d'interaccions entre agents. En la seva versió inicial, el principal inconvenient del CNP és que es tradueix en assignacions "egoistes" *greedy* perquè els subhastadors intenten assignar les seves tasques al millor postor (representants de robots). En aquest cas, la solució pot ser òptima per a l'agent guanyador però no necessàriament per al grup.

Hi ha diverses estratègies per millorar la qualitat de la solució. Una d'elles és considerar que les tasques de transport es poden dividir en subtasques: la fase de recollida (l'agent viatja al punt de recollida) i la fase de transport (l'agent està transportant l'article). A partir d'alguns criteris, com ara l'arribada de noves tasques, es pot plantejar la reassignació de robots que es troben en fase de recollida, amb l'esperança que les reassignacions millorin la qualitat de la solució. Un altre mecanisme de millora de la qualitat és refinar les heurístiques o canviar el mètode de valoració de les ofertes. També es pot millorar la qualitat de la solució fent que els robots comparteixin informació en lloc de comportarse de manera egoista (*Greedy*). Això es pot aconseguir incorporant un "agent pissarra"o fent que els licitadors transmetin les seves ofertes (des de qualsevol subhasta) a tots els subhastadors. En aquest cas, els subhastadors tindrien informació sobre altres subhastes i podrien assignar tasques tenint en compte la solució global.

1.1 Motivació

Com hem vist anteriorment, entre les diverses estratègies per resoldre el problema del MRTA [34] el protocol CNP funciona bé tant en entorns estàtics com dinàmics, especialment quan l'ordre d'execució de les tasques no importa [22].

Per poder comparar les solucions basades en CNP enfront de la solució òptima, hem desenvolupat un framework per provar MRTA en diferents escenaris. Hem definit un cas concret consistent en un magatzem de mida mitjana operat amb robots. Els valors òptims es van obtenir mitjançant la programació lineal amb CPLEX [18] que només va produir resultats exactes per als casos de mida més petita i, a mesura que creixia el nombre de robots i tasques, el

temps d'execució va suposar una limitació per a l'obtenció de resultats òptims. De fet, vam observar que no era capaç de generar resultats en menys temps del que era d'esperar per completar les tasques. Per exemple, no va poder generar solucions òptimes en 30 minuts per a un conjunt de tasques el temps d'execució de les quals era aproximadament el mateix (de 30 minuts), tal com s'observava mitjançant la simulació del cas. La solució bàsica de CNP va generar una assignació de tasques en menys de 3 minuts per al mateix cas, que només era un 9% més costosa que l'assignació donada pel CPLEX.

Per intentar reduir els increments del cost global, les tasques s'han de reassignar [17] per compensar el fet que, degut a l'evolució del sistema, les assignacions anteriors ara han esdevingut més costoses. Recentment, s'ha proposat una tècnica de reassignació [45] adaptada als vehicles aeris no tripulats per tal que periòdicament es comprovi la idoneïtat de les seves missions (és a dir, les tasques assignades). En el nostre cas [27], les subhastes CNP es repeteixen quan es produeixen diversos esdeveniments, inclòs un event periòdic. A diferència de l'exemple anterior, aquí els subhastadors són les tasques.

La introducció de subhastes secundàries a donat lloc a millores en totes les mètriques (cost de solució, cost mitjà, durada mitjana de la tasca i temps total de solució). Per exemple, s'ha produït una millora de fins a un 8% en el cost mitjà de les tasques [27]. Així i tot, el MRTA basat en CNP implica una gran càrrega de comunicacions entre els agents. De fet, hi ha propostes per reduir-lo, per exemple utilitzant una pissarra compartida i solucions basades en l'optimització de colònies de formigues per a les subhastes [47].

Una altre estratègia per minimitzar l'impacte de les repeticions de subhasta en el cost de les comunicacions és reduïr la necessitat d'aquestes "resubhastes". Per als casos en què no s'han produït esdeveniments significatius (per exemple, tasques noves o robots que s'uneixen o surten del sistema), les repeticions de subhasta es requereixen per compensar les assignacions prèvies incorrectes. Això es podria millorar utilitzant un mètode d'optimització determinista com l'algorisme hongarès [20].

En aquesta tesi, a més del mètode de subhastes repetitives tipus CNP que ja havíem desenvolupat [27], s'utilitza l'algorisme hongarès per millorar l'assignació de tasques aprofitant les subhastes simultànies que tenen lloc al sistema. En aquest cas totes les tasques pendents es converteixen en subhastadores en cada nou esdeveniment del sistema.

La idea és aprofitar la informació de les subhastes simultànies (que es produeixen al mateix temps) per fer assignacions que tinguin en compte totes les tasques implicades (subhastes) i robots (ofertes). Per això, l'agent robot que rep sol·licituds de diverses subhastes simultànies envia les seves ofertes per a totes les subhastes a tots els subhastadors. D'aquesta manera, cada subhastador coneix la resta de subhastes i els valors aportats pels licitadors, i la solució es manté descentralitzada, encara que el MRTA s'executa tantes vegades com tasques implicades en el procés. Utilitzant el mateix framework d'experimentació i escenari que per al MRTA de resubhastes [27], compararem el mètode *greedy* enfront la utiltització d'un algorisme d'optimització determinista. Això permetrè veure com aquest canvi d'estratègia d'assignació afecta les comunicacions i el volum de missatges, així com la qualitat del servei.

Capítol 2

El problema MRTA

Al capítol 1 hem vist que els magatzems automatitzats utilitzen els sistemes multi-robot (MRS) per a les tasques de transport intern per aprofitar els avantatges d'aquests tipus de sistemes, com l'escalabilitat, la robustesa i la flexibilitat.

En aquestes solucions els robots han de treballar de manera coordinada i comunicant-se entre ells. Així, han d'aconseguir una millora del seu rendiment individual a la vegada que s'aconsegueix una millora del sistema global. I, degut a aquesta necessitat de maximitzar el rendiment (o minimitzar els costos), juntament amb la complexitat d'aquests tipus de sistemes, el problema de l'assignació de tasques és un dels principals focus d'estudi en l'àmbit dels MRS.

Això vol dir que, tenint un conjunt de robots i un conjunt de tasques a realitzar per aquests, com assignem les tasques a aquests robots de manera que el cost global de totes les operacions sigui mínim o bé dins d'uns valors raonables? Aquest problema es coneix com a MRTA (*Multi-Robot Task Allocation*).

2.1 Descripció i formalització

Aquesta recerca està centrada en el cas dels magatzems automatitzats. En concret en un entorn on els robots tenen una zona d'"aparcament"on romanen mentre no estan operatius, uns punts de recollida on han de carregar els productes, i els punts de descàrrega on aquests s'emmagatzemen o es processen. L'arribada de tasques és dinàmica, no hi ha un inici d'operacions amb les tasques ja creades, sinó que aquestes van arribant amb el temps. Els robots, encara que no tinguin cap tasca assignada, es mouen aleatòriament pel magatzem, així quan entra una nova tasca augmenta la probabilitat de tenir un robot proper.

Considerem que tenim un conjunt de robots homogenis $R = \{r_0, ...r_n\}$. Cada robot només pot executar una tasca a la vegada. Com que cada producte només pot ser transportat per un robot a la vegada, tenim un sistema ST-SR (*Single-Task Single-Robot*) [14]

$P = \{d_i\}$ és el conjunt de ports de recollida dels productes i $U = \{U_i\}$ els ports de descàrrega.

El conjunt de tasques $T = \{t_0, ...t_m\}$ està composat per ordres de transport $t_i = (L_{pick}, L_{unload}), L_{pick} \in P, L_{unload} \in U$. Els paràmetres bàsics de cada ordre de transport són la localització del port de recollida (L_{pick}) i la localització

del port de descàrrega (L_{unload})).

El conjunt de costos $C = \{c_{ij}\}$ representa el cost que suposa per al robot i executar la tasca j. I el conjunt $A = \{a_{ij}\}$ són les assignacions., en aquest cas $a_{ij} = 1$ si el robot i està assignat a la tasca j, i 0 en la resta de casos.

Per tant el que volem és minimitar el cost de la solució global, representat a la funció 2.1

$$solCost = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij} * c_{ij} \tag{2.1}$$

2.2 Cost de la solució

A l'equació 2.1 hem vist que cada robot té associat un cost d'execució per cada tasca (c_{ij}), i la suma d'aquests ens donarà el cost global de la solució. Ara bé, depenent de la mètrica utilitzada, la variació del cost ens donarà informació sobre aspecte concrets del domini d'aplicació[26][3].

. En el cas del transport de productes dins dels magatzems automatitzats, i en aquest treball, les següents mètriques són les més rellevants[27]:

- **Distància total:** és distància recorreguda pel robot a l'hora de transportar els productes. Això pot ser interessant pel que fa a consum d'energia d'aquest, per exemple. Ara bé, com que aquesta mesura no fa referència el temps, no inclou les esperes que puguin haver fet els robots.
- **Temps de servei de la tasca:** és el temps des que una tasca entra al sistema fins que el robot la descarrega. És una mesura de la qualitat del servei des del punt de vista de les tasques. Aquesta mesura es pot veure molt influenciada per la configuració en nombre de robots o en colls d'ampolla d'arribada de tasques. Si hi ha pocs robots potser aquests transporten ràpidament i en poca distància les tasques, però les tasques que queden en estat d'espera fan augmentar la mitja del temps de servei.
- **Temps de completar la tasca:** aquesta mesura ens indica el temps que ha trigat la tasca en executar-se des que ha estat assignada a un robot (o des que ha estat assignada per primer cop en funció de l'estratègia d'assignació implementada). Aquest valor és interessant analitzar-lo juntament amb el temps de servei ja que un bon temps de completar la tasca, juntament amb un empitjorament del temps de servei ens confirma que hi ha un coll d'ampolla en algun punt del sistema i les tasques queden en espera al arribar. També és interessant minimitzar aquesta mesura als casos d'aplicació on els productes s'han de transportar ràpidament, tot i que poden esperar (per exemple amb refrigerats o productes perillosos).

2.3 Solució com a optimització determinista

El problema d'assignació de tasques es pot veure com un problema d'optimització matemàtica. Les solucions que usen aquests mètodes es basen en la cerca de la solució òptima dins l'espai de solucions guiades per una (o més) funció objectiu [2]. Aquesta cerca està acotada per un conjunt de restriccions en forma d'equacions lineals. Les solucions deterministes proporcionen resultats òptims, però tenen l'inconvenient de ser algorismes intractables a mesura que l'espai de

solucions creix, i això succeeix amb l'augment de robots i nombre de tasques. A més, aquests mètodes requereixen un coneixement previ de totes les tasques del sistema, l'entrada dinàmica de tasques implicaria la modificació del sistema d'equacions. Per això, en casos de pocs robots i tasques intenten adaptar-se a l'entrada de noves tasques amb execucions múltiples a mesura que arriben noves tasques. El mètode hongarès [20] és un exemple clàssic d'algorisme determinista.

A [27] es descarta la utilització d'aquests mètodes per al cas d'estudi d'aquesta tesi. En aquest cas es formulava el problema com a model de *Mixed Integer Linear Programming* (MILP)[13] per dissenyar una solució centralitzada, i CPLEX com a *solver* [18] per trobar la solució.

A partir de la descripció del problema (ports, distàncies entre ells, tasques i els seus punts de recollida i descàrrega, robots) el model generava un conjunt de nodes de tasques (V), arcs enllaçant-los (A) i el conjunt de costos dels arcs (C) de la següent manera:

- V és el conjunt de tasques a executar, una per cada producte $P = \{p_0, ..., p_n\}$, v_0 representa les operacions de recàrrega del robot, és el node inicial.
- A són els arcs $(i, j) \in V^2 : i \neq j$. Un $(i, j) \in A$ indica que un robot pot anar des de la destinació del producte p_i fins a la posició del producte p_j.
- C és el conjunt de costos per executar una tasca després de l'altra. El valor de c_{ij} representa el cost d'utilitzar l'arc $(i, j) \in A$, és a dir, executar la tasca j després de i.

El model té tres variables de decisió per ajustar-les per aconseguir el valor objectiu òptim: el nombre màxim de tasques que un agent mòbil pot executar (Q), les tasques que pertanyen a una ruta (seqüència ordenada de nodes), i les variables x_{ij} indicant si l'arc $(i, j) \in A$ pertany a la solució. En un model com aquest, el graf que representa l'espai de solucions es fa intractable a mesura que augmenta el nombre de robots i tasques, a més de les restriccions que afecten a les rutes. En el pitjor dels casos una cerca exhaustiva ens porta a una complexitat de $O(|K|!)^{|A|}$ per $|K|$ tasques i $|A|$ robots[25].

Excepte en el cas més senzill (només 25 tasques) l'algorisme d'optimització no va poder trobar la solució òptima en una hora, i va retornar la millor assignació que pot trobar en aquest temps. Tenint en compte que en un cas real ens poden arribar unes 250 tasques per hora, el mètode MILP es fa inviable, doncs el temps de càlcul de la solució és molt superior al temps d'execució de les tasques.[27]. Altres treballs, com [10] també descarten utilitzar MILP pel seu augment de complexitat a mesura que el sistema creix (en el seu cas, la formulació del model MILP tindria de l'ordre de 10^7 variables a més d'un nombre exponencial de restriccions).

2.4 Mètodes aproximats

Per tal de reduir el temps de computació, s'utilitzen mètodes aproximats que, si bé no donen una solució òptima, sí que en calculen una d'aproximada en un temps raonable. Aquí tenim des de mètodes metaheurístics [34] com els algorismes genètics, machine learning o swarm intelligence als mètodes de mercat basats en subhastes. En alguns casos les assignacions d'aquests mètodes només difereixen un 9%[27] o un 10%[15] de la solució òptima.

2.4.1 Mètode de subhastes basat en CNP

Els mètodes basats en conceptes de mercat (*market-based*) són els més àmpliament utilitzats en la gestió de magatzems automatitzats. En concret aquells basats en el concepte de subhastes del protocol *Contract Net* (CNP)[35]. Les subhastes són un mecanisme de coordinació ràpid, escalable i fàcilment distribuïble ja que es pot implementar amb un gestor de subhastes centralitzat o bé fer la gestió distribuïda entre diferents agents del sistema.

La idea d'aquest protocol es mostra molt simplificada a la figura 2.1. Al pas 1 el gestor de subhastes de $Tasca_1$ envia un missatge de tipus CFP(*Call For Proposals*) als robots, indicant a on s'ha de transportar l'article. Els robots responen amb un PRP (*Proposal*) al pas 2 indicant el seu cost per transportar aquest producte. El gestor de tasca avalua els valors que ha rebut dels robots i decideix que el guanyador és el robot 2, per això a pas 3 envia un ACCEPT al robot 2 perquè sigui l'encarregat de fer el transport.

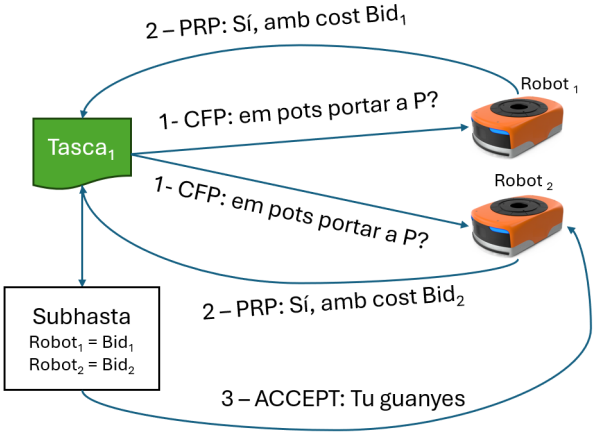


Figura 2.1: Mètode basat en subhasta

2.4.2 Repetició de subhastes

El mecanisme de subhastes s'adapta a l'entrada dinàmica de tasques, doncs cada vegada que entra un ordre de transport al sistema, el gestor d'aquesta nova tasca genera una nova subhasta. I, a més, en casos relativament petits pel que fa al nombre d'agents el cost d'aquesta solució només difereix un 9-10% respecte a l'òptim[27].

Ara bé, estem en un entorn on es poden produir esdeveniments inesperats. A més de l'entrada dinàmica d'ordres de transport, poden entrar nous robots, o bé algun obstacle fa canviar la ruta inicial d'un robot que estava fent un transport, etc. A més, els robots que han completat la descàrrega es mouen erràticament esperant noves ordres. A [27] es proposa aprofitar aquests events per repetir subhastes i millorar la qualitat de la solució. Però no totes les subhastes es poden repetir. Una ordre de transport es pot descomposar en dues parts: primer el robot ha d'anar al punt de recollida, i després ha de transportar el producte a la zona de descàrrega. Per això quan es produeixen certs events

al sistema (com, per exemple, l'entrada d'una nova ordre, la finalització d'una tasca, ...) els gestors de tasques tornen a enviar un CPF (*Call For Proposal*), encara que ja tinguin un robot anant cap al punt de recollida. I, els robots que ja tenen tasca assignada però estan a la fase d'anar al punt de recollida poden respondre a aquesta subhasta amb el seu valor de cost dins d'un missatge PRP. En aquest cas els robots que estan anant a un punt de recollida només responen a la subhasta de la seva tasca. Això permet que, abans que es carregui el producte, si hi ha un altre robot que pot fer el transport amb menys cost, es re-assigna la tasca a aquest nou robot.

2.4.3 Inconvenients de la repetició de subhastes

El mètode de la repetició de subhastes acosta una mica més el cost de la solució al valor òptim, aprofitant els esdeveniments que ocorren al sistema, l'estat actual de la tasca i el moviment erràtic dels robots que no tenen tasca assignada. Aquesta estratègia, però, té alguns inconvenients.

- **Càrrega de comunicacions:** s'ha de gestionar una nova situació: la re-assignació de tasques. L'augment del nombre de subhastes, i la gestió dels canvis d'assignació suposen un increment de les comunicacions que pot ser important en funció del nombre de robots i el layout del magatzem.
- **Mètode greedy:** cada gestor tasca només contempla els valors de cost d'aquesta tasca que li aporten els robots amb els missatges PRP. I tria el de menor cost. Aquest robot, un cop assignat ja no respon a les altres subhastes. Això vol dir que l'ordre en que s'executen les subhastes importa. És un comportament "egoista"(*greedy*).

Capítol 3

Solució: combinar mètodes

Hem vist que hi ha diversos mètodes de tractar el problema MRTA, i disposem de taxonomies[14][25] que ens permeten classificar el nostre sistema i veure els avantatges i inconvenients de cada mètode. I, a partir d'això, en aquest episodi es mostra la solució proposada i implementada en aquest treball.

3.1 Hipòtesi

Al capítol 2.3 hem vist que els mètodes d'optimització deterministes obtenen la solució òptima a l'assignació de tasques. Aquests mètodes, però, solen ser intractables a mesura que creix l'espai de solucions.

També sabem (veure 2.4) que els mètodes aproximats ens proporcionen una solució en un temps acceptable. En el cas dels mètodes de subhastes basats en CNP aquesta solució pot apropar-se a l'òptim[27]. A més, les subhastes proporcionen totes els avantatges de les solucions distribuïdes. Per altra banda les subhastes tenen l'inconvenient d'actuar de forma *greedy*, i la millora de la repetició de subhastes suposa una càrrega addicional a les comunicacions.

Aquest treball proposa combinar un mètode determinista amb un sistema de coordinació basat en repetició de subhastes. En concret la solució proposada a [27].

3.2 Objectiu

Aquest treball vol millorar la solució aportada a [27]. Millorar-la acostant-la a la solució òptima i reduint la càrrega de comunicacions. El principal inconvenient d'aquesta solució és que actua de forma cobdiciosa (*greedy*), per tant necessita incorporar la cooperació entre els gestors de tasques, que són els encarregats de gestionar les subhastes. Aquesta cooperació permet obtenir informació d'un conjunt de tasques en el moment de triar el guanyador de la subhasta. I és en aquest punt on la nova solució ha d'aplicar un mètode d'assignació determinista.

A la taula 3.1 veiem en un exemple aquesta necessitat de compartir informació. Tenim tres tasques i tres robots. Amb una estratègia *greedy*, en color vermell, cada subhasta es queda el seu cost mínim. En aquest cas $Task_0$ en primer lloc s'assigna $Robot_0$ (el seu mínim cost). A continuació $Task_1$ s'assigna $Robot_1$, que és el seu mínim disponible (el $Robot_0$ ja el té assignat la primera

tasca). Finalment $Task_2$ s'assigna el robot 2. Tenim doncs un cost global de $25 + 50 + 48 = 123$. Disposant de tota la informació de taula, podem veure de color verd l'assignació òptima. $Task_0$ s'assigna a $Robot_2$ (tot i que en aquesta subhasta té un valor que no és el mínim), i les tasques 1 i 2 s'assignen als robots 0 i 1 respectivament. Tenim, doncs, una assignació òptima amb un cost de $30 + 32 + 47 = 109$, un 11% menor que el cost de l'algorisme *greedy*.

	$Robot_0$	$Robot_1$	$Robot_2$
$Task_0$	25	29	30
$Task_1$	32	50	51
$Task_2$	45	47	48

Taula 3.1: Diferència entre assignació *greedy* (Vermell) i agrupant (verd)

3.3 Procediment

Per poder aplicar un mètode determinista en comptes de l'algorisme *greedy* que utilitza la solució de [27] cal agrupar la informació de les subhastes. Això es mostra a la figura 3.1, on tenim 3 tasques i 3 robots que participen a les respectives subhastes. A diferència del mètode *greedy* aquí cada robot recull totes les peticions de subhastes, a continuació envia a aquests gestors de tasques el seu valor de cost per aquesta tasca, seguit de tots els seus valors de cost per a les altres subhastes en les que participa. Com a resultat, el gestor de tasques corresponent obté una matriu amb els costos agrupats. En el cas de la figura 3.1 cada fila conté els valors de licitació d'una subhasta, i cada columna els valors de licitació d'un robot per cada subhasta.

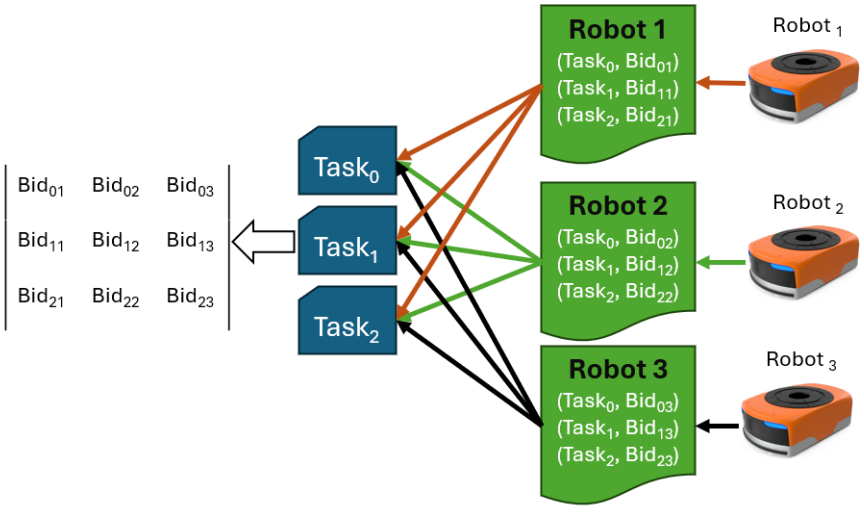


Figura 3.1: Agrupació de subhastes

3.3.1 El mètode Hongarès

Un cop tenim la informació d'una agrupació de subhastes ja podem utilitzar un mètode determinista d'assignació de tasques. En el cas d'aquest treball implementem el mètode hongarès, també conegut com algorisme d'assignació de Munkres[24]. Aquest algorisme resol problemes d'optimització en temps polinomial ($O(n^3)$). Com que aquest algorisme modela el problema d'assignació com una matriu de costos, es pot aplicar a la matriu de dades agrupades de les subhastes.

3.3.2 Munkres*

El mètode hongarès troba una assignació de tasques òptima, però aquesta solució potser no és única. En aquest cas, segons l'ordre de les columnes a la matriu de costos l'algorisme ens pot donar una o altra assignació òptima. Això pot suposar una càrrega encara més important de les comunicacions, doncs davant de cada event del sistema que ho requereixi, les noves subhastes provoquin una nova assignació innecessària. Encara més, si entren tasques noves al sistema, i es genera una nova matriu de costos que inclou tasques ja assignades, potser hi ha una de les assignacions òptimes que manté aquestes assignacions. Per tot això, en aquest treball s'ha implementat un algorisme, que utilitza el mètode hongarès només si és necessari. A més, l'aplica a submatrius si el cost de les assignacions d'aquestes més el cost de les assignacions prèvies és igual al de la nova assignació calculada a partir de la matriu d'assignacions original. Hem anomenat Munkres* a aquest algorisme.

Donat el conjunt de tasques $P = \{p_i\}$ que estan subhastant en un mateix temps de simulació, i el conjunt de robots $R = \{r_i\}$ que participen en aquestes subhastes, definim la matriu de licitacions (3.1):

$$B = \begin{pmatrix} b_{00} & ... & b_{0n} \\ ... & b_{ij} & ... \\ b_{m1} & ... & b_{mn} \end{pmatrix} \tag{3.1}$$

Aquesta matriu representa el conjunt de licitacions dels robots per les subhastes, on b_{ij} és el valor de licitació del robot j per a la tasca i. Així doncs, cada fila de la matriu representa el conjunt de valors per a la subhasta i, i cada columna representa el conjunt de valors amb que el robot j licita per a les diferents subhastes.

Aplicant l'algorisme de Munkres a B obtenim la matriu d'assignacions (3.2):

$$A = Munkres(B) = \begin{pmatrix} a_{00} & ... & a_{0n} \\ ... & a_{ij} & ... \\ a_{m1} & ... & a_{mn} \end{pmatrix} \tag{3.2}$$

On $a_{ij} = 1 \Leftrightarrow \exists p_i \in P, \exists r_j \in R / assigned(p_i) = r_j, a_{ij} = 0$ en la resta de casos. La funció *assigned* retorna el robot assignat a la tasca que li passem per paràmetre.

Tot i que l'algorisme d'assignació de Munkres retorna una assignació òptima, no retorna totes les possibles assignacions (si hi ha més d'una assignació òptima). La idea de l'algorisme, que podem anomenar Munkres*, és veure si el cost al aplicar Munkres sobre la matriu formada pels robots i subhastes no assignats,

sumat al cost de les assignacions prèvies és igual al cost que ens dóna les assignacions de la matriu A. En aquest cas podríem evitar reassignacions innecessàries, doncs tenim una altra assignació òptima.

Tenim que el cost òptim de les assignacions obtingudes per Munkres ens el dona (3.3):

$$Cost(B, A) = \sum (a_{ij} \cdot b_{ij}), a_{ij} \in A, b_{ij} \in B \tag{3.3}$$

I A^- (3.4) és el conjunt d'assignacions prèvies a A per a robots i tasques que participen a B:

$$A^- = \{(r_j, p_i)\}/r_j \in R, p_i \in P, assigned(r_j) = p_i \tag{3.4}$$

D'aquí que el cost de les assignacions prèvies (cost previ) sigui:

$$CP = \sum b_{ij}/\exists (r_j, p_i) \in A^- \tag{3.5}$$

I el cost dels robots i tasques no assignats es calcularà a partir de la matriu B^r:

$$B^r = (b_{ij})\,/b_{ij} \in B, \nexists (r_j, p_n) \in A^-, \nexists (r_m, p_i) \in A^- \tag{3.6}$$

És a dir, B^r és la matriu B menys les files de les subhastes ja assignades i les columnes dels robots ja assignats. Per exemple:

$$B = \begin{pmatrix} 5 & 2 & 4 & 3 \\ 4 & 2 & 7 & 3 \\ 5 & 2 & 4 & 3 \\ 9 & 11 & 13 & 18 \end{pmatrix}, A^- = \{(r_1, p_2)\}$$

$$B^r = \begin{pmatrix} 5 & - & 4 & 3 \\ 4 & - & 7 & 3 \\ - & - & - & - \\ 9 & - & 13 & 18 \end{pmatrix} = \begin{pmatrix} 5 & 4 & 3 \\ 4 & 7 & 3 \\ 9 & 13 & 18 \end{pmatrix}$$

A partir d'aquestes definicions podem definir l'algorisme 1, que anomenem Munkres*.

Algorisme 1 Algorisme Munkres*

```
1: ReassignedCost ← 0
2: A^r ← ∅
3: A ← MUNKRES(B)
4: C_A ← COST(B, A)
5: CP ← 0
6: B^r ← B
7: for all (r_j, p_i)) ∈ A^− do
8:     CP ← CP + b_ij
9:     B^r ← SUPRESSROW(B^r, i)
10:    B^r ← SUPRESSCOLUMN(B^r, j)
11: end for
12: if CP > 0 then
13:    A^r ← MUNKRES(B^r)
14:    ReassignedCost ← COST(B^r, A^r) + CP
15: end if
16: if ReassignedCost > C_A then
17:    Assign(A)
18: else if A^r != ∅ then
19:    Assign(A^r)
20: end if
```

Exemple d'execució de Munkres*

El següent és un exemple d'assignació utilitzant Munkres*, extret d'una de les simulacions executades per a l'estudi al capítol de resultats. Tenim 7 tasques per assignar: $T = \{t_{26}, t_{27}, t_{30}, t_{31}, t_{32}, t_{33}, t_{34}\}$ i 5 robots: $R = \{r_2, r_3, r_4, r_7, r_8\}$.

Els costos de cada tasca formen les files de la matriu de costos B, segons l'ordre que tenen a T. És a dir, la primera fila conté els costos de la tasca t_{26}, la segona els de t_{27}, etc. Les columnes tenen els costos de cada robot. Així la primera columna té els costos de r_2 per cada subhasta, la segona columna els costos de r_3, etc. Executant l'algorisme de Munkres sobre aquesta matriu obtenim la matriu d'assignacions A. Aquestes matrius tenen els següents valors:

$$B = \begin{pmatrix} 36 & 23 & 24 & 16,45 & 23,45 \\ 36 & 37 & 24 & 16,45 & 23,45 \\ 24 & 25 & 12 & 18,45 & 11,45 \\ 60 & 47 & 48 & 40,45 & 47,45 \\ 30 & 33 & 32 & 38,45 & 17,45 \\ 52 & 55 & 42 & 48,45 & 39,45 \\ 29 & 16 & 17 & 9,45 & 16,45 \end{pmatrix}, A = Munkres(B) = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Per tant, l'assignació que ens dóna Munkres és: $\{(r_2, t_{32}), (r_3, t_{34}), (r_4, t_{30}), (r_7, t_{27}), (r_8, t_{26})\}$, amb cost $30 + 16 + 12 + 16,45 + 23,45 = 97,9$. Les tasques 31 i 33 queden sense assignar.

En aquest exemple algunes de les tasques ja estaven assignades, tenim aquestes assignacions prèvies a $A^- = \{(r_3, t_{26}), (r_7, t_{27}), (r_8, t_{32})\}$ D'aquí obtenim la

matriu "reduïda"descartant les assignacions anteriors a la matriu de costos:

$$B^r = \begin{pmatrix} - & - & - & - & - \\ - & - & - & - & - \\ 24 & - & 12 & - & - \\ 60 & - & 48 & - & - \\ - & - & -- & - & - \\ 52 & - & 42 & - & - \\ 29 & - & 17 & - & - \end{pmatrix} = \begin{pmatrix} 24 & 12 \\ 60 & 48 \\ 52 & 42 \\ 29 & 17 \end{pmatrix}$$

Tenim, doncs, 4 tasques assignables, $\{t_{30}, t_{31}, t_{33}, t_{34}\}$, i dos robot a qui assignar-les, $\{r_2, r_4\}$. Aplicant Munkres sobre aquesta matriu ens dóna la matriu A^r, i d'aquí en treiem el cost de les noves assignacions:

$$A^r = \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} Cost^r = 29 + 12 = 41$$

Com que el cost de les assignacions prèvies (cost previ) és $CP = Cost(A^-) = 23 + 16, 45 + 17, 45 = 56, 9$ el cost total aprofitant les reassignacions és:

$$Cost\_reassignat = CP + Cost^r = 56, 9 + 41 = 97, 9$$

Con hem vist al principi d'aquest exemple, aquest és el cost de l'assignació que ens dóna l'algorisme de Munkres al aplicar-lo sobre la matriu de costos B. De fet, com que Munkres ens dóna una solució òptima, l'assignació que fem aprofitant les assignacions anteriors no pot ser millor, com a molt pot tenir el mateix cost, és a dir, ser una altra solució òptima, com en aquest cas. Per tant, agafem l'assignació que aprofita les assignacions prèvies: $\{(r_2, t_{34}), (r_3, t_{26}), (r_4, t_{30}), (r_7, t_{27}), (r_8, t_{32})\}$.

Capítol 4

El model multiagent

L'objectiu d'aquest treball és millorar la solució obtinguda a [27], per tant es basa en el mateix model d'agent. El MRS s'ha modelat com a sistema multiagent (MAS), format per dos tipus d'agent que gestionen les tasques (ordres de transport) i controlen els robots. A continuació veurem l'estructura dels agents, la descripció formal del comportament d'aquests i les adaptacions fetes per tal d'assolir l'objectiu d'aquest treball.

4.1 El model d'agent

En aquest sistema multiagent hi ha 2 tipus d'agents encarregats de la coordinació i el transport dels productes:

- **Managers**: són els agents encarregats de gestionar les tasques, iniciar les subhastes associades a una ordre de transport i comunicar-se amb els robots
- **Executors**: són els agents que controlen els robots.

Aquests agents estan formats per una estructura de capes. Cada capa és un controlador que s'encarrega d'una tasca específica, i es pot comunicar amb els altres controladors de l'agent. La capa de nivell superior s'encarrega de les comunicacions amb els altres agents i la capa inferior s'encarrega de la part reactiva, o física en el cas dels robots.

A la figura 4.1 veiem que els agents *Manager* estan formats per una capa deliberativa (DLB) i una capa reactiva. El controlador de la primera pot enviar i rebre missatges dels executors, és aquí on es gestiona la coordinació amb els altres agents del sistema. A nivell inferior tenim la capa interna reactiva (RCT), oculta als altres agents. Aquesta capa s'encarrega de controlar el pas del temps, gestionar temporitzadors interns i informar a la capa deliberativa que cal fer algun canvi d'estat. Per exemple, si DLB ha iniciat una subhasta i no ha rebut cap oferta dels robots passat un cert temps, li envia un senyal a RCT indicant que no hi ha ofertes (*no-bids*), i RCT inicia un temporitzador abans d'enviar un senyal a DLB perquè inicii una nova subhasta, així es dona temps als executors perquè finalitzin els seus estats actuals.

L'estructura de capes dels agents *Executor* la veiem a la figura 4.2. En aquest cas tenim 4 capes. La capa superior és el controlador de comunicacions

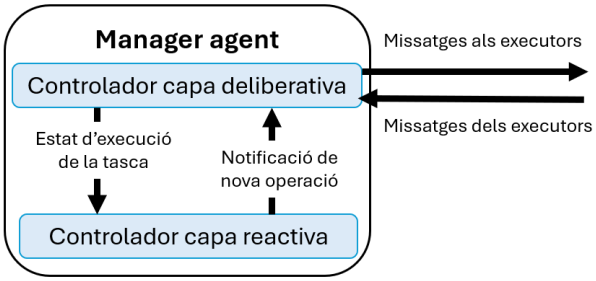


Figura 4.1: Estructura de capes de l'agent Manager

(L4), que es comunica amb la capa deliberativa dels *Managers*. Amb aquests s'intercanvien missatges referents a les subhastes, les ofertes, el transport de productes, etc. Just a sota tenim el planificador de rutes *Path Planner* (L3). Quan L4 necessita saber el cost per anar a un port concret, per exemple, li demana a L3 i aquest fa el càlcul de la ruta més curta. Si l'executor s'ha de moure el controlador L2 (seguidor de rutes) descompon la ruta calculada per L3 en segments fàcils de seguir, i els va enviant a L1. L1 respon afirmativament quan s'ha completat cada segment de ruta. En aquest cas L1 és el gestor de trànsit que resol els conflictes de moviment amb altres robots. Resolts els conflictes de trànsit, la capa L0 és la inferior, s'encarrega d'executar els moviments bàsics del robot, girs, detecció d'obstacles, etc.

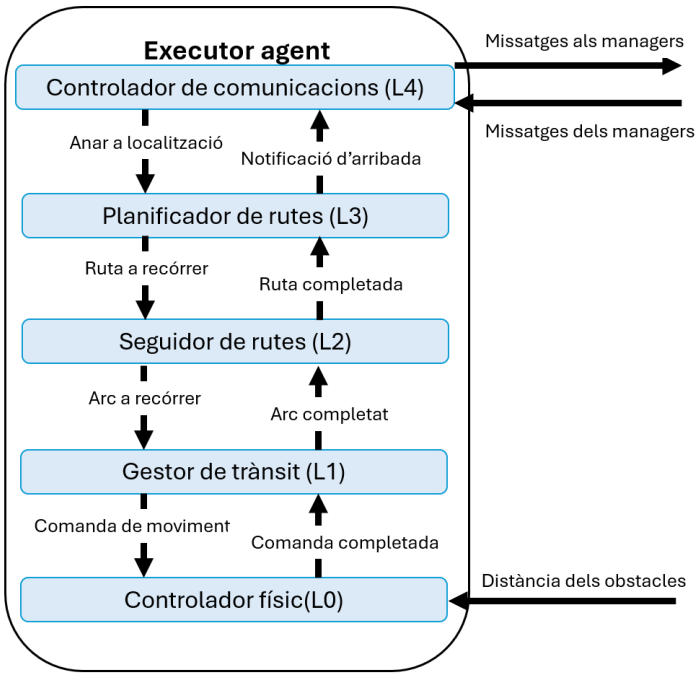


Figura 4.2: Estructura de capes de l'agent Executor

4.2 Model i formalització del comportament

A l'hora de definir el comportament dels controladors i la interacció entre ells ens interessa que aquest comportament sigui fàcilment predictible i verificable. A més ens interessa que aquest sistema sigui fàcilment integrable en un entorn de magatzems automatitzats o plantes industrials, doncs aquests entorns estan controlats per sistemes distribuïts i poblats per equips els controladors dels quals són, essencialment, màquines d'estat[7].

És per això que l'enfocament de [27] consisteix a utilitzar màquines d'estat finites (EFSM) per formalitzar i parametritzar el comportament dels agents.

4.2.1 Extended Finite State Stack Machines (EFS^{2}M)

Per definir el comportament dels controladors utilitzem les EFS^{2}M, que són una extensió de les EFSM (*Extended Finite State Machines*).

Les FSM es poden descriure com una tupla $(Q, \Sigma, \Omega, \Delta, q_0)$ on Q és el conjunt d'estats, Σ són les entrades, Ω és el conjunt de possibles sortides, q_0 és l'estat inicial i Δ és el la funció de transició $\Delta : Q \times \Sigma \to Q \times \Omega$

En les EFSM hi ha un conjunt addicional de valors de variables, en la forma $V : D \to D$, on N és el conjunt de noms de variables i D és el conjunt de valors que poden tenir. D'aquí que la funció de transició és $\Delta : (Q \times V) \times \Sigma \to (Q \times V) \times \Omega$.

En el cas de les EFS^{2}M aquestes tenen una pila d'estats[29], d'aquí que es defineixin com una tupla $(Q, V, U, \Sigma, \Omega, \Delta, q_0, v_0, u_0)$, on hem afegit el conjunt de valors de variables V, U és un conjunt de símbols que representen els estats de Q (la pila), q_0 és l'estat inicial, V_0 el valor inicial de les variables i U_0 és l'estat inicial de la cadena o pila de símbols.

Es representen gràficament igual que les EFSM, afegint petits hexàgons i cercles per representar les operacions amb la pila. Un petit cercle al final d'una fletxa indica que s'està posant a la pila (*push*) l'estat amb el nom que hi ha dins el cercle, o l'estat actual si no hi posem cap nom. També es poden posar els cercles a l'inici de la fletxa per indicar que s'està extraient l'últim estat apilat (*pop*). En el cas dels hexàgons el significat és similar al del cercle però en aquest cas si el posem al final de la fletxa indica una operació *pop* i l'operació contrària si el posem al principi de la fletxa.

4.2.2 *Manager*: controlador deliberatiu

Anem a veure la màquina d'estats amb el comportament de la capa deliberativa (DLB) de l'agent *Manager*. És la capa superior del model d'agent, i és la que s'ha adaptat en aquest treball per implementar la millora proposada. A la figura 4.4 es mostra una versió simplificada d'aquesta (no apareixen, per exemple els estats de càrrega al punt de recollida i descàrrega al punt de destí ja que no afecten la gestió de la subhasta).

Quan el controlador està en l'estat inicial (INIT) envia una petició de propostes a tots els executors (CFP). A la figura es mostra la variable *Omsg* que representa la cua de missatges a enviar. En aquest estat inicial el destinatari són tots els agents i el missatge també inclou les posició de recollida (*curPos*, on està actualment el producte) i el port de destí. A continuació guarda dins la pila l'estat actual i PRI_AUC, passant a GET. Aquest és l'estat on es recullen

les propostes dels executors. Fixem-nos que la variable CD (*Collect Deadline*) controla el nombre de cicles en que l'estat s'espera per anar recollint les propostes dins de la variable *Prps*. Una vegada finalitza aquesta recollida es passa a l'estat que treu de la pila, PRI_AUC. Abans, però, avalua les propostes rebudes per decidir quina és la millor oferta.

És en aquest punt on tenir informació de les subhastes d'altres *managers* ens permet aplicar l'algorisme mostrat al capítol 3.3.2. A l'estat PRI_AUC, pot passar que no tinguem cap executor guanyador, amb una estratègia *greedy* seria simplement que no tenim propostes, i amb informació compartida podria ser que l'algorisme d'assignació hagués donat una solució on aquest *manager* de moment no té executor assignable. En qualsevol cas, si no hi ha guanyador, s'inicialitzen variables i es passa a un estat RESTART on espera els cicles indicats a PR (*Primary Deadline*) abans de tornar a l'estat inicial. Si l'avaluació de les propostes ha donat un guanyador li envia un missatge ACCEPT d'acceptació, i passa a ser l'executor assignat.

El següent estat és d'espera (WAIT) mentre el robot assignat està anant al punt de recollida. En aquest estat podem veure dues coses importants. La primera és la variable RC, que representa els missatges rebuts del controlador de la capa inferior del *manager*, la part reactiva (RCT), i la segona és que aquí podem veure com funciona la repetició de subhastes. Per descriure el mecanisme de repetició de subhastes el diagrama mostra com, mentre està a l'estat WAIT el manager pot rebre un missatge RECALL de RCT. Això pot passar a l'estat WAIT perquè encara no s'ha arribat al punt de recollida, i un altre *manager* (que no estava disponible a la primera subhasta) podria millorar el cost de transport. Així doncs, si RCT envia un missatge RECALL a DLB mentre està a WAIT, aquest guarda a la pila l'estat *SEC_AUC*, envia un nou CFP i torna a l'estat GET per recollir noves ofertes. L'estat SEC_AUC es comporta bàsicament com el de la primera subhasta. Ara bé, aquí potser el guanyador ha estat un executor diferent del que teníem assignat. En aquest cas s'envien dos missatges (al diagrama es veuen assignats a la cua de missatges *Omsg*): un ABORT a l'executor que tenia assignat fins ara, i l'ACCEPT al guanyador. Si tot va bé i no arriba cap RECALL, l'estat WAIT rebrà un READY de l'executor assignat indicant que està al punt de recollida i pot començar a moure's. DLB li respondrà amb un ON perquè es posi en xarxa cap al punt de descàrrega i passarà a l'estat RIDE, d'on anirà a l'estat final (IDLE) en rebre un DONE. En aquest diagrama simplificat no es mostren els dos estats que representen l'estat de "carregantï "descarregant", doncs els seu funcionament és similar al RIDE i en aquest treball està centrat en les subhastes.

Consideració sobre el *Collect Deadline*

Abans hem vist que el valor d'aquesta variable ens indica el nombre màxim de cicles en que la màquina d'estats es manté en l'estat d'espera de propostes (GET). Aquest valor ha de ser suficient per assegurar que es reben totes les propostes dels executors. Es pot assignar manualment en funció dels paràmetres del sistema, analitzant les màquines d'estats o bé, simplement, a partir de l'experiència en execucions del sistema.

Si tot el sistema s'està executant dins un únic node de computació es pot calcular automàticament aquest valor tal i com es mostra a la figura 4.3. La figura és un fragment de la màquina d'estats del *Manager*, mostrant només

l'estat de recollida d'ofertes (GET) i l'estat en que es decideix la subhasta, Aquí no distingim entre subhasta primària i secundària. Al mostrar només aquest fragment no ens cal mostrar les operacions amb la pila d'estats. La variable S representa l'instant de temps en que s'entra a l'estat GET, i T és el valor actual del temps al sistema. Per això mentre S segueixi sent igual a T la EFSM romandrà en estat de recollida, augmentant el comptador de cicles C. En el moment que passi el temps la variable Cout(*cycleout*) s'actualitzarà amb el comptador de cicles C si el valor d'aquest és superior al que ja tenia Cout abans. Aquest valor es correspon amb el CD de la màquina d'estat completa.

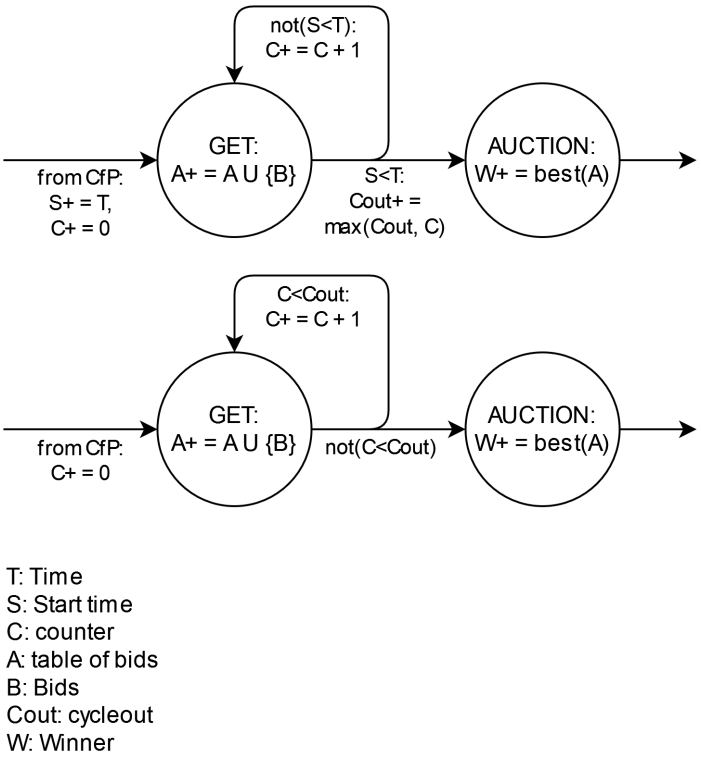


Figura 4.3: Càlcul del *cycleout* (EFSM superior) i mode d'ús (EFSM inferior)

En implementacions distribuïdes aquesta solució no funciona, ja que la comunicació entre nodes requereix el pas del temps a la vegada que els cicles interns que es mostren a la figura. En aquest model les transicions de les EFSM no fan avançar el temps del sistema mentre no calgui fer lectures/escriptures de l'entorn.

4.2.3 *Executor*: controlador de comunicacions

La màquina d'estats que formalitza el comportament del controlador L4 és la que podem veure a la figura 4.5. És una versió simplificada, on es destaca el comportament bàsic de l'agent juntament amb les adaptacions que s'han fet per aquest treball.

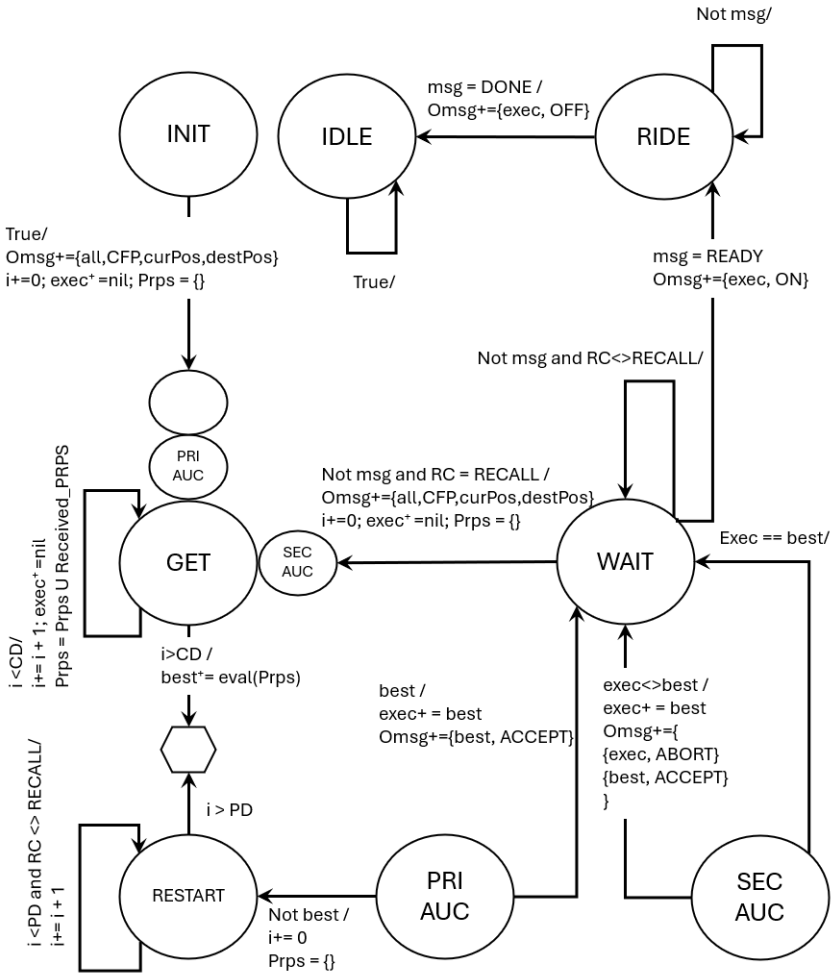


Figura 4.4: EFS^{2}M simplificada del control deliberatiu del *Manager*

Inicialment l'agent està lliure (estat FREE), estat on pot rebre diversos missatges d'inici de subhasta (CFP). Aquí hi ha la principal diferència respecte el controlador L4 de [27]. Es crea una llista de parells $(CFP_i.Manager, bid_i)$ on el primer element és l'identificador del *manager* que ha enviat el CFP i bid_i és el cost amb que aquest executor participa en aquesta subhasta. En resum, fa una llista de totes les subhastes en que participa i els valors de cost per aquestes subhastes. Fixem-nos també que aquesta llista s'envia com a resposta a tots els *managers* dins un missatge PRP.

En rebre un ACCEPT l'agent apila l'estat FREE, s'assigna el *manager* que l'ha acceptat i fa una assignació a "L4C". Aquesta és la cua de missatges cap al planificador de rutes (L3, la capa inferior del model d'agent). Està enviant la comanda per anar al punt de recollida de la tasca. I l'agent passa a l'estat BUSY mentre es dirigeix cap al punt de recollida.

L'estat BUSY és d'espera, en cas de rebre un missatge ABORT del *manager* traurà l'estat FREE de la pila i tornarà a l'inici. Aquí tornem a tenir una diferència important respecte la versió de [27]. En el seu cas, utilitzant una estratègia *greedy*, des de l'estat BUSY només responen al CFP si el que envia la petició és el *manager* que ja té assignat. És a dir, en el seu cas un cop s'assigna un executor aquest no es desassigna si no hi ha un altre executor lliure amb cost menor. En el cas d'aquest treball l'executor envia les seves ofertes a totes les subhastes, això ens permetrà que un executor canviï de *manager* encara que estigui assignat i sigui el de mínim cost pel *manager* que té assignat.

Si L3 informa que s'ha arribat al punt de recollida, s'envia un missatge READY al *manager*. Abans de passar a PICK, que és l'estat d'espera mentre es carrega l'article. Els estats LOADED i DROP són similars a PICK, i gestionen els estats de transport i descàrrega.

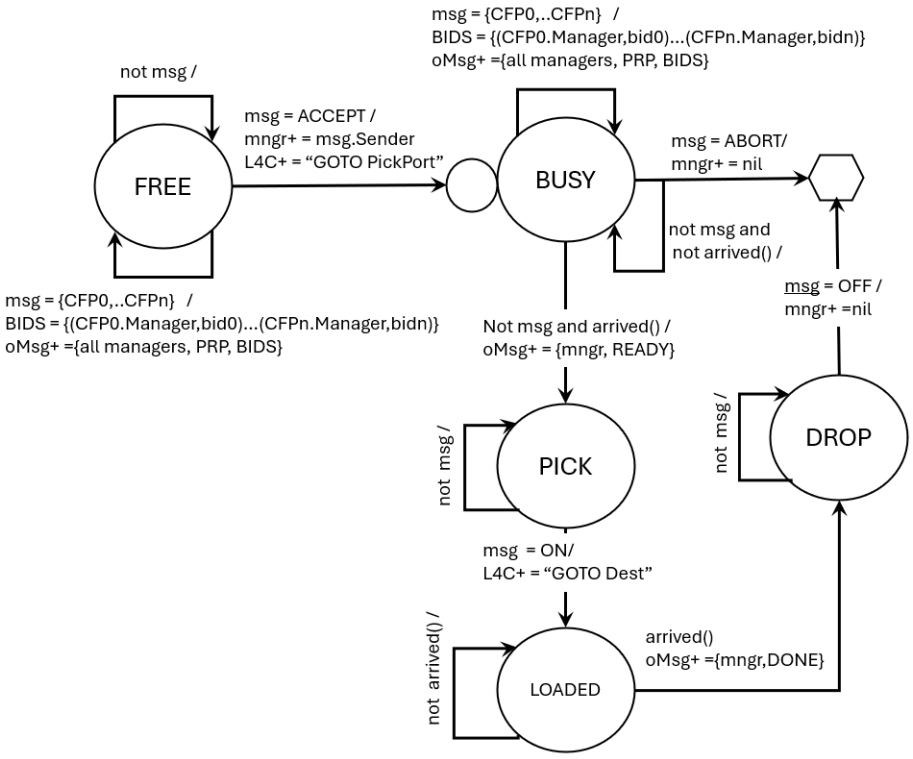


Figura 4.5: EFS^{2}M simplificada del controlador de comunicacions (L4)

4.2.4 Subhastes agrupades vs solució *Greedy*

A les seccions 4.2.1 i 4.2.3, mentre comentàvem el funcionament de les màquines d'estats que defineixen el comportament dels agents, hem vist una mica els punts on la solució *Greedy* de [27] difereix de la solució proposada en aquest treball. Aquí separem les dues solucions i expliquem principals diferències entre ambdues.

Solució *Greedy*

En aquesta solució, només entrar al sistema el *manager* envia un CFP a tots els executors. I aquests envien els PRP de resposta corresponents, independentment de l'estat en que estiguin.

En rebre un CFP els executors calculen la distància fins al punt de recollida més la distància entre el punt de recollida i el punt de descàrrega. I envien aquest valor, com a cost de la oferta dins d'un missatge PRP, sense cap més informació. Aquest missatge només va destinat al *manager* que està gestionant la subhasta. Si un executor ja està assignat a un *manager* només respon amb un PRP a aquest gestor de tasca. I refusa (senyal REF) la participació a les altres subhastes.

El *manager*, si està a l'estat de recollida (GET), en rebre els PRP els guarda mentre no es superi el temps d'espera per recollir les propostes. Passat aquest temps es treu el darrer estat guardat a la pila, que serà PRI_AUC o SEC_AUC, i s'avaluen les propostes guardades. Fixem-nos que aquí el *manager* es comporta com un agent BDI (*Belief-Desire-Intention*) [30][6], doncs guarda les ofertes dels executors com a un conjunt de creences (*beliefs*).

Pel que fa a l'avaluació de les propostes dels executors el *manager* recupera les propostes que li han anat arribant i escull com a executor guanyador el de menor cost. Aquí pot passar que hi hagi diversos executors que poden fer el transport amb el mateix cost mínim. Si la tasca no està assignada o bé l'executor assignat no està entre els guanyadors, s'escull un guanyador a l'atzar entre els que han aportat el cost mínim. Si l'executor que ja tenia assignat la tasca està entre els guanyadors es manté l'assignació per evitar càrrega de comunicacions.

Agrupació de subhastes

Aquesta és la solució implementada en aquest treball. Ja n'hem vist les característiques al capítol 3, vegem aquí els punts de diferència respecte la solució *Greedy* de[27].

La primera diferència respecte el mètode *Greedy* la tenim amb la primera subhasta. En aquest cas no s'envia el CFP només iniciar la tasca, sinó que hi ha una espera per tal de sincronitzar aquesta subhasta amb les dels altres *managers* actius en aquest moment. Una altra diferència important és que aquí tots els executors poden participar a qualsevol subhasta, si no estan en un dels estats de càrrega, transport o descàrrega. A més, tal com hem vist al capítol 3.3 els executors envien a tots els *managers* tota la informació que tenen referent a totes les subhastes. Per això no envien n missatges amb un únic valor, sinó que envien un missatge destinat als n *managers* amb una estructura de dades una mica més complexa.

A 4.1 veiem el contingut d'un d'aquests missatges PRP. En primer lloc l'indicador de missatge proposta (PRP), seguit de la llista amb destinataris, que són els *managers* m_i que han enviat missatges CFP. En tercer lloc hi ha el contingut, format pel *manager* al qual està assignat actualment l'executor, i la llista de parelles $(m_i, cost_i)$ on m_i és un dels *managers* que estan subhastant i $cost_i$ és el cost de la tasca corresponent.

$$msg = \{PRP, \{m_0...m_n\}, \{cur\_manager, \{(m_0, cost_0)...(m_n, cost_n)\}\}\} \quad (4.1)$$

A la part deliberativa del *manager* també es processen de forma diferent

aquests CFP respecte la solució *greedy*. El contingut de les propostes que hem vist a 4.1 s'utilitza per formar la matriu de costos que s'utilitzarà com a paràmetre a l'algorisme d'assignació de Munkres, del qual n' hem implementat i incorporat una versió. A partir dels PRP també es pot formar el vector de tasques ja assignades que s'utilitzarà per al nostre algorisme Munkres* 3.3.2. Tots els *managers* que participen en aquesta agrupació de subhastes executen una còpia de l'algorisme amb les mateixes dades.

Una diferència important respecte la versió *Greedy* és que ara l'avaluació de les propostes potser no dóna un guanyador per a la subhasta. Com que estem compartint informació pot passar, per exemple, que hi hagi més tasques per assignar que robots, i l'assignació menys costosa d'aquest grup de robots no inclou un *manager* que actualment té assignat un robot que està camí de recollida. En la versió *greedy* no es pot "perdre"el robot assignat. Així doncs, un *manager* en estat de subhasta, que té un executor assignat i després d'executar $Munkres^*$ no té guanyador de la seva subhasta, enviarà un ABORT al seu executor, doncs l'assignació del grup de subhastes l'ha assignat a un altre *manager*.

Capítol 5

L'entorn de simulació

Al capítol 4 hem vist els avantatges de definir el comportament dels agents mitjançant màquines d'estats. I per aprofitar aquests avantatges aquestes EFSM s'han de poder integrar en un simulador físic. Aquesta integració ha de permetre descriure els estats i transicions de la EFSM i, a més, permetre la sincronització entre les diferents màquines que componen el sistema.

5.1 Integració amb un simulador físic

Les màquines d'estats amb les que definim el comportaments dels agents requereixen de diversos cicles per realitzar les accions d'alt nivell. Per exemple des del moment en que un *manager* inicia la subhasta fins que envia el missatge d'acceptació (i potser el de refús en el cas de la repetició de subhastes). Tal com indiquen a [28], si executem aquestes EFSM en un simulador físic, on els cicles de les màquines d'estat es corresponguin amb el pas de simulació del simulador, operacions que haurien de ser immediates triguen temps. Això podria provocar problemes greus, doncs els agents estarien prenent decisions en funció d'un estat de l'entorn que ja és passat. Al el nostre grup de recerca[28] s'ha solucionat això separant els dos tipus de cicles de simulació. Als cicles de pas de temps, del simulador s'afegeix un comptador de cicles intern per les operacions de les màquines d'estat que no requereixen el pas del temps.

Això ho veiem esquematitzat a la figura 5.1. El comptador de pas de temps del simulador *sTime* s'utilitza per les tasques que tenen consum de temps com, per exemple estar esperant que un sensor ens doni una lectura concreta, o certs esdeveniments de l'entorn. Totes aquestes operacions de lectura o espera de dades de l'entorn i d'escriptura o modificació de l'entorn es mostren a la figura en forma de processos *Read environment* i *Write environment* respectivament. A la figura també veiem representades quatre màquines d'estats que, mentre no requereixin dades externes o esperar que passi un event extern, utilitzen el comptador intern de cicles per controlar les seves transicions.

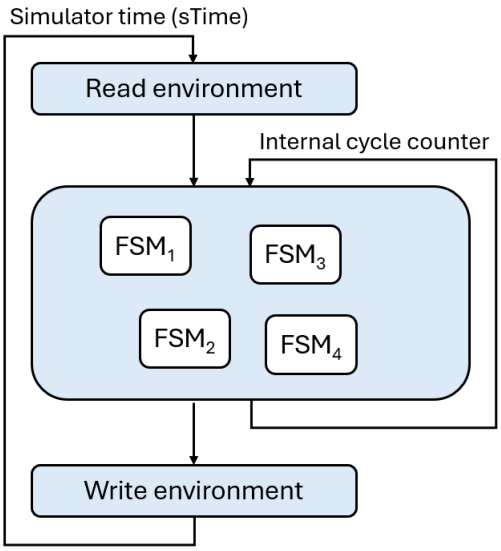


Figura 5.1: Comptador de cicles interns de les EFSM dins d'un simulador

5.2 La plataforma de simulació

La plataforma de simulació desenvolupada pel nostre grup de recerca, i adaptada per a la solució de [27] implementa el control de la simulació amb el bucle exterior per al pas del temps i el bucle interior pels cicles interns. Està desenvolupada en llenguatge Lua amb dues versions: una amb un simulador propi totalment desenvolupat el Lua, i una altra versió integrada amb el simulador CoppeliaSim[30]. Aquesta darrera versió permet fer simulacions en un entorn realista amb models de robots basats en robots reals. En tots dos casos el bucle principal de simulació és una implementació en Lua de l'Algorisme 2.

Abans del bucle principal s'inicialitza l'interval del pas de temps del simulador (línia 2) i la variable de temps del simulador (línia 3). Dins del bucle principal s'inicialitza el comptador de cicles (línia 7) i es llegeix la informació de l'entorn amb *Read_Environment* abans d'entrar al bucle intern. Fixem-nos que dins d'aquest bucle no s'incrementa el temps de simulació *simulation_time*. Dins d'aquest bucle el primer que es fa és fer que les EFSM llegeixin els missatges d'altres agents (línia 11). La funció *step()* s'encarregarà de processar aquests missatges i calcular els nous valors de les variables que defineixen l'estat intern, i també posa a la cua de sortida els missatges que s'enviaran als altres agents.

És important remarcar que les variables d'estat intern de les màquines romanen inalterables fins al final de cicle intern. En aquest moment *update()* les actualitzarà amb els nous valors calculats a *step()*. Això es fa després que *write_outputs()* hagi enviat els missatges a les altres màquines d'estat. Així es permet la generació de codi amb independència d'ordre entre les d'instruccions, tal com s'explica a [29]. Això vol dir que internament les màquines d'estat guarden tres versions de les seves variables internes: l'estat previ, l'actual, i el pròxim. Per exemple, un controlador pot guardar en una variable *State.curr* el valor "FREE". Dins del cicle actual *step()* cacula que canviarà d'estat al pro-

per cicle, així que ho guarda com a valor pel proper cicle *State.next* = "BUSY". Una vegada calculats els valors del proper cicle intern, s'envien els missatges pendents d'enviar, cap als altres agents (línia 13), i la funció *update()* actualitza les variables de la màquina d'estats amb els valors que tindrà al proper cicle. En l'exemple anterior *State.prev = "FREE"* i *State.curr = "BUSY"*. Així mantenim la sincronització entre totes les màquines d'estats, totes estan al mateix cicle i canvien els seus valors interns a cada pas d'aquest cicle intern.

Un altre aspecte important que cal tenir present és quan sortim d'aquest bucle de cicles interns perquè torni a córrer el temps del bucle principal. La resposta la tenim a la variable *internal_changes*. En el moment d'actualitzar les variables a *update()* les màquines d'estats retornen *true* si alguna de les variables que actualitzen ha variat durant el cicle actual. Si és així vol dir que encara no s'ha arribat a un punt d'estabilitat. A l'exemple d'abans, com que tenim una màquina d'estats on l'estat actual (FREE) és diferent del proper estat (BUSY) la funció *update()* mantindrà activada la variable *internal_changes* i el bucle intern continuarà, com a mínim, un cicle més. Quan el conjunt de màquines d'estat arribin al punt d'estabilitat, és a dir, sense canvis als seus estats, *update()* desactivarà *internal_changes* i el simulador sortirà del bucle intern. Abans de tornar al principi del bucle principal *write_environment* s'encarrega de fer totes les modificacions que afecten a l'entorn.

Algorisme 2 Control del temps de simulació i sincronització

```
1:  startTime ← GET_REALTIME()
2:  dt ← 0.05
3:  simulation_time ← 0
4:  INIT()
5:  while true do
6:      first ← true
7:      n_cycle ← 1
8:      internal_changes ← true
9:      READ_ENVIRONMENT()
10:     while internal_changes do
11:         READ_INPUTS(first)
12:         STEP()
13:         WRITE_OUTPUTS(first)
14:         internal_changes ← UPDATE()
15:         first ← false
16:         n_cycle ← n_cycle + 1
17:     end while
18:     WRITE_ENVIRONMENT()
19:     while (GET_REALTIME() - startTime < dt) do
20:     end while
21:     simulation_time ← simulation_time + dt
22:     startTime ← GET_REALTIME()
23: end while
```

Just abans de fer avançar el temps de simulació (línia 21) tenim un bucle d'espera (línies 19 i 20). Aquest només està implementat en els casos en que volem que el temps del simulador es sincronitzi amb el temps real. Si no fem

aquesta espera tindrem un temps de simulació final de n segons tot i que la simulació s'haurà executat en un temps de CPU molt menor. Per exemple, potser tenim totes les dades d'una simulació on s'han transportat 252 tasques en 40 minuts, tot i que el temps de CPU emprat en fer la simulació ha estat de 5 minuts. En el cas dels agents integrats en CoppeliaSim, que es mouen en un entorn realista i observable en temps real ens interessa aquesta sincronització entre el temps de simulació i el temps real. Per altra banda, si volem veure com es comporta el sistema fent la mitjana de moltes simulacions, ens interessa que les simulacions s'executin en el menor temps de CPU possible.

A la taula 5.1 podem veure un petit exemple d'execució d'una simulació. A la columna *time* tenim el valor del temps, la variable *simulation_time* del bucle principal, en segons. El comptador de cicles *cycle* està a la columna del costat. No es mostren els cicles on el sistema està estable i no hi ha canvis. En aquest exemple tenim un únic *manager* i dos executors, que anomenem M_1, E_1, E_2 respectivament.

Inicialment, als 0.05 segons (recordem que $dt = 0.05\ s$) E_1 i E_2 estan a l'estat inicial. Com que no hi ha canvis en el seu estat intern, passa el temps, esperant algun esdeveniment a l'entorn. Als 3.1s entra la ordre de transport que M ha de gestionar. En aquest moment M_1 ha d'iniciar la seva primera subhasta, així que s'atura el temps i entrem al bucle intern. El *manager* s'espera un cicle fins el 2 abans d'enviar el CFP d'inici de subhasta. A la taula es mostra aquest enviament al segon cicle indicant el destinatari del missatge i el tipus de senyal (*All, CFP*). Aquesta espera d'un cicle es fa per sincronització amb els altres *managers*. L'entrada d'un nou *manager* fa que la resta, si poden, enviïn un nou CFP per repetir la subhasta, al següent cicle. Si M_1 ja envia el CFP al primer cicle, el senyal dels altres aniria amb un cicle de retràs. En una implementació on es recullen els PRP dins un cicle, M_1 no quedaria agrupat amb la resta de subhastes.

Un cop enviat el senyal, al cicle 3 M_1 passa a l'estat *Get* esperant rebre les ofertes dels executors. Al cicle 4 els dos executors envien les seves propostes a M_1, i quan finalitza l'espera de propostes, al cicle 7 M_1 avalua les propostes i envia el senyal *Accept* al guanyador (E_1). L'exemple acaba en el moment en que M_1 està esperant (*Wait*) que E_1 (ocupat, en estat *Busy*) el reculli. Arribats a aquest punt (cicle 10) ja no hi ha canvis als agents,així que es surt del bucle intern, s'actualitza el que calgui de l'entorn i el temps avança. To t i que no es mostra a la taula, en aquest exemple el temps avança fins als 4 segons, moment en que entra un nou *Manager*, i torna a aturar-se el temps amb la subhasta que inicia el nou agent, i M_1 repeteix la subhasta.

5.2.1 Entrada dinàmica d'agents i activació de subhastes

El problema que estem tractant en aquest treball requereix que es pugin produir al llarg del temps esdeveniments en l'entorn que afectin a l'assignació de tasques. El més habitual és l'entrada de forma dinàmica de noves tasques de transport. I també la inclusió d'un nou robot al sistema, a més de qualsevol altre esdeveniment que pugui ser del nostre interès.

Per això la plataforma de simulació inclou els missatges de tipus CHANGE, que s'utilitzen per notificar als agents qualsevol canvi a l'entorn. Aquests missatges s'entreguen als agents com si fossin senyals provinents dels controladors de les capes inferiors. En el cas de la capa deliberativa dels *managers* es reben

		$Manager_1(M_1)$		$Executor_1(E_1)$		$Executor_2(E_2)$	
time	cycle	curState	MsgOut	curState	MsgOut	curState	MsgOut
0,05	1			**Free**		**Free**	
3,1	1	**Init**		Free		Free	
3,1	2	Init	All,CFP	Free		Free	
3,1	3	**Get**		Free		Free	
3,1	4	Get		Free	M_1, Prp_1	Free	M_1, Prp_2
3,1	5	Get		Free		Free	
3,1	6	Get		Free		Free	
3,1	7	**Pri_auc**	E_1,Accept	Free		Free	
3,1	8	**Wait**		Free		Free	
3,1	9	Wait		Free		Free	
3,1	10	Wait		**Busy**		Free	

Taula 5.1: Exemple del control d'execució en una subhasta simple

com a senyals RC de la capa reactiva (*Reactive Command*). D'entrada, cada vegada que entra un nou agent al sistema s'envia un CHANGE.

Els missatges CHANGE també els poden enviar els agents per indicar algun canvi produït per ells mateixos. Així que s'utilitzen per avisar que s'han de d'executar noves subhastes. En el cas de la solució *Greedy*, a més del moment en que es crea l'agent, s'envia en aquests moments:

- Temps d'espera de recollida del *manager* sobrepassat.
- El *manager* ha perdut la subhasta, o no hi ha participat.
- S'ha completat la tasca de descàrrega.

Quan un CHANGE arriba a un *manager* aquest el transforma en un senyal RC en funció de l'estat actual:

- Agent en estat FREE: $RC =' Call$"
- Agent en estat WAIT: $RC =$ "$Re - call$"

En el cas d'aquest treball si l'estat és INIT es genera un senyal RC "First-call". Així es crea l'espera d'un cicle que hem vist a la taula 5.1. A la solució *greedy* en el moment de l'entrada de la tasca, a més del CHANGE, ja s'envia el CFP. En el cas de la solució de subhastes agrupades només s'envia el CHANGE. Aquest provocarà, al cicle següent, rebre un senyal "First-call" de la capa reactiva i això iniciarà la primera subhasta.

5.2.2 Descripció de l'escenari d'execució

A l'hora de descriure el magatzem, en les dues versions (una implementada totalment en Lua, l'altra integrada a CoppeliaSim) la nostra plataforma permet introduir la descripció (*layout*) de l'escenari de simulació en forma de fitxer de text, en veiem un exemple a la figura 5.2.

El mapa està dividit en cel·les, per això al fitxer de text es descriu cada cel·la amb un caràcter específic. El significat d'aquests caràcters és el següent:

- Cel·la "carretera": anomenem així a les cel·les per on poden circular els agents. Aquí podem tenir diversos caràcters en funció de si el sentit de circulació és cap amunt, avall, esquerra o dreta. Aquests caràcters són "A", "V", "<", ">", respectivament.
- Ports de càrrega/descàrrega: es representen amb el símbol "@".
- Zona sense trànsit: per aquí no poden circular els agents, es simbolitza amb un "."quan volem que al simulador realista es representi com una cel·la buida, de color diferent, o bé amb un "#"si volem que la cel·la s'ompli amb un cub (per exemple, per dibuixar parets posant aquest símbol en cel·les consecutives).

Tots els ports que hem posat a les cel·les del mapa ("@") es poden configurar just després de descriure el *layout*. Només cal afegir al fitxer una fila començant per "@", seguit d'un número. Aquest identifica el port al mapa, Començant a comptar-los des de la fila superior i la primera columna a l'esquerra.

A continuació posem el nom que donem a aquest port. Per exemple, a la figura img:LayoutText "@1 Charge1"està indicant que el primer port que trobem a la part superior del mapa (després del darrer ".") és el port de recàrrega "Charge1". Després del nom del port es poden posar els noms dels ports que són

```
##############################
#.............@@@@@@@@@@#
#.............V<<<<<<<<<<#
#.VV<<<<<<<V<<<<<<<V<<<<<<<<#
#.VV@@@A@@@V@@@A@@@V@@@A@@@A#
#@VV@@@A@@@V@@@A@@@V@@@A@@@A#
#.>>>>>A>>>>>>>A>>>>>>>A>>>A#
#.VV@@@A@@@V@@@A@@@V@@@A@@@A#
#@VV@@@A@@@V@@@A@@@V@@@A@@@A#
#.VV<<<<<<<V<<<<<<<V<<<<<<<<#
#.VV@@@A@@@V@@@A@@@V@@@A@@@A#
#@VV@@@A@@@V@@@A@@@V@@@A@@@A#
#.>>>>>A>>>>>>>A>>>>>>>A>>>A#
#.>>>>>A>>>>>>>A>>>>>>>A>>>A#
#....@...@...@...@...@...@..#
##############################
@1 Charge1
@2 Charge2
@3 Charge3
@4 Charge4
@5 Charge5
@6 Charge6
@7 Charge7
@8 Charge8
@9 Charge9
@10 Charge10
@11 Charge11
@12 A1 Pick
```

Figura 5.2: Descripció de l'entorn de simulació (fragment)

possibles destinacions d'aquest, separats per un espai en blanc. El nom del port desté pot contenir només una part del nom, per indicar que el destinatari són

tots els ports que comencen amb aquest nom. Per exemple "@12 A1 Pick"ens indica que des del port A1 s'ha d'anar a qualsevol dels punts de recollida. Si no es posen destinataris tots els ports poden ser ports de destinació.

A partir d'aquest fitxer de descripció l'aplicació genera una estructura complexa de dades, la base de la qual és similar a un graf dirigit, que recull tota aquesta informació. En aquesta estructura els ports tenen coordenades, calculades a partir de la posició de la cel·la al fitxer de text, i la seva mida a la configuració del sistema. Podem veure els ports com a nodes d'un graf, aquests ports tenen "portes", que són les cel·les transitables adjacents. Això es deu a que els ports no són transitables, per això necessiten tenir al costat una cel·la transitable per poder-hi arribar a l'hora de la càrrega o descàrrega. En funció del senti de circulació de les portes tindrem arcs d'entrada, de sortida, en un sentit o un altre. Cada node corresponent a una cel·la transitable està lligat a una cel·la adjacent a la que es pot accedir respectant el sentit de circulació (per exemple, no té sentit anar des d'una ">"a una "<").

A la versió actual els agents es basen en aquesta estructura de dades per fer els seus càlculs de rutes mitjançant l'algorisme de Dijkstra[12].

La versió de l'aplicació integrada en CoppeliaSim genera una representació tridimensional com la que es mostra a la figura 5.3.

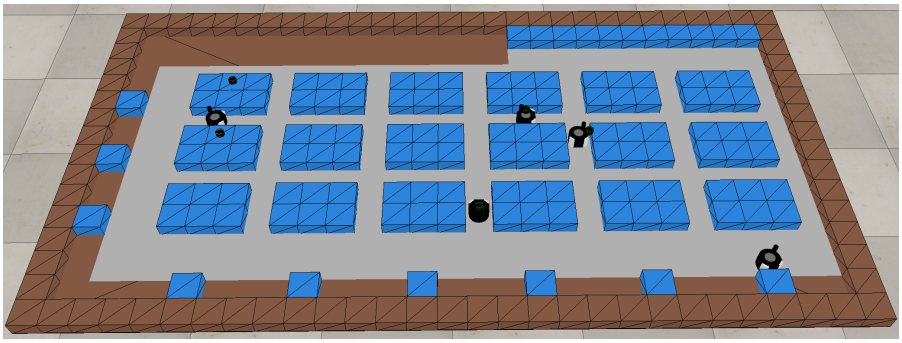


Figura 5.3: Representació tridimensional en CoppeliaSim

5.3 Integració de robots reals i virtuals

Anteriorment hem vist com es sincronitzen els diferents agents del sistema dins de la plataforma de simulació, amb l'algorisme 2 hem explicat con les màquines d'estats que s'executen dins el sistema comparteixen comptador de cicles i comptador de temps de la simulació. A més, l'algorisme permet executar la simulació a temps real.

En el cas de la plataforma totalment implementada en Lua l'entorn on es mouen els agents no és més que un mapa simbòlic, l'estructura de dades que hem vist com es pot generar des d'un fitxer de text. I els robots són una implementació de les 4 capes del nostre model d'agents amb una capa física (L0) molt simple.

Pel que fa a la plataforma integrada a CoppeliaSim l'enton ja no és simbòlic. Tot i que simulat, té propietats físiques, mesures realistes, etc. I la capa L0

s'executa en un robot virtual que és un model realista d'un robot real. I, encara més, CoppeliaSim permet controlar robots des de l'entorn de simulació. Amb això podríem simular el sistema de transport complert.

Anant més enllà, podríem tenir un sistema reals, amb robots reals on cada robot tingués un "bessó virtual", és a dir, un robot virtual dins de l'etorn simulat que reflectís dins l'entorn simulat tot el que fa el robot real. La capa L0 del nostre model d'agent també s'executaria en els robots reals. A la figura 5.4 veiem com s'integren els robots virtuals i/o reals amb les plataformes de simulació.

A la figura hi apareix un element central entre els diferents tipus de robot i l'agent simulat, la capa d'enllaç L0-L1 o sincronitzador (*synchronizers*) [8][9]. Aquesta capa permet enviar i rebre missatges entre el controlador de la capa L1 (navegador o gestor de trànsit) i la L0 que s'executa al robot, a més de sincronitzar el temps de simulació amb el temps real en que operen els robots. A més el sincronitzador pot enviar informació recollida de l'entorn simulat cap a l'entorn real i en sentit contrari. Així, per exemple, l'agent simulat pot "avisar"al robot real d'algun aspecte de l'entorn que aquest encara no ha detectat al món real.

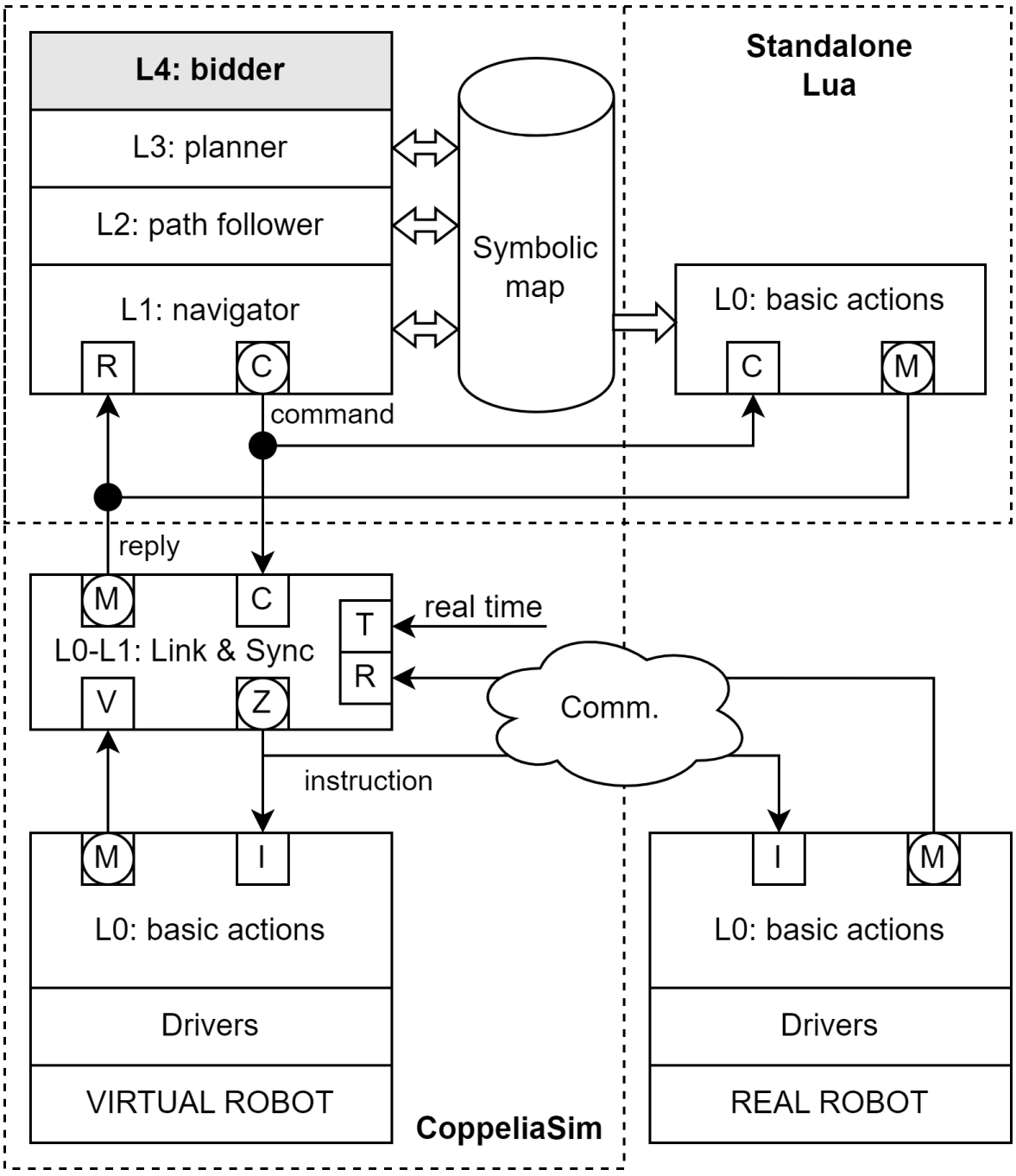


Figura 5.4: Sincronització entre reobots reals i virtuals

Capítol 6

Resultats

6.1 Escenari: model de magatzem automatitzat

Per testar i avaluar la solució proposada s'han fet les simulacions en el mateix entorn en què D. Rivas [27] va contrastar la seva solució amb subhastes repetitives amb les assignacions fetes amb CPLEX [18]. L'entorn simula un magatzem de mida mitjana com que el que es descriu a [37], però una mica més petit. En aquest cas, la planta del magatzem té unes dimensions de 27m x 14m on hi ha 5 àrees d'emmagatzematge (de la A a la E), tal com es mostra a la figura 6.1 En iniciar la simulació els robots es troben a la part superior, que és la zona de recàrrega, i, per a cada tasca, han de d'anar al punt de recollida, que és en una de les 5 àrees d'emmagatzematge, i portar la càrrega a un dels punts de recollida (*picking stations*).

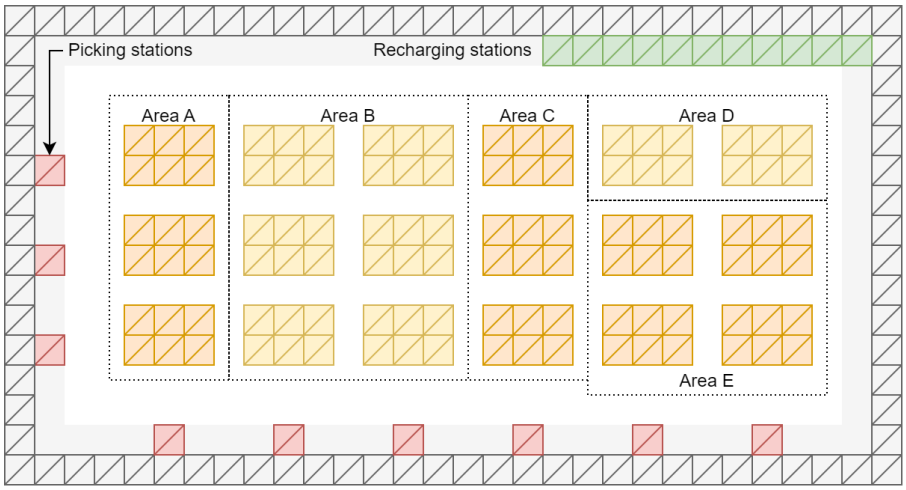


Figura 6.1: Model de magatzem per a les simulacions

6.1.1 Resultats del treball previ de D. Rivas [27]

En aquest treball previ, per tenir una perspectiva de la qualitat dels resultats obtinguts amb la solució basada en subhastes comparen els resultats amb els de la solució òptima del model MILP [13], executat amb CPLEX.

Per a la comparació entre el MODEL MILP i el model de subhastes es van fer execucions amb grups de 24, 50, 100 o 252 tasques. Era la simulació d'un sistema no dinàmic on totes les tasques venien donades per lots, conegudes prèviament. Es va fer aquesta simplificació per reduir la complexitat i permetre a CPLEX trobar la solució òptima en un temps relativament curt. Atès que al model MILP es van entrar els ports de l'escenari i les distàncies mínimes entre aquests, els costos resultants són de solucions ideals. No hi ha interferències entre agents, ni moviments aleatoris, etc. Per al mètode de subhastes es van calcular els costos estimats, és a dir, la suma d'ofertes guanyadores dels executors a les subhastes.

Pel que fa a la solució del sistema multiagent es va executar la versió sense repetició de subhastes. Per tant, cada vegada que un robot rebia una assignació de tasca ja no la deixava fins al final del transport.

En el cas de 25 tasques el cost de les subhastes era un 17% superior a l'òptim, i en el cas de 252 tasques ja era només un 9% superior aproximadament. Aquests resultats van ser encoratjadors perquè el cas de 252 tasques per hora és que s'aproxima més a un entorn logístic real. El més important és que, ,exceptuant el cas de 25 tasques, el mètode MILP no va poder trobar l'assignació òptima en una hora i va retornar la millor solució que havia trobat fins aquell moment. Aquest mètode resulta ineficaç, tenint en compte que en un cas real entrarien unes 25 tasques cada 5 minuts. Al mateix temps, la solució basada en subhastes entre agents trigava poc més de dos minuts de mitjana per resoldre el cas de 252 tasques. El treball també deixa clar que pretén minimitzar costos locals, estratègia *greedy*, no podent arribar mai a l'òptim d'una solució exhaustiva MILP.

Un altre aspecte negatiu del mètode d'optimització lineal és que el model busca minimitzar el cost de la solució i el nombre màxim de tasques robot. I proporciona resultats que compleixen aquests requisits però potser no són atractius per als usuaris d'aplicacions reals. Per a les solucions òptimes requereix sempre més d'un 20% de robots respecte la quantitat de tasques. Per exemple 23 agents per 100 tasques.

El resum és que, tot i que *greedy* i subòptim, el sistema presenta millors resultats per a sistemes logístics amb un gran volum de tasques per hora que el mètode d'optimització lineal.

Després de la comparació respecte el mètode d'optimització es va analitzar l'impacte de la repetició de subhastes en els resultats, comparant la solució MRTA de subhasta única amb la de repetició de subhastes. En aquest cas el sistema llança subhastes primàries i secundàries cada vegada que es produeix un canvi que podria provocar una reassignació (finalització, arribada de nova tasca, finalització del temps d'espera al lloc de càrrega,...). En aquest cas la simulació s'aturava al completar les 252 tasques. Els resultats tornen a ser positius, amb una millora del 8% pel que fa al cost mig de les tasques, i entre un 5 i un 6% del temps mitjà d'execució d'una tasca. També s'observa que el temps en completar tota la solució només baixa un 0,25% quan es passa de 5 a 10 robots, i conclouen que per un nombre elevat de robots el temps total de simulació depèn més de la separació temporal entre les tasques que el temps que es triga a completar les

tasques,

6.2 Variables i criteris de comparació de les solucions

Els valors obtinguts a les simulacions fetes amb Munkres* s'han comparat amb els resultats de les simulacions amb la solució de subhastes repetitives "egoistes"(*greedy*). Per fer aquesta comparació, a més de mostrar els valors obtinguts directament, calculem la variació del percentatge que es mostra a 6.1

$$VP = \frac{valor_I - valor_B}{valor_B} \times 100 \tag{6.1}$$

On $Valor_B$ és el valor base d'on es parteix per fer la comparació, i $Valor_I$ és el valor d'interès. Aquest valor representa el percentatge de creixement o decreixement respecte el valor inicial de la variable. En aquest cas el % de guany o pèrdua de la variable a la simulació de Munkres* respecte al valor anterior en la solució *Greedy*.

6.2.1 Agents participants en les subhastes

La principal diferència entre *Greedy* i Munkres* és l'estratègia utilitzada en les subhastes. En el primer cas aquestes no estan agrupades i cada tasca controla els participants i avalua els seus valors de licitació, i en el segon tenim subhastes agrupades, on totes les tasques comparteixen la informació de licitació de tots els agents. Per poder comparar aquestes dues estratègies pel que fa al nombre d'agents participants, es compten tots els participants dins d'un pas de temps de simulació. Per tant, en un *timestep* del simulador es compten tots els robots que estan licitant i totes les tasques que han iniciat subhasta.

6.2.2 Cost en metres recorreguts

La primera possible mètrica per mesurar el cost de la solució és la distància total recorreguda pels robots del sistema. Com que això inclou els robots que s'han mogut per anar a recollir una tasca, però no han arribat perquè un altre robot ha gunyat una subashta posterior podem veure com afecta el fet de compartir informació entre tasques al cost total. En el cas Munkres* l'assignació, tot i executar-se a cada agent-tasca, és global i com més agents participin en les subhastes més hauria de millorar el cost.

6.2.3 Cost en temps

Utilitzant el temps (en segons) com a mètrica podem calcular el cost de diverses formes. Des del punt de vista de les tasques, són importants el temps de servei i el temps en que es completen. Per tant, és interessant veure com es comporta la nova solució calculant el cost del sistema en funció d'aquestes variables:

- $T_{satisfy} = t_{unload} - t_{init}$. És el temps de servei de la tasca, des de que entra al sistema (T_{init}) fins que es descarrega al punt de destinació (T_{unload}).

- $T_{complete} = t_{unload} - t_{assign}$. És el temps que en que es completa una tasca des de que s'assigna per primer cop a un robot (t_{assign}) fins que es descarrega.

6.2.4 Cost de les comunicacions

La millora del CNP proposada a [27] implica l'execucio de múltiples subhastes independents, i la repetició d'aquestes a partir de certs events que entren al sistema. Amb Munkres* les subhastes estan coordinades, encara que es puguin executar diverses instàncies del mateix algorisme d'assignació. Això hauria d'implicar una reducció considerable de les comunicacions (encara que suposi un cost extra de computació local). Per això el nombre de missatges enviats i rebuts, per tasques i robots, és una altra mètrica important per avaluar la solució.

Cal tenir en compte que la reducció en el nombre de missatges a transmetre facilita poder construir solucions plenament distribuïdes, on cada agent s'executa en un node (ordinador o robot) diferent del sistema.

6.3 Planificació fixa

Per contrastar els resultats amb els obtinguts per [27] s'han fet múltiples simulacions mantenint la taxa d'arribada d'ordres de transport de les 5 àrees del magatzem a 96, 75, 45, 12 i 24 per hora, respectivament com es descriu a [37]. Tenim, doncs, un model on arriben 252 tasques per hora, això generaria 504 ordres de transport (la meitat per portar la càrrega des del magatzem al punt de recollida, i l'altra meitat per tornar el prestatge o palet buit cap al magatzem).

Per avaluar la solució proposada en aquesta tesi es simulen les 252 ordres de transport consistents en la recollida dels articles als prestatges i el seu transport als punts de recollida. Per repetir les condicions exactes de simulació mentre només variem paràmetres de configuració del magatzem, el framework permet executar un pla concret on les 252 ordres entren al sistema en temps concrets i en l'ordre establert pel pla. Així es pot analitzar com es comporten les diferents solucions enfront la mateixa situació.

Hem realitzat múltiples simulacions on les mateixes 252 tasques entren al sistema, amb les dues solucions (*Greedy i Munkres**), i variant el nombre de robots de 5 a 10, tenint en compte que és un magatzem petit.

6.3.1 Agents participants en les subhastes

A la taula 6.1 veiem un resum de la participació dels robots, i dels agents associats a les tasques a les subhastes. Quan tenim pocs robots el nombre de tasques participants és molt elevat mentre que el nombre de robots és petit en les dues solucions. Això és normal perquè les tasques s'acumulen i estan esperant a ser servides pels pocs robots que hi ha. En que sí que és rellevant pel que fa al cost de les dues solucions és que en el cas de Munkres* participen un 10% o més de robots a les subhastes a partir de 7 robots (amb 5 o 6 robots només un 6% però també és important). Com que en el cas de Munkres* les subhastes són agrupades, i es fa una assignació basada en el mètode hongarès, la participació de més robots i més tasques permet formar una matriu d'assignació

d'rodre $m \cdot n$ on m és el nombre de tasques (subhastes) i n el nombre de robots (columnes). En el cas de la solució *Greedy* parcitipen un nombre menor de robots perquè quan una tasca té assignat un robot, aquest només participa en les "re-subhastes" d'aquesta tasca.

	Greedy		Munkres*		Variació %	
Robots	Tasques	Robots	Tasques	Robots	% Tasques	% Robots
5	25,13	2,16	27,83	2,29	10,74	6,13
6	18,51	2,47	20,27	2,63	9,49	6,42
7	13,76	2,75	14,73	3,02	7,06	9,91
8	9,92	3,10	10,70	3,47	7,85	11,98
9	6,75	3,54	7,40	4,00	9,51	13,12
10	4,77	4,18	5,42	4,72	13,73	12,92

Taula 6.1: Agents participants a les subhastes

6.3.2 Cost en distància (metres)

A la taula 6.2 veiem resumit el cost de les dues solucions en distància recorreguda, metres en aquest cas. Les diferències van des duns 231m (3,2% de millora) fins als 420m (6,14% de millora). Recordem que aquesta distància és la suma de les distàncies recorregudes pels robots que han estat assignats a les tasques. Aquesta millora del cost en distància és el resultat directe de la millor assignació que fa el mètode hongarès, tenint en compte que el valor amb que els robots participen a les subhastes és el cost en metres, és a dir, la distància fins al punt de recollida de la tasca més la distància fins al punt de descàrrega.

Robots	*Greedy*	Munkres*	Variació %
5,00	6835,00	6415,00	-6,14
6,00	6970,00	6668,00	-4,33
7,00	7278,00	7047,50	-3,17
8,00	7653,00	7391,00	-3,42
9,00	8056,50	7794,00	-3,26
10,00	8452,00	8107,50	-4,08

Taula 6.2: Cost en metres recorreguts pels robots

6.3.3 Temps de servei i de completar la tasca

Des del punt de vista de les tasques la taula 6.3 ens proporciona una mesura del cost en funció del temps en que aquestes són ateses i servides. Primerament, en tots els casos, el temps en completar les tasques (des que s'assignen un robot fins que descarreguen) és entre un 15% i un 17% menor en el cas de Munkres*. Això es trasllada al temps de servei, és a dir, el temps en que triguen a completar-se des que entren el sistema. Tenim fins a un guany del 10,8% amb 8 robots. Hi ha un cas una mica especial a comentar. En el cas de 5 robots el temps de servei és un 0,25% major amb Munkres*. Això es deu a que tenim pocs robots, i les

tasques queden "en espera"en els dos casos. Com que Greedy és egoista quan la primera tasca s'assigna un robot ja no perd l'assignació i, per tant, acaba abans. Com a molt podria canviar de robot en una "re-subhasta", però seguiria assignada (el robot assignat només participa a subhastes d'aquesta tasca). Així i tot la variació del cost tendeix al 0% i en quant afegim més robots ja passa a ser millor Munkres*.

	Greedy		Munkres*		Variació %	
Robots	tServei	tCompletar	tServei	tCompletar	% tServei	% tCompletar
5,00	52033,83	15085,63	52163,00	12657,50	0,25	-16,10
6,00	43406,65	15434,05	41033,65	13109,40	-5,47	-15,06
7,00	36352,60	15956,50	35043,72	13537,55	-3,60	-15,16
8,00	32310,50	16590,80	28842,50	14039,30	-10,73	-15,38
9,00	26755,05	17239,97	25032,25	14349,72	-6,44	-16,76
10,00	22008,02	17919,80	21131,25	14744,12	-3,98	-17,72

Taula 6.3: Temps de servei i en completar les tasques (en segons)

6.3.4 Comunicacions

Els resultats pel que fa a les comunicacions es mostren separats en dues taules: a la taula 6.4 hi ha el resum de missatges enviats i rebuts per les tasques, i a la taula 6.5 el mateix en el cas dels robots.

En aquest cas hi ha una reducció propera al 70% pel que fa als missatges enviats per les tasques i rebuts pels robots, i propera al 80% dels missatges enviats pels robots i rebuts per les tasques. Això demostra que l'execució d'un únic procés d'assignació amb les dades agrupades, encara que s'executi a diverses unitats d'execució simultàniament amb exactament les mateixes dades, redueix considerablement les interaccions entre els agents pel que fa a la coordinació.

Amb Munkres* el procés d'assignació fa una assignació òptima en un cert temps d'execució, no es fan n assignacions a partir de n subhastes. El mateix procés ja indica als robots reassignats que han de deixar la tasca anterior, i aquests robots tampoc han de refusar tasques assignades perquè ja saben que estan rebent una assignació òptima. A més amb Munkres∗ es reaprofiten assignacions anteriors (si es manté el cost òptim), tal com es veu al capítol 3.3.2.

	Greedy		Munkres*		Variació %	
Robots	Enviats	Rebuts	Enviats	Rebuts	% Enviats	% Rebuts
5	79590,50	1831742,00	18626,00	191812,00	-76,60	-89,53
6	56885,00	1451763,00	13577,00	164968,00	-76,13	-88,64
7	40691,00	1023103,00	10156,50	142311,00	-75,04	-86,09
8	27883,00	724578,50	7865,00	124914,00	-71,79	-82,76
9	17545,50	484274,00	6098,50	108943,50	-65,24	-77,50
10	10226,00	290450,50	5109,50	98524,50	-50,03	-66,08

Taula 6.4: Missatges enviats i rebuts per les tasques

	Greedy		Munkres*		Variació %	
Robots	Enviats	Rebuts	Enviats	Rebuts	% Enviats	% Rebuts
5	275630,00	276051,00	50254,00	84567,00	-81,77	-69,37
6	227231,00	227658,00	42606,00	70812,00	-81,25	-68,90
7	184287,00	184765,50	35425,00	57975,00	-80,78	-68,62
8	140150,00	140690,00	28329,00	47226,00	-79,79	-66,43
9	94758,00	95347,00	22106,50	36284,00	-76,67	-61,95
10	60328,00	61032,50	17356,50	29430,50	-71,23	-51,78

Taula 6.5: Missatges enviats i rebuts pels robots

6.4 Distribució probabilística

Les simulacions que hem vist a 6.3 s'han executat sobre un pla d'execució fixat, on sempre arriben les mateixes 252 tasques en els mateixos instants de temps. Aquest pla d'execució es va utilitzar per fer simulacions en el cas de [27], i és útil per comparar les solucions ja que una distribució diferent de les tasques podria alterar les conclusions. I, a més, és útil per poder verificar manualment situacions concretes (sobretot en el cas de pocs robots, només cal tornar a repetir la simulació fins al punt que es vol estudiar). Així i tot, com que la distribució de les tasques al llarg del temps també pot ser determinant, hem fet també les simulacions generant els plans aleatòriament a la vegada que es manté l'escenari descrit a [37].

Per això s'ha afegit a l'entorn de simulació una opció per generar les tasques aleatòriament però complint les següents condicions, i s'han fet múltiples simulacions amb aquests plans aleatoris realistes:

- Arriben 252 tasques per hora.
- Punts de recollida aleatoris.
- Tasques a les àrees A, B, C, D, E del magatzem en quantitats 96, 75, 45, 12 i 24 per hora respectivament.
- Concentració de tasques amb oscil·lacions periòdiques de 5 minuts, segons la probabilitat $prob(x) = 0,5 + 0,5\cos(\frac{2\pi x}{300})$

A les taules següents es mostren els resultats (en mitjanes) de les simulacions executant la solució *Greedy* amb repetició de subhastes i la versió amb agrupació de subhastes i algorisme d'assignació de *Munkres*. Els resultats de l'execució amb múltiples plans aleatoris confirma els resultats vistos a 6.3, amb petites variacions degudes a la particularitat d'aquell pla d'execució concret.

Agents per subhasta

A la taula 6.6 els valors confirmen el que ja hem vist a la taula 6.1. Les columnes de tasques mostren el nombre mitjà de tasques que participen a les subhastes, i les columnes de robots el nombre de robots que hi participen. En el cas dels robots aquests participen més a les subhastes, i això hauria de permetre fer una millor assignació amb el mètode de *Munkres*.

	Greedy		Munkres*		Variació %	
Robots	Tasques	Robots	Tasques	Robots	Tasques	Robots
5	25,94	2,16	28,67	2,28	10,54	5,75
6	19,47	2,44	20,52	2,61	5,37	7,07
7	14,33	2,76	15,35	3,03	7,16	9,71
8	11,06	3,14	11,26	3,49	1,82	11,34
9	7,48	3,66	7,64	4,08	2,11	11,59
10	6,22	4,18	6,51	4,70	4,74	12,56

Taula 6.6: Agents per subhasta

Cost de la solució en metres recorreguts

La taula 6.7 ens mostra una tendència a la reducció de la diferència a mesura que augmenta el nombre de robots. Cal tenir en compte que la solució *Greedy*, en aquest cas (252 tasques) només diferia un 9% [27] de la solució òptima amb 10 agents. A diferència de la taula 6.2 aquí el descens de la variació de % és progressiu perquè tenim la mitja de simulacions de casos diversos. En el cas de la taula 6.2 una concentració concreta de tasques en un període pot fer que el mètode hongarès aprofiti més la informació compartida. Aquest descens de la diferència de cost entre les dues solucions és raonable a mesura que augmenta el nombre de robots i no hi ha concentració d'arribada de tasques, doncs podem tenir subhastes on poques tasques tenen ofertes de molts robots. En qualsevol cas, en valors absoluts la diferència en metres va des de més de 300 en el cas de 5 robots fins als 250 en el cas de 10 robots, Això en menys d'una hora, uns 6000 o 7000 metres al dia.

Robots	Greedy	Munkres*	Var. %
5	6839,50	6477,00	-5,30
6	7119,00	6786,00	-4,68
7	7378,00	7008,50	-5,01
8	7654,00	7405,50	-3,25
9	7928,00	7675,00	-3,19
10	8156,50	7912,00	-3,00

Taula 6.7: Cost en metres recorreguts

6.4.1 Temps de servei i en completar la tasca

La taula 6.8 consolida el que havíem vist a la taula 6.3. En primer lloc en tots els casos el temps en completar una tasca ha baixat entre un 15 i un 16%. Es confirma que amb millors assignacions en distància més ràpid és el transport.

baixat entre un 15 i un 16%. Es confirma que amb millors assignacions en distància més ràpid és el transport.

En el cas del temps de servei es confirma que les reassignacions poden afectar considerablement al temps de servei al augmentar el temps d'espera d'assignació de les tasques. En el cas de la taula 6.3 el % de variació en el cas del temps de

servei tenia oscil·lacions al ser un cas particular de distribució. En la mitjana de simulacions amb plans aleatoris la millora d'aquest paràmetre augmenta amb el nombre de robots utilitzats. Ara bé, en el cas de només 5 robots el temps de servei és un 3,67% pitjor. Tenim doncs, que quan hi ha pocs robots i saturació de tasques en espera, aquestes "perden"els robots assignats amb la repetició de subhastes. La gràfica de la figura 6.2 ens resumeix aquesta millora.

	Greedy		*Munkres**		Variació %	
Robots	tServei	tCompletar	tServei	tCompletar	tServei	tCompletar
5	57490,98	15105,25	59600,78	12736,00	3,67	-15,68
6	47485,95	15559,80	46957,65	13211,78	-1,11	-15,09
7	40286,82	16113,62	38780,10	13697,62	-3,74	-14,99
8	34955,12	16597,77	32858,87	14056,62	-6,00	-15,31
9	28972,97	17077,60	26909,10	14464,47	-7,12	-15,30
10	25605,80	17444,85	23162,77	14717,02	-9,54	-15,64

Taula 6.8: Cost en temps de transport i temps de servei en segons

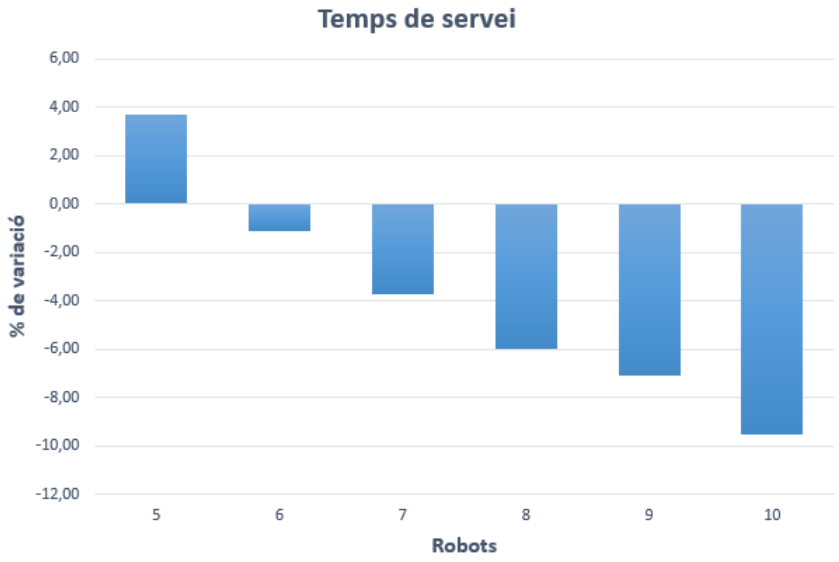


Figura 6.2: Variació del temps de servei

6.4.2 Comunicacions dels plans d'execució aleatoris

Les taules 6.9 i 6.10 mostren la mitja de missatges enviats i rebuts pels *managers* i robots respectivament. Els valors no només confirmen el que hem vist a les taules 6.4 i 6.5 sinó que, com a mitjana d'un major nombre d'execucions aleatòries, milloren els resultats.

	Greedy		Munkres*		Variació %	
Robots	Eviats	Rebuts	Enviats	Rebuts	Enviats	Rebuts
5	82678,00	2010840,00	19052,00	198493,00	-76,96	-90,13
6	59854,00	1597534,00	13471,00	163952,50	-77,49	-89,74
7	43038,50	1102980,50	10334,50	143510,50	-75,99	-86,99
8	32438,00	875246,00	8091,50	129012,50	-75,06	-85,26
9	19541,00	553302,50	6089,50	109537,50	-68,84	-80,20
10	15281,00	461572,50	5447,00	104869,00	-64,35	-77,28

Taula 6.9: Missatges enviats i rebuts pels gestors de tasques (*managers*)

	Greedy		*Munkres**		Variació %	
Robots	Eviats	Rebuts	Enviats	Rebuts	Enviats	Rebuts
5	279674,50	280093,00	51351,50	86297,50	-81,64	-69,19
6	240167,50	240580,50	42058,00	70160,00	-82,49	-70,84
7	191994,50	192478,50	35810,50	59297,00	-81,35	-69,19
8	159968,00	160496,50	29977,00	49037,00	-81,26	-69,45
9	107621,00	108151,00	21924,50	36583,50	-79,63	-66,17
10	91065,00	91694,00	20415,00	34109,50	-77,58	-62,80

Taula 6.10: Missatges enviats i rebuts pels robots

6.4.3 Impacte dels errors en les comunicacions

Fins ara hem constatat la millora dels resultats respecte la solució de [27]. A més, però, hem de veure si la nova solució segueix sent robusta enfront els errors a les comunicacions. Per verificar aquest aspecte la plataforma de simulació permet introduir la probabilitat de que els missatges es perdin en el moment d'enviar-los.

Hem simulat el transport de 50 tasques generades segons el model probabilístic descrit anteriorment. Les hem executat amb 5 i 10 robots robots al sistema. Les simulacions es fan amb probabilitats d'error del 0 fins al 60% en intervals del 10% i s'aturen quan es completen les 50 tasques. En tots aquests casos el sistema ha completat el transport de les tasques, demostrant la robustesa i fiabilitat de la solució. Vegem a continuació els resultats d'aquestes simulacions per veure si, tot i que el sistema és robust i segueix assolint l'objectiu, el temps o les condicions en que s'assoleix aquest objectiu està dins d'uns límits acceptables.

Temps de simulació

El temps total que triguen a executar-se les 50 tasques, pel que fa al temps simulat (no al temps del sistema) es veu afectat de forma més exponencial a mesura que augmenta la probabilitat d'error en el cas de 5 robots. Amb 10 robots l'augment no té tanta pendent. Això es pot veure gràfica de la figura 6.3.

L'eix horitzontal és la probabilitat d'error entre 0 i 1, l'eix vertical és el temps en segons. La gràfica mostra, per exemple, que entre 0 i 0,1 de probabilitat

d'errors la simulació només triga uns 150 segons més en el pitjor dels dos casos (5 robots), mentre que si la probabilitat d'error està entre el 0,5 i 0,6 la diferència ja és de 1000 segons, també amb 5 robots. Amb 10 robots la diferència és de una mica menys de 500 segons.

Així i tot, en qualsevol d'aquests casos el sistema segueix fent el transport de les 50 tasques tot i que es poden perdre més de la meitat dels missatges. Es un cas molt extrem on es triga 40 minuts més a fer el transport si treballem amb 5 robots. Amb pèrdua de missatges d'un 20% el transport només triga uns 4 minuts més (3 minuts amb 10 robots). Com que en els entorns de magatzems automatitzats no tindrem una pèrdua del 60% dels missatges el sistema és prou robust com per seguir fent el transport en temps raonable, per probabilitats d'error inferiors al 50%.

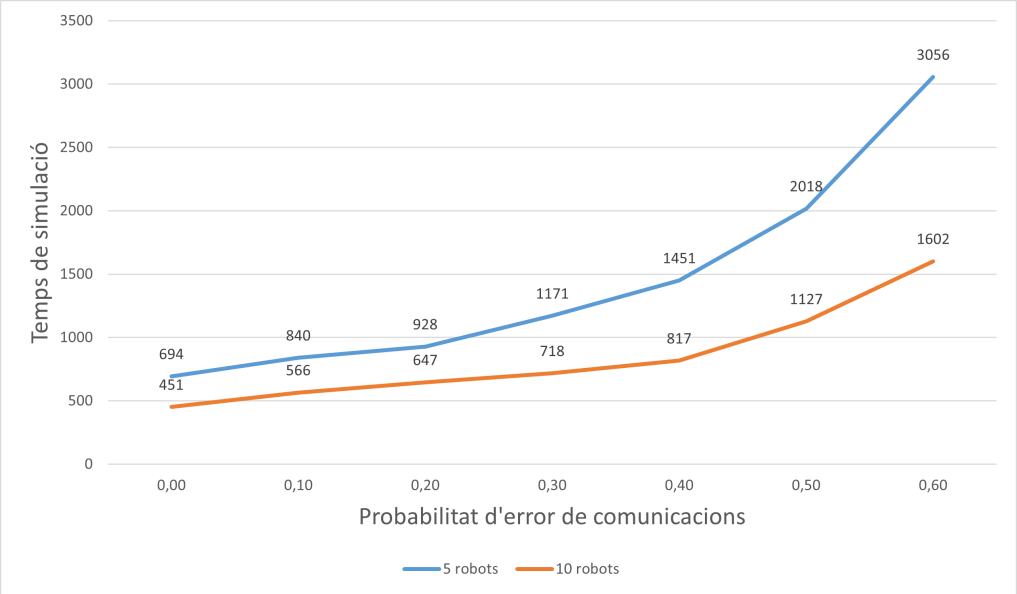


Figura 6.3: Impacte dels errors de comunicacions en el temps de simulació

Augment del temps de servei per tasca

Com que una de les mètriques usades en el càlcul del cost es basa en el temps de servei de les tasques, anem a veure com afecten els errors al temps de servei mitjà per tasca. A la gràfica de la figura 6.4 hi veiem l'increment del temps de servei mitjà de les tasques respecte a la solució sense errors de comunicacions. Es mostra pel casos de 5 i 10 robots.

Fins a la probabilitat del 40% el creixement del temps de servei pràcticament és igual en els dos casos. Això vol dir que el sistema és prou robust perquè amb 5 robots l'impacte dels errors sigui igual que amb 10. Tot i que es triga més amb 5 robots, per l'acumulació de tasques i el temps d'espera al tenir menys robots, l'augment de temps al introduir errors és proporcionalment el mateix que amb 10 robots. A partir del 50% el temps de servei s'incrementa molt més amb 5 robots.

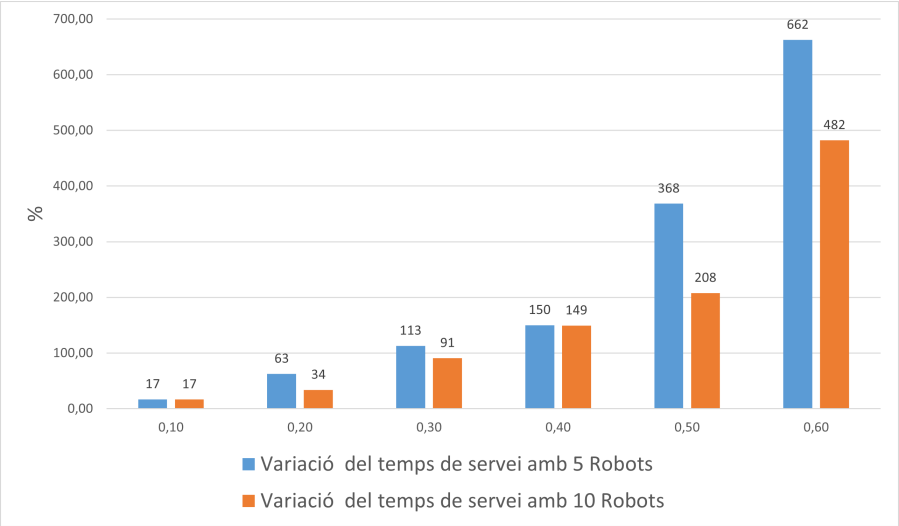


Figura 6.4: Impacte dels errors de comunicació en el temps de servei

Assignacions de tasques

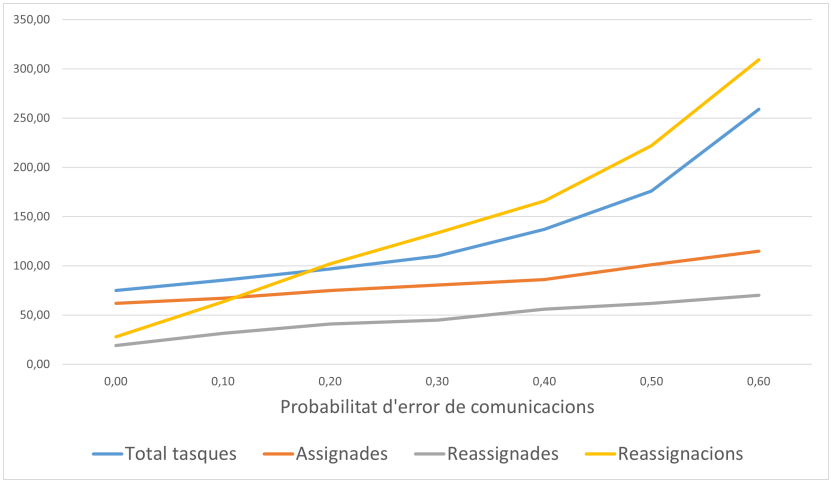
El sistema assoleix l'objectiu global de transportar les tasques, i ho fa en un temps raonable si la probabilitat d'error està per sota del 40%. Així i tot, cal analitzar el nombre de tasques que entren el sistema durant el temps de la simulació, si totes aquestes tasques són assignades, i quantes vegades són assignades o "reassignades".

A la gràfica de la figura 6.5 es mostra l'efecte dels errors de comunicacions en la quantitat de tasques assignades i els canvis d'assignació. Aquí l'eix horitzontal mostra la probabilitat d'error entre 0 i 1, i l'eix vertical és el nombre de tasques per cadascun dels 4 conceptes que mostra la gràfica. La gràfica superior és pel cas amb 5 robots i la inferior quan tenim 10 robots.

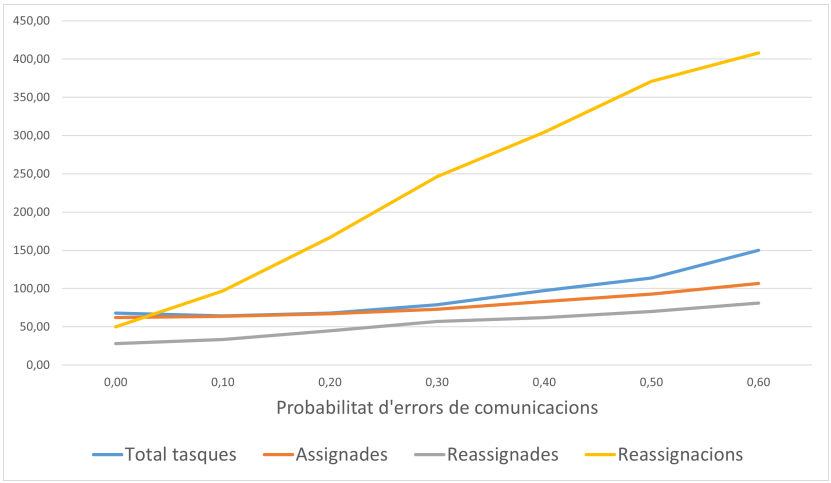
En primer lloc tenim el total de tasques arribades al sistema, recordem que en els testos d'error volem completar 50 tasques. si no hi ha errors de comunicacions entren 75 i 68 tasques per 5 i 10 robots respectivament ja que mentre els robots estan ocupats segueixen arribant tasques.

En el cas més extrem del test, amb pèrdues del 60% de missatges, arriben a entrar el sistema fins a 259 tasques amb 5 robots i 150 amb 10. Tot i aquesta diferència de tasques que arriben en els dos casos, el nombre de tasques assignades és pràcticament el mateix. Aquest valor és el nombre de tasques que han tingut un robot assignat en algun moment de la simulació. Per tant el sistema està assignant més o menys les mateixes tasques, tot i que en un dels casos pot trigar molt més a transportar-les degut als errors de comunicacions i a disposar de la meitat de robots. Aquí aconseguim dos aspectes clau dels sistemes multi-robot: robustesa i escalabilitat.

La variable "Reassignacions"és la que puja més a les gràfiques, i la que té major diferència entre el mínim i el màxim. Fa referència a tots els canvis d'assignació que hi ha hagut al sistema. Quan tenim 5 robots i sense errors, de mitjana es fan 28 reassignacions mentre que amb una probabilitat d'error



(a) Amb 5 robots



(b) Amb 10 robots

Figura 6.5: Tasques i assignacions en funció dels errors de comunicacions

de 0,6 es fan 309 reassignacions. Amb 10 robots aquest valor va de 50 a 408 reassignacions

Repartint el nombre de reassignacions entre les tasques assignades tenim una mitja de 0,45 i 0,8 reassignacions per tasca en el cas de l'execució sense errors en els casos de 5 i 10 robots respectivament. Això vol dir que les assignacions inicials són prou bones, degut a l'aplicació de Munkres* sobre subhastes agrupades, i no calen gaires reassignacions.

En el cas del 60% de pèrdua de missatges tenim 2,69 i 3,8 reassignacions per tasca (casos de 5 i 10 robots), unes 7 i 5 vegades més en cada cas. Al perdre's missatges alguns *timeouts* de les màquines d'estat (per exemple al sobrepassar el temps de recollida d'ofertes, estat GET) provoquen que es tornin a llançar subhastes, o bé algun missatges d'acceptació es perden, etc. Tot això fa augmentar el llançament de noves subhastes. A més, la pèrdua d'algunes

ofertes amb la informació de les subhastes del robot pot fer que s'executin per separat subhastes que es podrien haver agrupat o, encara més, que algun robot participi en agrupacions diferents de subhastes. En qualsevol cas l'algorisme és prou robust com per refusar assignacions quan calgui, mentre aplica l'algorisme d'optimització determinista a les subhastes que pot agrupar.

Capítol 7

Conclusions i treball futur

7.1 Conclusions

Aquest treball presenta la meva recerca en la millora dels costos d'operació de sistemes multi-robot. La contribució d'aquesta tesi rau en la definició i implementació d'un mètode híbrid per a l'assignació de tasques on es combina un mètode d'optimització determinista, que normalment no seria aplicable, amb un mètode basat en subhastes que és el que s'aplica habitualment per sistemes dinàmics, amb la intenció d'aconseguir una certa cooperació entre els agents per millorar l'eficiència del sistema.

Aprofitant la meva experiència prèvia en la recerca i implementació de sistemes multiagent [41][31] he implementat la solució de la meva recerca adaptant el sistema multiagent de la solució [27]. Aquest tipus d'entorn, de tipus industrial, també s'ajusta a la meva experiència prèvia de recerca i treball en la integració de sistemes heterogenis industrials mitjançant l'"agentificació" (transformació en sistemes multiagent) [39] [42].

He modificat la definició formal dels agents en forma de màquines d'estats, així el nou comportament dels agents fa que col·laborin generant l'agrupació de subhastes [38]. He incorporat una implementació pròpia del mètode Hongarès d'assignacions i he desenvolupat l'algorisme Munkres* [44]. Aquest aspecte és una contribució important, doncs en el treball anterior es destacava que els mètodes deterministes, tot i donar resultats òptims, són inviables per trobar la solució quan el nombre de tasques és nombrós. Degut a la seva lentitud, amb temps de càlcul de la solució molt superior al d'execució. Ara bé, en aquest treball Munkres* aplica el mètode hongarès només en agrupacions de subhastes, i pot ser distribuïble, cosa que el fa ràpid i robust.

Per obtenir els resultats i percentatges de millora comentats anteriorment he fet centenars d'experiments i simulacions amb el sistema MAS adaptat. Variant el nombre de robots, el nombre de tasques, la forma en que aquestes entren dinàmicament al sistema, etc. Per exemple, he fet un generador de tasques per la plataforma de simulació que permet generar plans d'execució de tasques on aquestes entren de forma aleatòria i complint uns criteris concrets. Aquests criteris poden ser, tasques per hora, funció de probabilitat oscil·lant perquè hi hagi períodes amb major arribada de tasques, àrees del magatzem amb més arribada de tasques, etcètera.

Al final, la solució que es proposa en aquest treball modifica les subhastes per incorporar informació d'altres agents i trobar solucions de manera cooperativa, que, amb un cost de treball relativament baix, aconsegueix millorar els costos d'operació respecte de la solució no cooperativa (*greedy*).

Aquesta recerca s'ha centrat en la seva aplicació en magatzems automatitzats. En aquests tipus de sistemes, el treball previ d'en Daniel Rivas [27] va demostrar que amb un mecanisme *greedy* de subhastes aplicat a l'assignació de tasques de transport a robots en magatzems es pot arribar a aconseguir costos d'operació i temps de servei molt propers als que donarien les solucions òptimes. En els casos en què es podia obtenir la solució òptima, la solució amb subhastes era només entre un 17 i un 9% pitjor.

La solució que es proposa en aquesta tesi aconsegueix millorar els costos globals de les assignacions fetes amb subhastes *greedy* amb els mateixos entorns. En el cas de la distància recorreguda s'arriba fins a una millora del 5%. En termes de temps de servei, la millora pot arribar fins a un 9%, tot i que hi ha casos en què no s'aconsegueix cap millora. Finalment, pel que fa al temps de transport, s'aconsegueix una millora al voltant del 15% en tots els casos de l'entorn d'experimentació.

Cal tenir en compte que, en la indústria i en els magatzems automatitzats, millores prop del 5% ja resulten interessants perquè redueixen els costos d'explotació de les plantes corresponents amb una inversió fàcilment recuperable.

En una altra línia de millora, el nombre de comunicacions (missatges enviats i rebuts) es redueix entre un 60 i un 90%, cosa que permetria distribuir la implementació del sistema. De fet, ara tenim agents gestors que s'estan executant en un únic node de computació i es poden comunicar amb robots físics. En baixar dràsticament la càrrega de comunicacions es pot redistribuir el sistema de manera que grups (o individus) d'agents gestors s'executin en nodes de computació diferents.

7.2 Treball futur

Com que els resultats són encoratjadors i la possibilitat de connectar robots reals al sistema és prou directa la primera línia de continuació seria explorar d'altres casos reals en empreses que hi puguin estar interessades. En aquest cas caldria treballar en la integració amb altres sistemes llegats (*legacy systems*), formalitzar màquines d'estats que s'adaptin a diversos tipus d'agents nous que entrarien el sistema. De fet, ja he participat en la implementació de sistemes multiagent en empreses existents [39][42][40].

Un dels aspectes que no s'ha tractat específicament és com s'avaluen els costos de transport. De fet, els robots calculen les rutes sense tenir en compte les esperes causades pel trànsit i en funció dels paràmetres del seu model físic.

En aquest darrer cas el càlcul dels costos es podria treballar a la capa física, fer prediccions de consum d'energia, optimització de moviments a baix nivell, com per exemple a [11] on s'incorpora un model predictiu del cost de desplaçament d'un robot.

Per tenir en compte el trànsit i les esperes que provoca es poden incorporar estratègies i algorismes de *Multi-Agent Pathfinding* (MAPF) [36] al *Pathfinder* dels robots.

Tot i que ja hem vist que l'ús de camins mínims acaba generant solucions

bones d'assignació segurament incorporar el MAPF per al càlcul dels costos permetria reduir encara una mica més els costos d'operació i els temps de servei.

Aplicant el MAPF podem determinar plans de ruta exclusius (en temps i espai) per a grups de robots. Així els costos que els robots proporcionarien a les subhastes, per exemple, podrien ser en temps de la ruta amb el nombre mínim de conflictes possible. Aquesta línia de recerca futura necessitaria l'avaluació i adaptació des d'algorismes clàssics de MAPF [33, 16] fins a algorismes basats en cerques en arbres de restriccions [32] o d'altres mètodes [46]. De fet, hi ha un interès creixent en aplicacions de MAPF en entorns similars o iguals al magatzem automatitzat que s'ha estudiat en aquest treball, especialment en situacions on hi ha una arribada contínua de tasques [4, 43].

Bibliografia

[1] Aakriti Agrawal et al. "DC-MRTA: Decentralized Multi-Robot Task Allocation and Navigation in Complex Environments". A: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pàg. 11711 - 11718. DOI: 10.1109/IROS47612.2022.9981353.

[2] Haris Aziz et al. "Task Allocation Using a Team of Robots". A: *Current Robotics Reports* 3 (ag. de 2022). DOI: 10.1007/s43154-022-00087-4.

[3] Mohamed Badreldin, Ahmed Hussein i Alaa Khamis. "A Comparative Study between Optimization and Market-Based Approaches to Multi-Robot Task Allocation". A: *Advances in Artificial Intelligence* 2013 (gen. de 2013), pàg. 11. DOI: 10.1155/2013/256524.

[4] Matteo Bellusci, Nicola Basilico i Francesco Amigoni. "Multi-Agent Path Finding in Configurable Environments". A: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS '20. Auckland, New Zealand: International Foundation for Autonomous Agents i Multiagent Systems, 2020, pàg. 159 - 167. ISBN: 9781450375184.

[5] Ali Bolu i Ömer Korçak. "Adaptive Task Planning for Multi-Robot Smart Warehouse". A: *IEEE Access* 9 (2021), pàg. 27346 - 27358. DOI: 10.1109/ACCESS.2021.3058190.

[6] Michael E. Bratman, David J. Israel i Martha E. Pollack. "Plans and Resource-Bounded Practical Reasoning". A: *Philosophy and AI: Essays at the Interface*. The MIT Press, des. de 1991. ISBN: 9780262315951. DOI: 10.7551/mitpress/5352.003.0003. URL: https://doi.org/10.7551/mitpress/5352.003.0003.

[7] Curtis Cohenour. "Teaching Finite State Machines (FSMs) as Part of a Programmable Logic Control (PLC) Course". A: 2017. URL: https://api.semanticscholar.org/CorpusID:67055954.

[8] I.F. Chaile i L. Ribas. "MASYM, a Framework to Deploy Synchronized Industrial Systems Based on Any ABM Simulator". A: *IEEE Latin America Transactions* 13.10 (2015), pàg. 3244 - 3252. DOI: 10.1109/TLA.2015.7387228.

[9] Ismael Chaile i Lluís Ribas-Xirgo. "Running Agent-based-models Simulations Synchronized with Reality to Control Transport Systems". A: *Automatika* 57 (oct. de 2016), pàg. 452 - 465. DOI: 10.7305/automatika.60-1.1243.

[10] Shushman Choudhury et al. "Efficient Large-Scale Multi-Drone Delivery Using Transit Networks". A: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pàg. 4543 - 4550. DOI: 10.1109/ICRA40945.2020.9197313.

[11] Pragna Das. "Adaptive multi-robot control through on-line parameter identification at system level". Tesi doct. Autonomous University of Barcelona, Spain, 2018. URL: http://hdl.handle.net/10803/650337.

[12] Edsger W. Dijkstra. "A note on two problems in connexion with graphs". A: *Numerische Mathematik* 1 (1959), pàg. 269 - 271. URL: https://api.semanticscholar.org/CorpusID:123284777.

[13] Eduardo Feo Flushing, Luca M. Gambardella i Gianni A. Di Caro. "Simultaneous task allocation, data routing, and transmission scheduling in mobile multi-robot teams". A: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pàg. 1861 - 1868. DOI: 10.1109/IROS.2017.8206002.

[14] Brian Gerkey i Maja Mataric. "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems". A: *I. J. Robotic Res.* 23 (set. de 2004), pàg. 939 - 954. DOI: 10.1177/0278364904045564.

[15] Matthew Gombolay, Ronald Wilcox i Julie Shah. "Fast Scheduling of Multi-Robot Teams with Temporospatial Constraints". A: juny de 2013. DOI: 10.15607/RSS.2013.IX.049.

[16] Peter E. Hart, Nils J. Nilsson i Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". A: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pàg. 100 - 107. DOI: 10.1109/TSSC.1968.300136.

[17] Bradford Heap i Maurice Pagnucco. "Repeated Sequential Single-Cluster Auctions with Dynamic Tasks for Multi-Robot Task Allocation with Pickup and Delivery". A: *Multiagent System Technologies*. Ed. de Matthias Klusch, Matthias Thimm i Marcin Paprzycki. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pàg. 87 - 100. ISBN: 978-3-642-40776-5.

[18] Rory Keeley. *CPLEX Optimization Studio 22.1.1 is available*. 2022. URL: https://community.ibm.com/community/user/ai-datascience/discussion/cplex-optimization-studio-2211-is-available (cons. 13-12-2022).

[19] Alaa Khamis, Ahmed Hussein i Ahmed Elmogy. "Multi-robot Task Allocation: A Review of the State-of-the-Art". A: vol. 604. Maig de 2015, pàg. 31 - 51. ISBN: 978-3-319-18299-5. DOI: 10.1007/978-3-319-18299-5_2.

[20] H. W. Kuhn. "The Hungarian method for the assignment problem". A: *Naval Research Logistics Quarterly* 2.1-2 (1955), pàg. 83 - 97. DOI: https://doi.org/10.1002/nav.3800020109. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800020109. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

[21] Man Li, Huaiyu Wu i Lin Chen. "Research on Multi-robot Cooperative Task Assignment Based on Cooperative Game". A: *2021 4th International Conference on Intelligent Autonomous Systems (ICoIAS)*. 2021, pàg. 424 - 429. DOI: 10.1109/ICoIAS53694.2021.00082.

[22] Aleksis Liekna, Egons Lavendelis i Arvids Grabovskis. "Experimental Analysis of Contract NET Protocol in Multi-Robot Task Allocation". A: *Applied Computer Systems* 13 (gen. de 2012). DOI: 10.2478/v10312-012-0001-7.

[23] Lantao Liu i Dylan Shell. "Optimal Market-based Multi-Robot Task Allocation via Strategic Pricing". A: juny de 2013. DOI: 10.15607/RSS.2013.IX.033.

[24] James R. Munkres. "ALGORITHMS FOR THE ASSIGNMENT AND TRANSIORTATION tROBLEMS*". A: 1957. URL: https://api.semanticscholar.org/CorpusID:15996572.

[25] Ernesto Nunes et al. "A taxonomy for task allocation problems with temporal and ordering constraints". A: *Robotics Auton. Syst.* 90 (2017), pàg. 55 - 70. URL: https://api.semanticscholar.org/CorpusID:11784030.

[26] George S. Oliveira et al. "Efficient Task Allocation in Smart Warehouses with Multi-Delivery Stations and Heterogeneous Robots". A: *2022 25th International Conference on Information Fusion (FUSION)*. 2022, pàg. 1 - 8. DOI: 10.23919/FUSION49751.2022.9841337.

[27] Daniel Rivas. "Execution-driven dynamic Multi-Robot Task Allocation model readily ap-plicable in real scenarios". PhD thesis. School of Engineering.Cerdanyola, Spain.: Universitat Autònoma de Barcelona, febr. de 2023.

[28] Daniel Rivas i Lluís Ribas-Xirgo. "Integrating State-Based Multi-Agent Task Allocation and Physical Simulators". A: *ROBOT2022: Fifth Iberian Robotics Conference*. Ed. de Danilo Tardioli et al. Cham: Springer International Publishing, 2023, pàg. 576 - 587. ISBN: 978-3-031-21062-4.

[29] Daniel Rivas et al. "Synthesis of Controllers from Finite State Stack Machine Diagrams". A: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. 2018, pàg. 1179 - 1182. DOI: 10.1109/ETFA.2018.8502451.

[30] Eric Rohmer, Surya P. N. Singh i Marc Freese. "V-REP: A versatile and scalable robot simulation framework". A: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pàg. 1321 - 1326. DOI: 10.1109/IROS.2013.6696520.

[31] J. Saiz-Alcaine et al. "Multi-Agent System Architecture of a Nano-Bio-Characterization Plant". A: *Actas del X Workshop en Agentes Físicos, WAF 2009*. Cáceres, Spain, set. de 2009. ISBN: 978-84-692-3220-0.

[32] Guni Sharon et al. "Conflict-Based Search for Optimal Multi-Agent Path Finding". A: *Proceedings of the International Symposium on Combinatorial Search* 3 (ag. de 2021), pàg. 190. DOI: 10.1609/socs.v3i1.18222.

[33] David Silver. "Cooperative Pathfinding." A: gen. de 2005, pàg. 117 - 122.

[34] George Marios Skaltsis, Hyo-Sang Shin i Antonios Tsourdos. "A survey of task allocation techniques in MAS". A: *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2021, pàg. 488 - 497. DOI: 10.1109/ICUAS51884.2021.9476736.

[35] Smith. "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver". A: *IEEE Transactions on Computers* C-29.12 (1980), pàg. 1104 - 1113. DOI: 10.1109/TC.1980.1675516.

[36] Roni Stern et al. "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks". A: *Proceedings of the International Symposium on Combinatorial Search* 10 (set. de 2021), pàg. 151 - 158. DOI: 10.1609/socs.v10i1.18510.

[37] Hongtao Tang et al. "Research on Equipment Configuration Optimization of AGV Unmanned Warehouse". A: *IEEE Access* 9 (2021), pàg. 47946 - 47959. DOI: 10.1109/ACCESS.2021.3066622.

[38] Jonatan Trullas-Ledesma i Lluis Ribas-Xirgo. "Cooperative Contract-Net Protocol-Based Multi-Robot Task Assignment". A: *Workshop de Agentes Físicos 2023 (WAF 2023)*. Universidad Rey Juan Carlos in Aranjuez, Madrid, Spain, nov. de 2023.

[39] Jonatan Trullas-Ledesma i Lluis Ribas-Xirgo. "Experiences in the 'agentification' of a manufacturing plant". A: *2008 2nd IEEE International Conference on Digital Ecosystems and Technologies*. 2008, pàg. 319 - 325. DOI: 10.1109/DEST.2008.4635173.

[40] Jonatan Trullas-Ledesma i Lluis Ribas-Xirgo. "Integrating physical and software sub-systems in a manufacturing environment through agentification". A: *Actas de la IX edición Workshop de Agentes Físicos (WAF 2008)*. Vigo, Spain, set. de 2009. ISBN: 978-84-8158-399-1.

[41] Jonatan Trullas-Ledesma i Lluis Ribas-Xirgo. "Simulación de las comunicaciones en el sistema de robots hormiga ANTSYS". A: *VI Workshop en Agentes Físicos, WAF2005*. Granada, Spain, set. de 2005, pàg. 75 - 82.

[42] Jonatan Trullas-Ledesma i Lluis Ribas-Xirgo. "Transforming SME manufacturing plants into evolvable systems through agents". A: *2009 3rd IEEE International Conference on Digital Ecosystems and Technologies*. 2009, pàg. 469 - 474. DOI: 10.1109/DEST.2009.5276766.

[43] Jonatan Trullas-Ledesma et al. "A study on applied cooperative path finding". A: *Actas de la XVII edición Workshop de Agentes Físicos (WAF 2016)*. Málaga, Spain, juny de 2016.

[44] Jonatan Trullàs-Ledesma, Daniel Rivas i Lluís Ribas-Xirgo. "Improving Auction-based Dynamic Multi-Robot Task Allocation with Hungarian Algorithm". Article submitted for publication in the journal Robotics and Autonomous Systems, Elsevier, April 2024. 2024.

[45] Gang Wang, Xiao Lv i Xiaohu Yan. "A Two-Stage Distributed Task Assignment Algorithm Based on Contract Net Protocol for Multi-UAV Cooperative Reconnaissance Task Reassignment in Dynamic Environments". A: *Sensors* 23.18 (2023). ISSN: 1424-8220. DOI: 10.3390/s23187980. URL: https://www.mdpi.com/1424-8220/23/18/7980.

[46] Yuzhan Wu et al. "Autonomous Last-Mile Delivery Based on the Cooperation of Multiple Heterogeneous Unmanned Ground Vehicles". A: *Mathematical Problems in Engineering* 2021 (març de 2021), pàg. 1 - 15. DOI: 10.1155/2021/5546581.

[47] Jiarui Zhang, Gang Wang i Yafei Song. "Task Assignment of the Improved Contract Net Protocol under a Multi-Agent System". A: *Algorithms* 12 (abr. de 2019), pàg. 70. DOI: 10.3390/a12040070.