Universidad Central "Marta Abreu" de Las Villas
Facultad de Matemática – Física – Computación
Dpto. Computación

Trabajo de Diploma

Aplicación de técnicas de Inteligencia Artificial en la solución del problema de ubicación en el diseño de bases de datos distribuidas.

Autor: Darién Rosa Paz

Tutores: Dra. Luisa M. González González
MSc. Abel Rodríguez Morffi

Santa Clara
2006

Hago constar que el presente trabajo fue realizado en la Universidad Central Marta Abreu de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencias de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del tutor

Firma del jefe del Seminario

RESUMEN

El auge alcanzado por los sistemas de información distribuidos y el desarrollo de los manejadores de bases de datos comerciales, ha hecho que los sistemas de bases de datos distribuidas se conviertan en una alternativa viable para satisfacer las necesidades de información actuales.

La distribución de los datos es un problema crítico que afecta de manera considerable el desempeño global de los sistemas distribuidos, debido a que influye directamente en la eficiencia del procesamiento de las consultas. Debido a la complejidad del problema, las diferentes propuestas de solución presentadas hasta la fecha, han coincidido en dividir el proceso de diseño de la distribución en dos fases seriadas: la fragmentación y la ubicación de los fragmentos en los sitios de la red.

La presente investigación, titulada “Aplicación de técnicas de Inteligencia Artificial en la solución del problema de ubicación en el diseño de bases de datos distribuidas”, aborda el problema de la asignación de fragmentos en el diseño de bases de datos distribuidas, tomando como referencia el modelo matemático Özsu, al cual se le da solución aplicando las técnicas de Algoritmos Genéticos y el algoritmo Q-Learning del Aprendizaje Reforzado.

El producto final de la investigación es un asistente que será insertado en una herramienta general para el diseño de bases de datos distribuidas.

SUMMARY

The boom reached by the distributed information systems and the development of the commercial data base managers, has made possible that the distributed data base systems become into a feasible alternative for satisfying the nowadays information needs.

The data distribution is a critical problem which affects in a considerable way the overall performance of the distributed systems; given that it has a direct influence on the efficiency of the process of consulting. Due to the complexity of the problem, the several proposed solutions presented until today, have in common that they divide the process of designing the distribution into two successive phases: the fragmentation and the allocation of the fragments in the network sites.

The present research, entitled “Application of Artificial Intelligence techniques to the solution of the allocation problem in the distributed data bases design”, deals with the problem of the assignation of fragments in the design of distributed data bases, taking as a reference the Özsu mathematical model, which is solved applying the techniques of Genetic Algorithms and the Q-Learning algorithm of the Reinforcement Learning. The final product of the research is an assistant which will be inserted into a general tool for the design of distributed data bases.

INDICE

INTRODUCCION

En los últimos tiempos se ha producido un auge notable de los sistemas de información distribuidos, debido, entre otros factores, a la gran cantidad de información que cada día se va generando, a los avances en la tecnología de las comunicaciones y al decremento en los costos de las computadoras. Los sistemas distribuidos se adaptan mejor a las necesidades de las organizaciones descentralizadas ya que reflejan más adecuadamente su estructura y tienen importantes ventajas con respecto a los centralizados (Özsu y Valduriez, 1991), aunque al mismo tiempo añaden nuevas complejidades en su diseño e implementación.

De este modo los Sistemas de Gestión de Base de Datos (SGBD) han avanzado de centralizados a distribuidos, donde tanto los datos como las aplicaciones se encuentran físicamente distribuidos en los diferentes nodos de una red de computadoras, aunque lógicamente pertenecen a una base de datos única; de manera que la distribución sea transparente al usuario, y parezca que se está accediendo a una Base de Datos Centralizada (BDC).

Una Base de Datos Distribuida (BDD) está conformada por un conjunto de localidades (sitios) interconectadas por una red de computadoras; en cada sitio hay un Sistema de Gestión de Bases de Datos (SGBD) que funciona autónomamente, pero que requiere interactuar con los demás, pues los datos que solicitan las aplicaciones posiblemente se encuentran ubicados en sitios diferentes.

El diseñador de una base de datos distribuida tiene la tarea de definir la distribución de la información entre los sitios de la red. Este problema se le conoce como diseño de la distribución y nace de la necesidad de especificar las unidades de almacenamiento adecuadas junto con su ubicación; trazándose como objetivo lograr una máxima localidad, o sea, que los datos se encuentren en el sitio donde más se necesiten y aprovechar la distribución para paralelizar ciertas operaciones, repartiendo la carga de trabajo. De esta manera el diseño de la distribución lleva implícito dos problemas fundamentales: la fragmentación de la base de datos y la ubicación de los fragmentos en los sitios de la red.

La ubicación de los datos afecta notablemente el desempeño de los sistemas distribuidos, debido a que el tiempo y el costo requeridos para el procesamiento de las solicitudes dependen en gran parte del lugar donde se encuentren

almacenados, ya sea en un solo nodo o que estén distribuidos en varios sitios de la red.

El problema de la ubicación ha sido estudiado ampliamente por diferentes investigadores, aunque aun no se ha resuelto totalmente, debido a su dificultad tanto en la formulación del modelo matemático como en la fase de solución del mismo, ya que todos hacen uso de consideraciones y modelan diferentes parámetros, por lo que son aplicables solo bajo ciertas especificaciones (Mei y Sheng, 1992). En muchos de ellos se impone la condición de que cada fragmento se encuentre almacenado en un solo sitio de la red, de esta forma se desaprovechan las ventajas de la replicación a cambio de que el espacio de soluciones disminuya considerablemente.

Para dar solución a estos complejos modelos de optimización se emplean enfoques exactos y heurísticos, los cuales han resuelto parte del espectro de problemas reales. Los métodos exactos resultan factibles para solucionar problemas pequeños, no así para problemas mayores que son tratados mediante técnicas heurísticas.

En el pasado reciente se han desarrollado trabajos de investigación que tratan el problema de la asignación de fragmentos en bases de datos distribuidas. Ejemplo de ello es la aplicación del método de Algoritmos Genéticos en la solución del modelo matemático Özsu en (Águila, 2001) y el empleo de los algoritmos SARSA y Q-Learning de Aprendizaje Reforzado para solucionar los modelos FURD en (González et al., 2001) y Özsu en (Hernández, 2005) respectivamente. En los dos últimos casos no se tuvo en cuenta la replicación de fragmentos.

Dando continuidad a las investigaciones mencionadas y ante la necesidad de incorporar las facilidades de la replicación de fragmentos, se impone la necesidad de desarrollar nuevas herramientas para solucionar el problema de la asignación de fragmentos; empleando métodos heurísticos para resolver un modelo que simule lo más posible las situaciones de la vida real, en un tiempo aceptable y con un costo computacional considerablemente bueno.

Por lo anteriormente expuesto se define como problema científico la no existencia de una herramienta que resuelva el problema de asignación de fragmentos en el diseño de bases de datos distribuidas, aplicando un algoritmo de Aprendizaje Reforzado y que tenga en cuenta la replicación.

En aras de solucionar la situación planteada, se establecen las hipótesis siguientes:

- Si es factible la aplicación de un algoritmo de Aprendizaje Reforzado para dar solución al problema de asignación en el diseño de bases de datos distribuidas, teniendo en cuenta la replicación de fragmentos.
- Si el algoritmo de Aprendizaje Reforzado empleado iguala o supera las prestaciones de los Algoritmos Genéticos en la solución de dicho problema.

Estas hipótesis quedarán validadas si:

- Se implementa el algoritmo Q-Learning para dar solución al modelo matemático Özsu y se comprueban los resultados obtenidos con la ayuda de métodos exactos, para instancias pequeñas del problema y otros métodos heurísticos para instancias mayores.
- Se obtienen soluciones semejantes a las obtenidas por el método de Algoritmos Genéticos para instancias pequeñas o medianas del problema, considerando la optimalidad de las soluciones y el tiempo que demora en encontrarlas.

Para dar cumplimiento a las hipótesis planteadas anteriormente, se define el sistema de objetivos siguiente:

Objetivo general

Resolver el problema de la asignación de fragmentos en el diseño de bases de datos distribuidas aplicando las técnicas de Algoritmos Genéticos y Aprendizaje Reforzado.

Objetivos específicos

1. Construir un marco teórico-referencial, derivado de la revisión de la literatura nacional e internacional, que exponga los principales aspectos relacionados con el diseño de bases de datos distribuidas; los Algoritmos Genéticos y el Aprendizaje Reforzado.
2. Implementar un módulo que resuelva el problema de asignación planteado en el modelo matemático Özsu, usando el algoritmo Q-Learning.
3. Actualizar el módulo de asignación implementado en (Águila, 2001) que usa el método de Algoritmos Genéticos.

4. Implementar un asistente para guiar el proceso de asignación de fragmentos. Este asistente será insertado en una herramienta general de diseño de bases de datos distribuidas.

Esta investigación se encuentra estructurada en tres capítulos: en el primero se hace referencia a las cuestiones teóricas que influyen en el diseño de bases de datos distribuidas, haciendo énfasis en el modelo matemático Özsu que se utiliza para representar el problema de la ubicación en el diseño de la distribución; el segundo muestra una descripción detallada de los principales aspectos relacionados con las técnicas de Algoritmos Genéticos y Aprendizaje Reforzado, lo cual fundamenta su posterior implementación en la solución del problema de optimización planteado en el capítulo I; en el tercero se explican los aspectos generales del diseño y la implementación del asistente, las consideraciones hechas al modelo matemático, y las pruebas realizadas con los resultados obtenidos.

CAPITULO I. DISEÑO DE BASES DE DATOS DISTRIBUIDAS MODELO DE ASIGNACION

En este capítulo se hace referencia a las principales cuestiones teóricas del diseño de bases de datos distribuidas, haciendo énfasis en el modelo matemático Özsu, que se utiliza para dar solución al problema de la ubicación en el diseño de la distribución.

1.1 Diseño de bases de datos distribuidas

El diseño de Bases de Datos Distribuidas (BDD) se deriva del diseño convencional de bases de datos centralizadas, añadiéndosele los problemas presentes en el diseño de la distribución. Como se puede apreciar en la figura 1.1, el diseño de la distribución incluye la adquisición de datos, la fragmentación de la base de datos, la ubicación y replicación de los fragmentos y la optimización local.

En el caso de SGBD Distribuidos (SGBDD), la distribución de las aplicaciones incluye tres elementos (Chakravarthy et al., 1993):

1. La distribución del software del SGBD distribuido.
2. La distribución de los programas de aplicación que se ejecutan sobre este.
3. La distribución de los datos.

El primer aspecto no constituye un problema en esta investigación, pues se admite que existe una copia del SGBDD en cada sitio en los que se encuentran almacenados los datos. El segundo aspecto representa de manera implícita que el sistema distribuido responde a la estructura de la empresa, de forma que los programas de aplicación se ubican en los lugares donde son necesarios. De este modo, se asume que la red ya está diseñada o será diseñada en un paso posterior, por lo que se pondrá énfasis en la distribución de los datos.

1.1.1 Estrategias de diseño

A la hora de abordar el diseño de una base de datos distribuida se puede optar principalmente por dos tipos de estrategias: la estrategia ascendente (Bottom-Up) y la estrategia descendente (Top-Down). La modelación de un problema real se presenta usualmente de una manera no pura, pues no resultaría extraño a la hora de abordar un trabajo de diseño de una base de datos que se pudiesen emplear en diferentes etapas del proyecto cualquiera de las alternativas, que en la práctica no siempre se pueden aislar, pero que para su estudio conviene diferenciarlas.

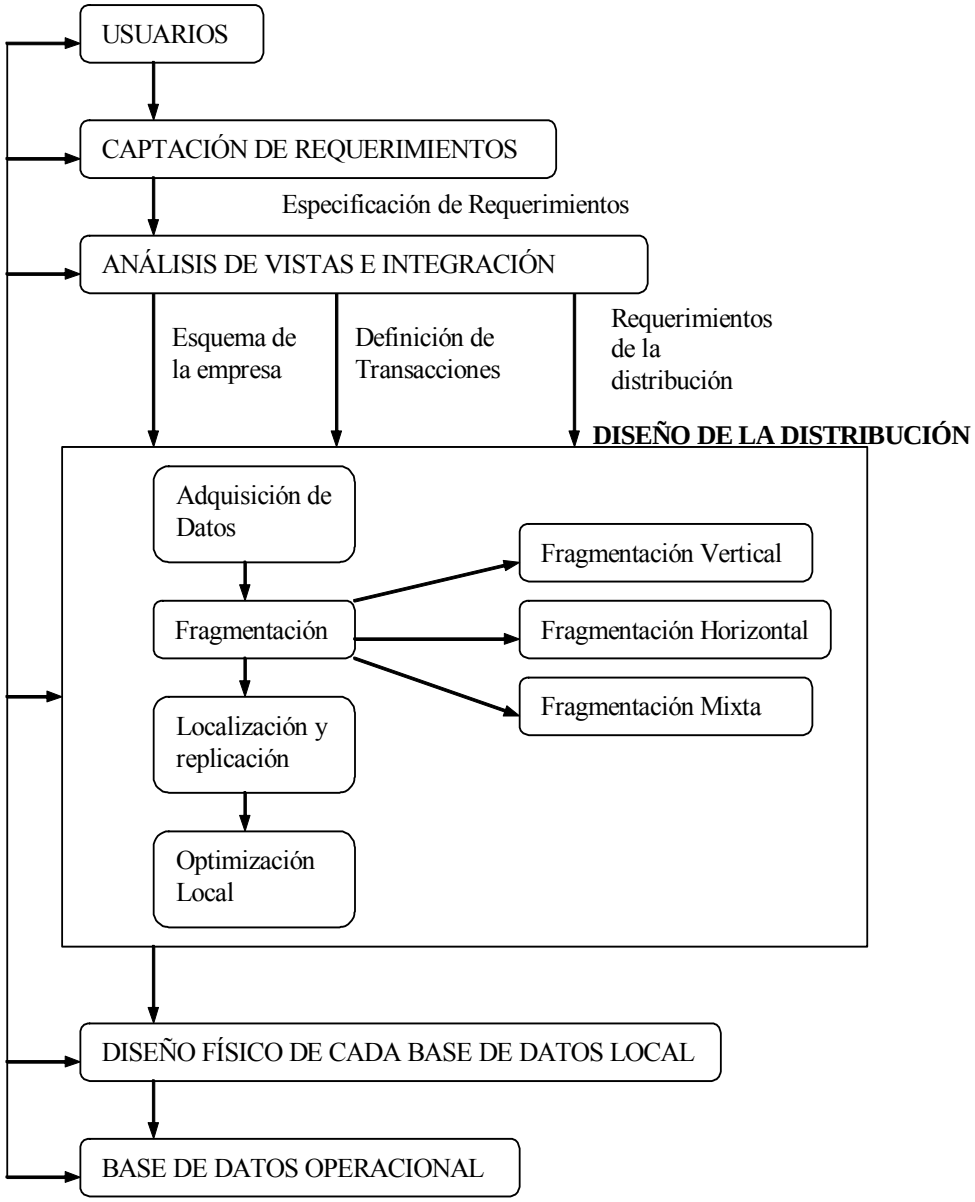


Figura 1.1 Metodología de diseño de bases de datos distribuidas. (Chakravarthy et al, 1992)

1.1.1.1 Estrategia de diseño Bottom-Up

La estrategia ascendente podría aplicarse cuando ya existen sistemas instalados y haya que proceder a un diseño a partir de un número de pequeñas bases de datos existentes, con el fin de integrarlas en una sola. Se partiría de los esquemas conceptuales locales y se trabajaría para conseguir el esquema conceptual global. Aunque este caso se puede presentar con facilidad en la vida real, se prefiere

pensar en un problema donde se parte de cero y se avanza en el desarrollo del trabajo siguiendo la estrategia descendente.

1.1.1.2 Estrategia de diseño Top-Down

La versión top-down es adecuada cuando un sistema de bases de datos se construye por primera vez, sin tener restricciones de otros sistemas ya instalados y que deban ser integrados de alguna manera al sistema distribuido. Un esquema de este proceso puede observarse en la figura 1.2.

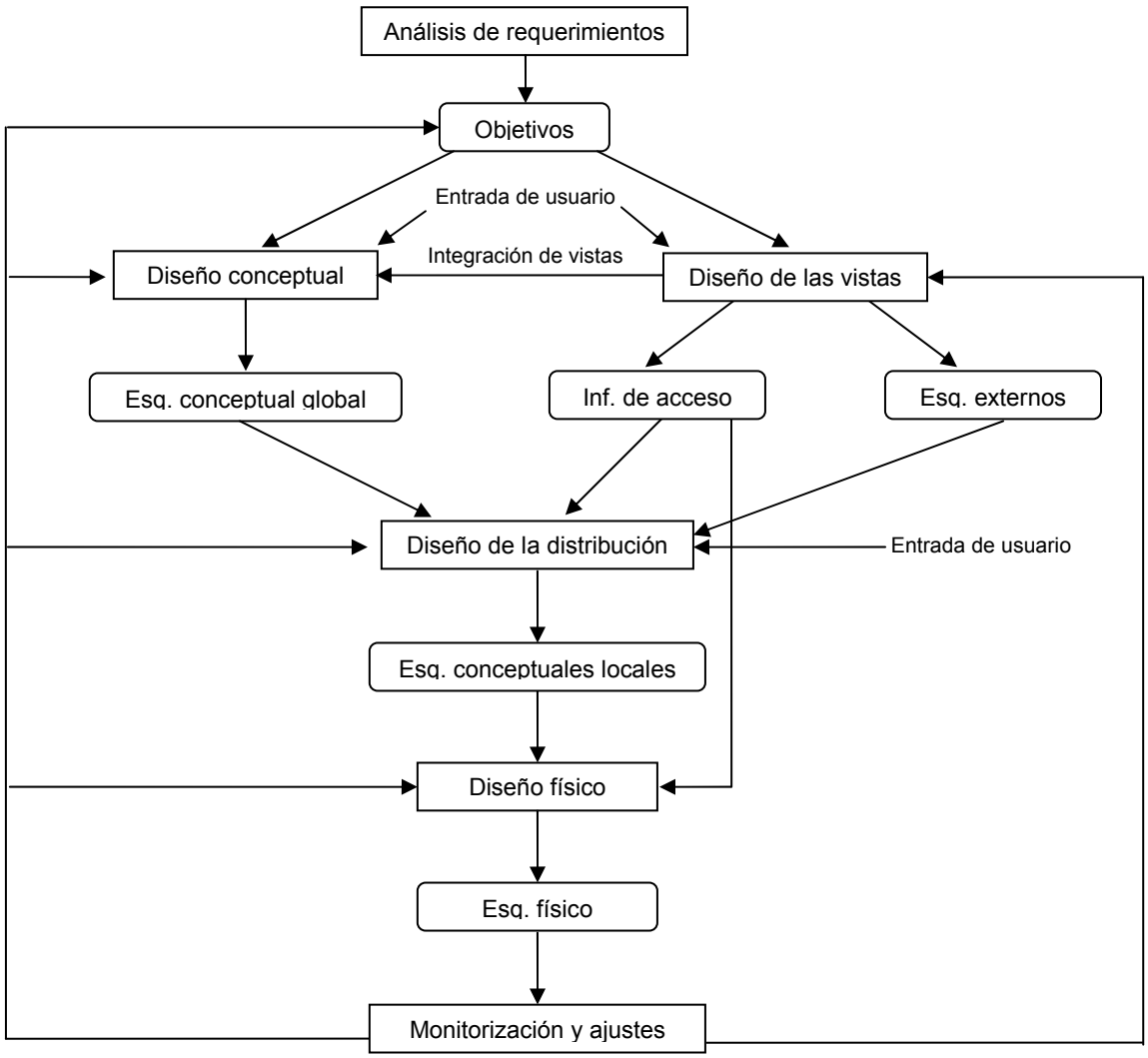


Figura 1.2 Esquema del proceso de diseño Top-Down. (Özsu y Valduriez, 1999)

El análisis de los requerimientos define al ambiente del sistema y extrae tanto los datos como los procesos que necesitan todos los usuarios potenciales de la BD.

El diseño de vistas y el diseño conceptual son dos procesos que se realizan en paralelo. El diseño de vistas tiene que ver con la definición de interfaces para los usuarios finales. El diseño conceptual es el proceso por el cual es analizado el

problema para determinar los tipos de entidades y las relaciones entre dichas entidades. Se puede dividir este proceso en dos grupos de actividades relacionados entre sí: análisis de entidades y análisis funcional. El análisis de entidades está relacionado con la determinación de las entidades, sus atributos, y las relaciones entre ellos. El análisis funcional, por otra parte, se corresponde con la determinación de las funciones fundamentales con las cuales se relaciona el problema modelado.

Existe una relación entre el diseño conceptual y el diseño de vistas. En un sentido, el diseño conceptual puede ser interpretado como una integración de las vistas de usuarios. Este aspecto es de vital importancia ya que el modelo conceptual debería soportar no solo las aplicaciones existentes, sino que debería estar preparado para futuras aplicaciones.

En las actividades de diseño conceptual y diseño de vistas en BDD, el usuario necesita especificar las entidades y debe determinar qué aplicaciones van a ser soportadas sobre la base de datos, también necesita colectar información estadística sobre dichas aplicaciones. La información estadística incluye la especificación de la frecuencia de las aplicaciones del usuario, el volumen de información, entre otras, que se describen de forma detallada en (Artiles, 2001).

El Esquema Conceptual Global (ECG) y la información sobre los patrones de acceso obtenidos como resultado del diseño de vistas son entradas para el paso del diseño de la distribución. El objetivo en esta etapa es diseñar los Esquemas Conceptuales Locales (ECL) distribuyendo las entidades en los sitios del sistema distribuido.

El diseño físico trasforma los esquemas conceptuales locales en estructuras físicas, ubicadas en los dispositivos de almacenamiento disponibles en los sitios correspondientes.

Debido a que el proceso de diseño está en continuo movimiento, requiere un constante monitoreo, ajustes periódicos y puesta a punto; y es posible regresar a reconsiderar estados anteriores ya sea de instrumentación de la BD como de las vistas de los usuarios (Águila, 2001).

1.1.2 Diseño de la distribución de datos

Existen diversas formas de afrontar el problema del diseño de la distribución. Las más usuales se muestran en la figura 1.3.

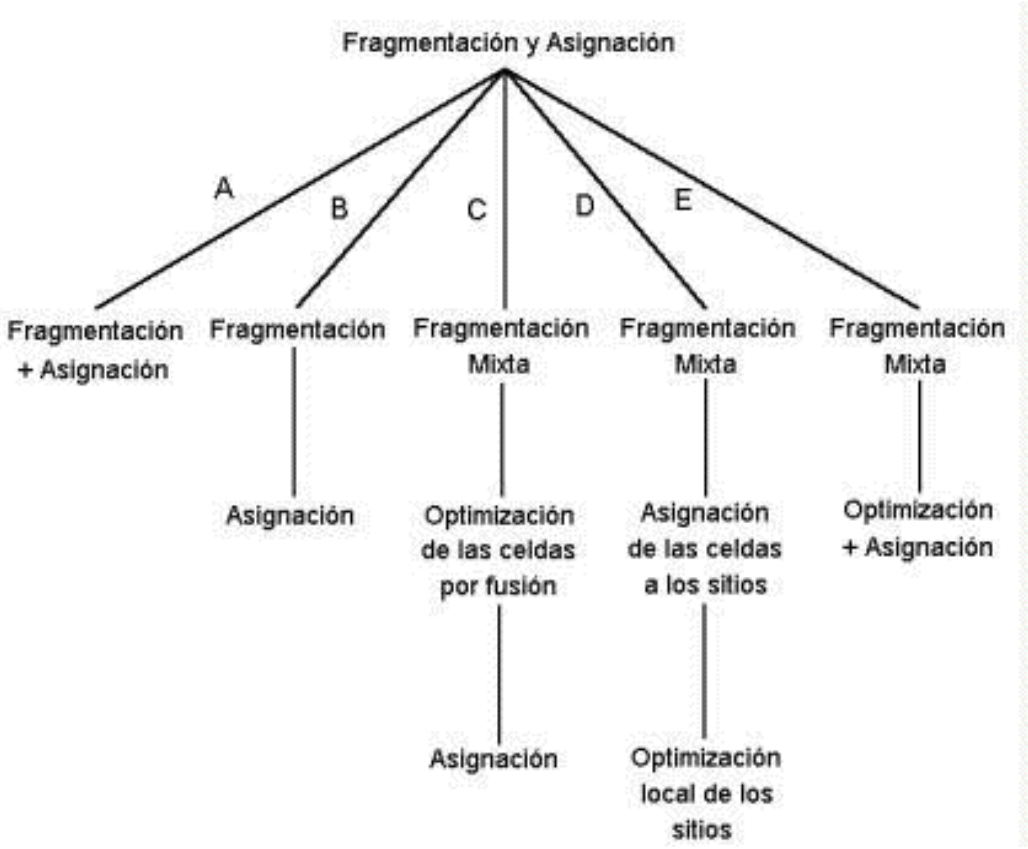


Figura 1.3 Alternativas del diseño de la distribución. (Morell, 1999)

En el primer caso: caso A, los dos procesos fundamentales: la fragmentación y la asignación, se abordan de forma simultánea. Esta metodología se encuentra en desuso, sustituida por el enfoque en dos fases: caso B, donde se realiza primeramente la partición para luego asignar los fragmentos generados. Esto se debe a dos razones: primero, para manejar mejor la complejidad del diseño y segundo, porque es relevante desde el punto de vista conceptual ya que la fragmentación trata los criterios lógicos que motivan la división de una relación global mientras que la distribución tiene que ver con la ubicación física de los datos en los diferentes sitios de la red. Sin embargo, esta separación debe hacerse con mucho cuidado porque en general no es posible determinar la fragmentación y localización óptima resolviendo estos dos problemas independientemente ya que ellos están muy interrelacionados (Morell, 1999). El resto de los casos se comentan en los epígrafes siguientes.

1.1.2.1 Fragmentación

El diseño de la fragmentación es el primer problema que debe enfrentar la distribución de datos y su propósito es obtener porciones no solapadas de la BD que sean unidades lógicas de asignación.

El dilema principal de la fragmentación radica en encontrar la unidad apropiada de distribución. Una relación no es una buena unidad por muchas razones.

Primero, las vistas de la aplicación normalmente son subconjuntos de relaciones. Además, la localidad de los accesos de las aplicaciones no está definida sobre relaciones enteras, pero sí sobre subconjuntos de las mismas. Por ello, sería normal considerar como unidad de distribución a estos subconjuntos de relaciones.

Segundo, si las aplicaciones tienen vistas definidas sobre una determinada relación, considerándola una unidad de distribución que reside en varios sitios de la red, se puede optar por dos alternativas. Por un lado, la relación no estará replicada y se almacena en un único sitio, o existe réplica en todos o algunos de los sitios en los cuales reside la aplicación. Las consecuencias de esta estrategia son la generación de un volumen de accesos remotos innecesario. Además, se pueden realizar réplicas innecesarias que causen problemas en la ejecución de las actualizaciones y puede no ser deseable si el espacio de almacenamiento está limitado.

Tercero, la descomposición de una relación en fragmentos, tratados cada uno de ellos como una unidad de distribución, permite el proceso concurrente de las transacciones. También la relación de estas relaciones, normalmente, provocará la ejecución paralela de una consulta al dividirla en una serie de subconsultas que operará sobre los fragmentos.

Un inconveniente que trae consigo la fragmentación es el problema del control semántico. Como resultado de la fragmentación, los atributos implicados en una dependencia se descomponen en diferentes fragmentos, los cuales pueden destinarse a sitios diferentes. En este caso, la sencilla tarea de verificar las dependencias puede resultar una tarea de búsqueda de los datos implicados en un gran número de sitios (Özsu y Valduriez, 1991).

Tipos de fragmentación

Dado que una relación se corresponde esencialmente con una tabla y la cuestión consiste en dividirla en fragmentos menores, inmediatamente surgen dos

alternativas lógicas para llevar a cabo el proceso: la división horizontal y la división vertical. La división o fragmentación horizontal trabaja sobre las tuplas, dividiendo la relación en subrelaciones que contienen un subconjunto de las tuplas que alberga la primera. La fragmentación vertical, en cambio, se basa en los atributos de la relación para efectuar la división. Estos dos tipos de partición podrían considerarse los principales y básicos. Sin embargo, existen otras alternativas. Fundamentalmente, se habla de fragmentación mixta o híbrida cuando el proceso de partición hace uso de los dos tipos anteriores. La fragmentación mixta puede llevarse a cabo de tres formas diferentes: desarrollando primero la fragmentación vertical y posteriormente aplicando la partición horizontal sobre los fragmentos verticales (denominada partición VH), o aplicando primero una división horizontal para luego, sobre los fragmentos generados, desarrollar una fragmentación vertical (llamada partición HV), o bien, de forma directa considerando la semántica de las transacciones. Otro enfoque distinto y relativamente nuevo consiste en aplicar sobre una relación, de forma simultánea y no secuencial, la fragmentación horizontal y la fragmentación vertical; en este caso se generará una rejilla y los fragmentos formarán las celdas de esa rejilla, cada celda será exactamente un fragmento vertical y un fragmento horizontal, de este modo el grado de fragmentación alcanzado es máximo, y no por ello la descomposición resultará más eficiente.

Volviendo a la figura 1.3, puede observarse como los casos C y D se basan en la mencionada generación de la rejilla, con la diferencia que en el primero de ellos se produce una fusión, una desfragmentación de las celdas, agrupándolas de la manera más adecuada para obtener un mayor rendimiento, ya que los fragmentos generados son muy pequeños. En el segundo caso se asignan las celdas a los sitios y luego se realiza una rigurosa optimización en cada sitio. El caso E sería aquel en el que se utiliza la fragmentación VH o la HV (Morell, 1999).

1.1.2.2 Ubicación

Asumiendo que la base de datos es adecuadamente fragmentada se procede a ubicar los fragmentos en los diferentes sitios de la red. Cuando los datos son ubicados pueden estar replicados o mantenidos como una sola copia. Las razones para la replicación son la confiabilidad y la eficiencia de las solicitudes de solo lectura. Si existen múltiples copias de un elemento de datos habrá una mayor

probabilidad de que alguna copia de los datos esté accesible en algún lugar, incluso si ocurriera una falla en el sistema. Mejor aún, las solicitudes de solo lectura que acceden a los mismos artículos de datos pueden ser ejecutadas en paralelo ya que las copias existen en múltiples sitios. Por otra parte, la ejecución de solicitudes de actualización causa problemas, ya que el sistema tiene que asegurar que todas las copias de los datos sean propiamente actualizadas. De aquí, que la decisión con relación a la replicación es una cuestión que depende de la proporción de la solicitudes de solo lectura con respecto a las solicitudes de actualización. Esta decisión afecta a casi todos los algoritmos del SGBDD y a las funciones de control.

Una BD no replicada (comúnmente llamada una BD particionada) contiene fragmentos que son ubicados en los sitios, y existe una copia única de cada fragmento en la red. En caso de replicación, o bien la BD existe completa en cada sitio (BD replicada completamente), o los fragmentos son distribuidos en los sitios de tal manera que la copias de un fragmento puedan estar en múltiples sitios (BD replicada parcialmente). En esta última el número de copias de un fragmento puede ser una entrada para el algoritmo de ubicación o una variable de decisión cuyo valor lo determina el propio algoritmo.

En la tabla 1.1 se puede ver una comparación entre estas tres alternativas de replicación con respecto a varias funciones del SGBDD.

	Completa	Parcial	Particionada
Procesamiento de Solicitudes	Fácil	Igual Dificultad	
Administración de Directorios	Fácil o No Existente	Igual Dificultad	
Control de Concurrencia	Moderada	Difícil	Fácil
Confiabilidad	Muy Alta	Alta	Baja
Aplicación en la vida real	Posible aplicación	Realista	Posible aplicación

Tabla 1.1 Comparación de las alternativas de replicación. (Özsu y Valduriez, 1998)

Requerimientos de información

Muchos factores contribuyen a que se realice un diseño de distribución óptimo. La organización lógica de la BD, la ubicación de las aplicaciones, las características de acceso de las aplicaciones a la BD, y las propiedades de los sistemas de computación en cada sitio; todas tienen una influencia sobre las decisiones de

distribución. Esto hace que sea muy compleja la formulación de un problema de distribución.

La información que se necesita para el diseño de distribución está relacionada con: información sobre la base de datos, información sobre las aplicaciones e información sobre la red de comunicación. La última categoría es completamente cuantitativa en cuanto a su naturaleza y es más usada en los modelos de ubicación que en los algoritmos de fragmentación (Özsu y Valduriez, 1999).

1.1.2.3 Objetivos del diseño de la distribución de los datos

Durante el diseño deben observarse los siguientes objetivos o aspectos que se desean priorizar:

- Localidad del procesamiento: Distribuir los datos atendiendo a la maximización de la localidad del procesamiento responde a un principio simple de colocar los datos tan cerca como sea posible de las aplicaciones que los usan. La forma más simple de caracterizar la localidad es considerar dos tipos de referencias: locales y remotas. Diseñar una distribución que maximice localidad del procesamiento puede hacerse añadiendo la cantidad de referencias locales y remotas correspondientes a cada fragmentación candidata y asignar los fragmentos eligiendo la mejor solución.
- Disponibilidad y confiabilidad de datos distribuidos: Ambas son ventajas de los sistemas distribuidos, un alto grado de disponibilidad se logra para aplicaciones de solo-lectura si se almacenan múltiples copias de la misma información, de forma que el sistema tenga alternativas de solución si alguna de ellas no está disponible. La confiabilidad también se logra mediante la existencia de múltiples copias, pues es posible recuperar copias dañadas o destruidas a partir de otra. Como los daños pueden obedecer a catástrofes físicas, deben tenerse copias en lugares geográficamente separados.
- Distribución de la carga de trabajo: La distribución de la carga de trabajo sobre los sitios se hace sobre la base de utilizar la potencia de los computadores de cada sitio y maximizar paralelismo en la ejecución de las aplicaciones. Como que la distribución de la carga puede afectar la localidad del procesamiento es necesario correlacionar ambos objetivos.
- Disponibilidad y costo de almacenamiento: La capacidad de almacenamiento de cada sitio debe tenerse en cuenta, usualmente el costo de almacenamiento no

es importante comparado con otros costos, no obstante, las limitaciones de almacenamiento deben ser consideradas.

Atender simultáneamente a todos los criterios es difícil, pues conduce a modelos de optimización muy complejos; con frecuencia se consideran algunos objetivos como restricciones, o se priorizan algunos objetivos en unos estados de diseño y se introducen otros en estados posteriores (Rodríguez, 1998).

1.2 Modelo de asignación

La asignación de recursos a través de los nodos de una red de computadoras es un problema que ha sido estudiado extensivamente. No obstante, la mayoría de estos trabajos no solucionan el problema del diseño de bases de datos distribuidas, sino el de ubicar ficheros individuales en una red de computadoras.

A continuación se define el problema de ubicación.

Se asume que hay un conjunto de fragmentos $F = \{f_1, f_2, ..., f_n\}$ y una red constituida por los sitios $S = \{s_1, s_2, ..., s_m\}$ en los cuales se están ejecutando un conjunto de aplicaciones $Q = \{q_1, q_2, ..., q_q\}$. El problema de ubicación consiste en encontrar la distribución "óptima" de *F* sobre *S*.

Un detalle muy importante que necesita ser discutido es la definición de optimalidad. La optimalidad puede ser definida respecto a dos medidas (Özsu y Valduriez, 1999):

- Costo mínimo: La función de costo consiste en el costo de almacenar cada f_i en un sitio s_j, el costo de solicitar f_i en el sitio s_j, el costo de actualizar f_i en todos los sitios donde esté almacenado, y el costo de comunicación.
- Desempeño: La estrategia de ubicación está diseñada para mantener una métrica del desempeño. Dos de las más conocidas son: minimizar el tiempo de respuesta y maximizar el rendimiento del sistema en cada sitio.

A continuación se plantea una forma general del modelo de ubicación de fragmentos cuyo objetivo es minimizar el costo total de procesamiento y almacenamiento, sujeto a algunas restricciones (Özsu y Valduriez, 1999).

min. (Costo Total)

Sujeto a:

Restricciones de tiempo de respuesta.

Restricciones de almacenamiento.

Restricciones de procesamiento.

La variable de decisión es x_{ij} y se define como:

$$x_{ij} = \begin{cases} 1, F_i almacenado\_enS_j \\ 0, e.o.c. \end{cases}$$

No existen modelos heurísticos generales que tengan como punto de partida el conjunto de fragmentos y den como solución una ubicación cercana al óptimo sujeto a los tipos de restricciones mencionadas anteriormente. Los modelos desarrollados hasta el momento asumen un gran número de simplificaciones y son aplicables a ciertas formulaciones muy específicas; por lo que se presenta un modelo relativamente general y se analizan algunas heurísticas que se pueden emplear para resolverlo.

1.2.1 Información requerida

Para el proceso de asignación se necesitan datos cuantitativos sobre la base de datos, las aplicaciones que se ejecutan sobre esta, la red de comunicación y sobre cada sitio de la red. A continuación se detalla cada aspecto.

Información sobre la base de datos

- $sel_i(f_j)$: Se define como selectividad de un fragmento f_j con respecto a una solicitud q_i, como el número de tuplas de f_j que tienen que ser accedidas para procesar q_i.
- $size(f_j) = card(f_j) * length(f_j)$. El tamaño de un fragmento f_j dado por la cantidad de tuplas del fragmento y su longitud en bytes.

Información sobre las aplicaciones

La mayoría de la información relacionada con las aplicaciones ya ha sido recopilada durante el proceso de fragmentación, pero para el modelo de ubicación se requiere alguna información adicional:

- RR_{ij}: Número de accesos de lectura que una consulta q_i realiza sobre un fragmento f_j durante su ejecución.
- UR_{ij}: De forma similar representa el número de accesos de actualización.

Se necesitan también dos matrices: UM y RM con elementos u_{ij} y r_{ij} respectivamente, especificados como sigue:

$$u_{ij} = \begin{cases} 1, q_i actualiza F_j \\ 0, e.o.c. \end{cases} \qquad r_{ij} = \begin{cases} 1, q_i lee F_j \\ 0, e.o.c. \end{cases}$$

- $o(i)$: Sitio donde se origina la consulta q_i.
- MaxTime (i): Tiempo de respuesta máximo admisible para cada aplicación.

Información sobre los sitios

Para cada sitio se necesita conocer su capacidad de almacenamiento y su velocidad de procesamiento. Para captar estos valores se pueden elaborar funciones o simplemente pueden ser estimados.

- USC_k: Costo unitario de almacenar un bloque de datos en el sitio s_k.
- SPC_k: Costo de procesar una unidad de trabajo en el sitio s_k. La unidad de trabajo pudiera ser igual a los valores de RR y UR.

Información sobre la red

En este modelo se asume la existencia de una red simple, donde el costo de comunicación pueda ser definido en términos de un frame de datos.

- g_{ij} : Costo de comunicación por frame entre los sitios s_i y s_j.
- *fsize*: Tamaño (en bytes) de un frame.

Se pudiera realizar un modelo más elaborado sobre la red que incluyera otros parámetros como la capacidad de los canales, la distancia entre los sitios, etc.; pero con esto se obtendría un modelo mucho más complejo.

En un ambiente distribuido los fragmentos de datos son ubicados en sitios diferentes, de manera que en el procesamiento de las consultas se combinan datos de diferentes sitios para obtener la respuesta final. Para hacer esto de una forma económica se requiere de una buena función de costo que justifique la fragmentación de los datos, contemple las frecuencias de uso de todas las consultas y actualizaciones, y los sitios a los que los resultados deben ser enviados. En la mayoría de los sistemas de la vida real, toda la información requerida para la solución del modelo es prácticamente imposible de recopilar, ya que el sistema no tiene un comportamiento estático en el tiempo y la cantidad de datos que se requieren para el modelo podría ser abrumadora.

1.2.2 Función de costo

La función de costo total tiene dos componentes: el procesamiento de solicitudes y el almacenamiento. Se expresa de la siguiente forma:

$$\mathrm{TOC} = \sum_{\forall q_i \in Q} \mathrm{QPC}_i + \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} \mathrm{STC}_{jk}$$

donde QPC_i es el costo de procesamiento de las solicitudes de la aplicación q_i, y STC_{jk} es el costo de almacenamiento del fragmento f_j en el sitio s_k.

El costo de almacenamiento está dado por:

$$\mathrm{STC}_{jk} = \mathrm{USC}_k \cdot \mathrm{size}(F_j) \cdot x_{jk}$$

Las dos sumatorias calculan el costo de almacenamiento total en todos los sitios, para todos los fragmentos.

El costo de procesamiento de las solicitudes es un poco más difícil de especificar. La mayoría de los modelos para resolver el clásico problema de ubicación de ficheros en sistemas distribuidos lo separan en dos componentes: el costo de procesamiento para solo lectura, y el costo de procesamiento para la actualización. En este caso, que se trata de un modelo para el problema de ubicación de base de datos, se especificará como el costo de procesamiento (PC) y el costo de transmisión (TC). Así, el costo de procesamiento de solicitudes (QPC) para la aplicación q_i sería:

$$\mathrm{QPC}_i = \mathrm{PC}_i + \mathrm{TC}_i$$

El costo de procesamiento PC, contiene tres factores, el costo de acceso (AC), el costo del chequeo de restricciones de integridad (IE), y el costo del control de concurrencia (CC):

$$\mathrm{PC}_i = \mathrm{AC}_i + \mathrm{IE}_i + \mathrm{CC}_i$$

La especificación de cada uno de esos factores de costo depende de los algoritmos usados para realizar estas tareas.

La especificación detallada de AC sería:

$$\mathrm{AC}_i = \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} \left(u_{ij} \times \mathrm{UR}_{ij} + r_{ij} \times \mathrm{RR}_{ij}\right) \times x_{jk} \times \mathrm{SPC}_k$$

Los dos primeros términos de la fórmula anterior calculan el número de accesos de solicitudes de usuarios q_i al fragmento f_j.

Nótese que $(UR_{ij} + RR_{ij})$ refleja el número total de accesos de solo lectura y de actualización. Se asume que el costo total de procesar las solicitudes es el mismo. La sumatoria calcula el número total de accesos a todos los fragmentos referenciados por q_i. La multiplicación por SPC_k da como resultado el costo del acceso al sitio. Y se usa de nuevo x_{jk} para seleccionar solo los valores de costo de los sitios donde los fragmentos han sido almacenados.

Hay una cuestión muy importante que destacar aquí. La función de costo de acceso asume que procesar una consulta incluye su descomposición en un conjunto de subconsultas, cada una de las cuales se ejecuta sobre un fragmento almacenado en un sitio, y la transmisión del resultado hacia el sitio donde fue originada la solicitud. Esta es una visión muy reducida del problema, la cual no tiene en consideración la complejidad del procesamiento de la base de datos. Por ejemplo la función de costo no tiene en cuenta el costo de realizar acoples (si fuese necesario), lo cual puede ser ejecutado de varias maneras. En un modelo que sea más real que el modelo genérico, estos aspectos no deben ser omitidos.

El costo del chequeo de restricciones de integridad puede ser especificado muchas veces como la componente de procesamiento, excepto en caso de que el costo de la unidad local de procesamiento pudiera cambiar y alterar el verdadero costo de chequeo de restricciones de integridad.

Los costos operativos de la transmisión de datos para las consultas de actualización y de solo lectura son bien diferentes. En las consultas de actualización es necesario actualizar tantos sitios como réplicas existan, mientras que en las consultas de solo lectura, es suficiente con acceder a solo una de las copias. Además, al final de una solicitud de actualización no hay transmisión de datos de regreso hacia el sitio que originó la solicitud, solo un mensaje de confirmación, mientras que en las consultas de solo lectura se puede producir una enorme transmisión de datos.

La componente de actualización de la función de transmisión es:

$$TCU_i = \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} u_{ij} \times x_{jk} \times g_{o(i),k} + \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} u_{ij} \times x_{jk} \times g_{k,o(i)}$$

El primer término representa el envío del mensaje de actualización desde el sitio $o(i)$ que origina q_i, a todas las réplicas del fragmento que requieren ser actualizadas. El segundo término es para la confirmación.

El costo de las consultas de solo lectura es:

$$TCR_i = \sum_{\forall F_j \in F} \min_{S_k \in S} (r_{ij} \times x_{jk} \times g_{o(i),k} + r_{ij} \times x_{jk} \times \frac{sel_i(F_j)}{fsize} \times length(F_j) \times g_{k,o(i)})$$

El primer término en *TCR* representa el costo de transmitir la consulta de solo lectura hacia los sitios que tienen copias de fragmentos que necesitan ser accedidos.

El segundo término representa el costo para la transmisión de los resultados desde esos sitios hasta el sitio que originó la solicitud.

La ecuación establece que entre todos los sitios que contienen copias de un mismo fragmento, solo el sitio que logre el mínimo costo de transmisión total es el que debe ser elegido para la ejecución de la operación.

Resumiendo, la función de costo de transmisión para la solicitud q_i puede plantearse como:

$$TC_i = TCU_i + TCR_i$$

la cual define la función de costo total.

Restricciones

La restricción de tiempo de respuesta:

$q_i \leq \text{MaxTime}(i), \ \forall q_i \in Q$

La restricción de almacenamiento:

$\sum_{\forall F_j \in F} STC_{jk} \leq$ capacidad de almacenamiento en el sitio $s_k, \forall s_k \in S$

La restricción de procesamiento:

$\sum_{\forall q_i \in Q}$ procesar la ejecución de q_i en el sitio $s_k \leq$ capacidad de procesamiento de $s_k, \forall s_k \in S$

1.3 Métodos de solución

Se han desarrollado diversos métodos para dar solución a los problemas de optimización combinatoria. Estos métodos cuentan con enfoques exactos y heurísticos, los cuales han resuelto parte del espectro de problemas reales. En

esta sección se mencionan algunos algoritmos que usan estos métodos y las limitaciones que cada uno de ellos presenta para su aplicación.

1.3.1 Métodos exactos

Los problemas de optimización discreta pueden resolverse mediante la enumeración total, seleccionando la solución que mejor costo proporcione a través de la evaluación de cada una de las soluciones factibles del problema.

Otra clase de método es la de enumeración parcial, permitiendo resolver este tipo de problemas de manera exacta. Entre los métodos que pertenecen a esta clase se pueden citar el método de Planos de Corte, el método de Ramas y Cotas y el método de Programación Dinámica.

En las aplicaciones generalmente se involucran problemas de gran tamaño. Unas de las limitaciones en la utilización de los algoritmos exactos es que el tiempo de solución crece de manera exponencial respecto al tamaño del problema, por lo que no resulta factible su implementación (Lin, et al., 1998).

1.3.2 Métodos heurísticos

Cuando se tiene un problema de optimización combinatoria de tipo NP-completo y se desea resolver una instancia grande del mismo, se tiene como opción dar una "buena solución" al problema, aunque puede no ser la mejor. De ahí que se han desarrollado métodos denominados heurísticos. Estos métodos son procedimientos simples, a menudo basados en el sentido común, que se supone ofrecerá una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido (Lin, et al., 1998).

Las técnicas heurísticas que más éxito han tenido en la resolución de problemas de tipo combinatorio son: Recocido Simulado, Algoritmos Genéticos, Búsqueda Tabú, Grasp, Redes Neuronales y Aprendizaje Reforzado.

Existen ciertas limitaciones que involucran la utilización de estos métodos, el principal inconveniente es que se enfrentan a la posibilidad de estacionarse en óptimos locales, no obstante varias técnicas siguen diferentes estrategias para evitar este problema; de igual manera buscan "buenas soluciones" en un tiempo de cómputo razonable, pero no garantizan soluciones óptimas; además de que no es fácil describir explícitamente la cercanía al óptimo en la mayoría de los problemas.

1.4 Propuestas de solución

Se ha demostrado que el problema de dar solución al modelo de optimización antes expuesto pertenece a la clase NP-Completo (Özsu y Valduriez, 1999), por lo que el uso de algoritmos exactos requiere un costo computacional demasiado alto, y se hace necesario el uso de técnicas heurísticas para su solución. En esta investigación se emplean los métodos de Algoritmos Genéticos y Aprendizaje Reforzado.

CAPITULO II. ASPECTOS GENERALES DE LOS ALGORITMOS GENETICOS Y EL APRENDIZAJE REFORZADO

En este capítulo se muestra una descripción detallada de los principales aspectos relacionados con los Algoritmos Genéticos y el Aprendizaje Reforzado. Este marco teórico servirá para fundamentar la posterior implementación de ambas técnicas de Inteligencia Artificial, en la solución del problema de asignación en el diseño de bases de datos distribuidas.

2.1 Algoritmos Genéticos

Los Algoritmos Genéticos (AG) surgen como herramientas para la solución de complejos problemas de búsqueda y optimización, producto del análisis de los sistemas adaptativos en la naturaleza, y como resultado de abstraer la esencia de su funcionamiento.

El término Algoritmo Genético se usa por el hecho de que simulan los procesos de la evolución darwiniana, a través del uso de operadores genéticos que operan sobre una población de individuos que "evoluciona" de una generación a otra.

El desarrollo de toda la teoría relacionada con el tema no solo ha servido para lograr un método eficiente de búsqueda, sino que ha permitido abstraer y explicar rigurosamente el proceso adaptativo de los sistemas naturales.

2.1.1 ¿Qué son los Algoritmos Genéticos?

Los Algoritmos Genéticos son métodos de búsqueda de propósito general basados en los principios de la genética natural, es decir, son algoritmos de búsqueda basados en los mecanismos de la selección natural y la genética. En ellos se mantiene una población que representa a un conjunto de posibles soluciones, la cual es sometida a ciertas transformaciones con las que se trata de obtener nuevos candidatos y a un proceso de selección sesgado en favor de los mejores.

Son un ejemplo de método que explota la búsqueda aleatoria "guiada" que ha ganado popularidad en los últimos años debido a la posibilidad de aplicarlos en una gran gama de campos y a las pocas exigencias que impone al problema.

2.1.2 Terminología usada

En el trabajo con AG se maneja una serie de términos "importados" de la genética natural. No siempre es adecuada la analogía, pero estos son comúnmente aceptados:

- Población: Conjunto de individuos o cromosomas. Equivale a una muestra aleatoria del espacio de solución o un conjunto de soluciones alternativas.
- Cromosoma: Un cromosoma es un portador de la información genética que transmite cada uno de sus genes. Una posible solución.
- Gen: Cada uno de los rasgos o características que conforman el cromosoma. También se les llama parámetros o aspectos.
- Genotipo: En biología se le llama al "paquete" genético total en su forma interna. En la terminología de AG será la información genética de todo el cromosoma en forma codificada.
- Fenotipo: Se le llama en genética al paquete genético tal y como interactúa con el medio exterior. En los AG artificiales serían los aspectos del cromosoma decodificados.
- Locus: Es la posición de un gen el cromosoma.
- Alelo: Es el valor asociado a un gen.

2.1.3 Componentes de un Algoritmo Genético

Los AG trabajan a partir de una población inicial de estructuras artificiales que se van modificando repetidamente mediante la aplicación de los siguientes operadores genéticos:

- Operador de selección o Darwiniano.
- Operador de cruzamiento o Mendeliano.
- Operador de mutación.

En el uso de los AG es necesario encontrar una posible estructura para representar las soluciones. Pensando este asunto como el problema de buscar en un espacio de estados, una instancia de esta estructura representa un punto o un estado en el espacio de búsqueda de todas las posibles soluciones. Así, una estructura de datos en el AG consistirá en uno o más cromosomas (frecuentemente uno), el cual se representa comúnmente como una cadena de bits, pues existen otras representaciones.

Cada cromosoma (cadena) es una concatenación de un número de subcomponentes llamados genes. La posición de un gen en el cromosoma se conoce como locus y sus valores como alelos (Gálvez, 1998).

Al optimizar una estructura usando un AG se necesita una medida de su calidad en el espacio de búsqueda. La función de adaptabilidad es la encargada de esta tarea. Esta función se obtiene directamente a partir de la función de evaluación $f(x)$ del correspondiente problema. La función de evaluación no tiene por qué estar expresada de forma cerrada como una función objetivo clásica; basta con que proporcione un índice de idoneidad para cada uno de los candidatos a solución que se le presenten. De todos modos, es conveniente que el procedimiento de obtención de dicho índice se pueda implementar con facilidad en una computadora, dado que la evolución de todo el algoritmo va a depender de él.

Los AG realizan una maximización por defecto, para los problemas de minimización los valores de la función objetivo pueden ser negados con vistas a tomar valores positivos para producir así la adaptabilidad.

2.1.4 Funcionamiento de un Algoritmo Genético

Un AG simple genera una población inicial de n estructuras (cadenas, cromosomas o individuos).

Los operadores actúan transformando la población: el operador de selección o Darwiniano realiza la selección de las cadenas de acuerdo a su adaptabilidad para el posterior apareamiento, incrementa geométricamente la presencia de los individuos aventajados y reduce la presencia de los retrasados, aunque no introduce nuevos individuos; el operador de cruzamiento o Mendeliano realiza la recombinación del material genético de dos cadenas padres; el operador de mutación produce una alteración de un gen dentro de un cromosoma o cadena a sus diferentes formas alelomorfas, introduce variedad en el juego de esquemas de la población y proporciona un mecanismo de seguridad frente a posibles pérdidas de información valiosa, visto de esta manera, es un operador secundario, de ahí que se aplique con bastante menor frecuencia que el cruce (Pérez, 1996).

Para cada uno de estos operadores está asociado el uso de probabilidades y la generación de números aleatorios.

Una vez completada la acción de los tres operadores se dice que ha transcurrido un ciclo generacional.

Luego se repite el mismo proceso mientras no se garantice el criterio de parada, figura 2.1.

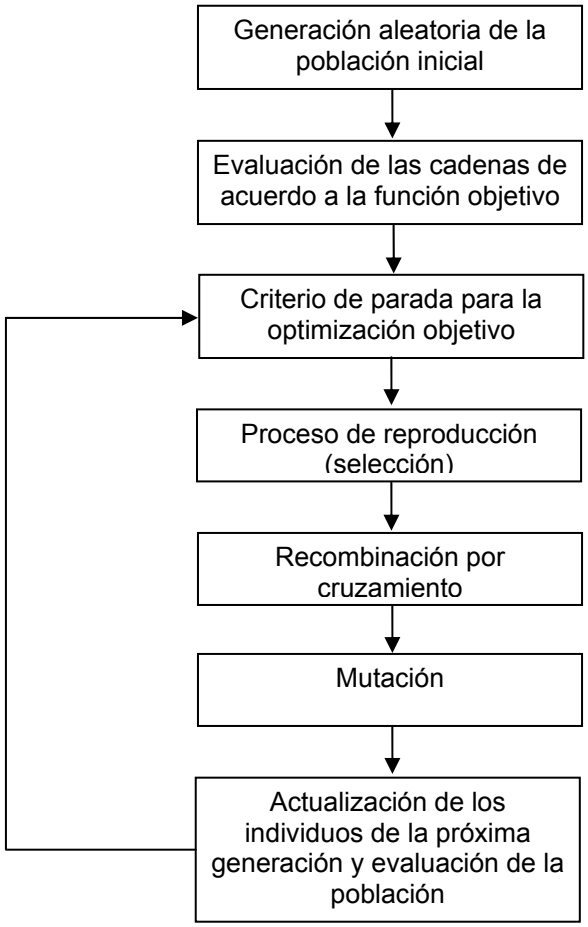


Figura 2.1. Diagrama funcional de un Algoritmo Genético. (Gálvez, 1998)

2.1.5 Principales elementos y operadores de un Algoritmo Genético

En esta sección se explicarán detalladamente los diferentes componentes de un AG, los operadores y sus variantes.

2.1.5.1 Codificación

Dado un problema factible de ser representado por un conjunto de parámetros conocidos como genes, se le llama codificación al proceso de unir los mismos para formar una cadena de valores (cromosoma).

En genética, ese conjunto representado por un cromosoma en particular es referido como genotipo. Este contiene la información necesaria para construir un organismo conocido como fenotipo. Esos mismos términos se aplican en los AG.

La adaptación de cada individuo depende de su fenotipo, el cual es posible inferir de su genotipo, puede calcularse desde el cromosoma si se utiliza la función de adaptación.

Las cadenas binarias han sido utilizadas tradicionalmente para la codificación debido a la simplicidad de su implementación y porque esta representación maximiza el número de esquemas producidos. Los operadores de cruzamiento y mutación son propios de las cadenas binarias. Cuando el alfabeto usado para codificar las soluciones es superior al binario, es necesario redefinir los operadores de cruzamiento y mutación adecuadamente.

Existen otros tipos de codificaciones, como la codificación dinámica, que permite variar el número e interpretación de los bits asignados a un parámetro particular a través de la ejecución; las representaciones de longitud variable; y la codificación delta, cuyos bits en las cadenas expresan una distancia con respecto a alguna solución parcial previa (Gálvez, 1998).

2.1.5.2 Elementos relativos a la población

El análisis de la población incluye varios elementos que van desde la cantidad de individuos que la integran hasta el número que se reemplazan en cada iteración.

Tamaño de la población

El tamaño de la población es una de las elecciones más importantes a la hora de sintonizar un AG debido a su influencia en la convergencia del mismo. Si el tamaño de la población es demasiado pequeño tendrá una acusada tendencia a la convergencia prematura y a representar pobremente el espacio de soluciones; por otro lado si es demasiado grande seguramente desperdiciará recursos computacionales sin obtener mejoras apreciables. Por eso es conveniente estudiar las posibilidades de conseguir un mecanismo que adapte el tamaño de la población a las circunstancias particulares de la búsqueda (Pérez, 1996).

Sobre esta disyuntiva y como un trabajo teórico, Goldberg obtuvo en su investigación que el tamaño óptimo de una población de cadenas binarias crece exponencialmente con la longitud de la cadena (Goldberg, 1989).

Recientemente se ha desarrollado una variedad de Algoritmos Genéticos que hace evolucionar el tamaño de la población, se comienza con una población inicial pequeña y se va ampliando según progrese la evolución.

Generación de la población inicial

En términos generales la población inicial debe ser lo más variada posible. Es necesario que la distribución de aptitudes sea uniforme para evitar la convergencia prematura.

En la práctica se utilizan fundamentalmente dos vías para obtener la población inicial con que el AG comienza su trabajo:

1. Generación aleatoria de los individuos:

   La manera más simple de generar la población inicial es seleccionar cada carácter de la cadena de forma totalmente aleatoria, hasta completar toda la población, teniendo en consideración el rango de validez de cada uno. En caso de que el alfabeto sea binario, la probabilidad de que cada bit sea 1 es 50%.

2. Sembrado de individuos:

   El conocimiento específico puede ayudar para elegir una población inicial factible y/o próxima al óptimo. Conviene que la población inicial contenga la mayor cantidad posible de puntos factibles. En problemas fuertemente restringidos es difícil obtener puntos factibles y se deberá comenzar a iterar con una cantidad muy reducida de ellos. En caso de que eso ocurra no es conveniente usar un tamaño de población muy pequeño dado que perjudica gravemente al requisito de diversidad, ni tampoco introducir una gran cantidad de puntos no factibles ya que se reduce grandemente la eficiencia del Algoritmo Genético.

   Tradicionalmente se ha resuelto este problema repitiendo los puntos factibles en la población inicial, ello no elimina los inconvenientes señalados, pero reduce su peligrosidad (Holland, 1992). Se han desarrollado trabajos que han encontrado que el sembrado en una población con soluciones de alta calidad obtenidas de otra técnica heurística, ayuda al AG a encontrar mejores soluciones más rápido que con un comienzo aleatorio (Gálvez, 1998).

2.1.5.3 Mecanismos de reproducción o selección

Este proceso se encarga de la elección, según ciertos criterios, de los mejores individuos de una población específica. Los individuos seleccionados participan en el cruce.

Los mecanismos de selección son muy variados, distinguiéndose tres grupos fundamentales según el grado de intervención del azar en el proceso:

1. Selección directa: Se toma un subconjunto de individuos de la población siguiendo un criterio fijo, del estilo de “los k mejores”, ”los k peores”, etc.

2. Selección aleatoria simple o equiprobable: Se asignan a todos los elementos de la población base las mismas probabilidades de formar parte de la muestra, y se constituye la misma mediante ensayos de Bernuolli simples.
3. Selección estocástica: Se asignan probabilidades de selección o puntuaciones a los elementos de la población base en función (directa o indirecta) de su aptitud. La muestra se constituye de modo estocástico haciendo uso de dichas puntuaciones. De este modo, no se realiza un proceso determinista a la hora de la selección, lo cual permite una simulación más exacta del proceso natural.

Al implementar AG se usan diferentes mecanismos estocásticos de selección en dependencia del problema. Para ilustrarlos se considerará el problema de seleccionar dos individuos de la población $P = \{x_1, x_2, x_3, x_4\}$, siendo las puntuaciones asociadas $p_1 = 0.1$, $p_2 = 0.3$, $p_3 = 0.1$ y $p_4 = 0.5$ respectivamente.

Selección por el método de la ruleta

Se consideran las puntuaciones estrictamente como probabilidades de elección para formar la muestra, y se constituye la misma realizando k ensayos de una variable aleatoria con dicha distribución de probabilidades.

Dado que habitualmente solo se dispone de un generador de números aleatorios simples (uniformemente distribuidos entre 0 y 1), para simular un ensayo de la distribución $\{p_1,..., p_n\}$ se hace lo siguiente:

1. Se evalúa la función de adaptabilidad en cada individuo.
2. Se calcula la suma total de las evaluaciones.
3. Se calcula la puntuación asociada a cada individuo como la fracción de la suma total que le corresponde.
4. Se calculan las puntuaciones acumuladas

$$q_0 = 0$$

$$q_i = p_1 + ... + p_i \qquad (\forall i = 1,..., n)$$

5. Se genera un número aleatorio simple $r \leftarrow \mathrm{URand}[0,1)$.
6. Se elige el individuo x_i que verifique

$$q_{i-1} \le r < q_i$$

Para seleccionar k individuos se modifican los dos últimos pasos:

5. Se generan k números aleatorios simples $r_j \leftarrow \text{URand}[0,1)$, $(\forall j = 1,...,k)$
6. Para cada $j = 1,...,k$ se elige el individuo x_i que verifique

$$q_{i-1} \leq r_j < q_i$$

Para el ejemplo propuesto las puntuaciones acumuladas son

$q_1 = 0.1$, $q_2 = 0.4$, $q_3 = 0.5$ y $q_4 = 1.0$ respectivamente

y los $k = 2$ números aleatorios simples son: $r_1 = 0.02433$ y $r_1 = 0.51772$

De esta manera el primer elemento de la muestra será x_1 dado que $q_0 < r_1 < q_1$ y el segundo será x_4 dado que $q_3 < r_2 < q_4$.

Cuando se genera un número aleatorio, la probabilidad de que esté en el intervalo de un individuo estará en función de su puntuación y por lo tanto de su adaptación al medio.

Nótese que existe la posibilidad de que algunos individuos puedan ser elegidos varias veces en una muestra (los que tengan mayor puntuación o un golpe de buena suerte con los r_j) y que otros individuos pueden no ser elegidos nunca (los que tengan menor puntuación o mala suerte).

Debido a sus buenas propiedades teóricas este es el método de selección que se utiliza por defecto en los AG. No obstante, por diversos motivos que se verán a continuación, sus prestaciones en la práctica no están siempre a la altura de las expectativas teóricas (Davis, 1991).

Otras variantes de selección

Sucede que en las generaciones iniciales de un AG los valores de adaptabilidad promedio son bajos. La presencia de algunas cadenas con un valor de adaptabilidad relativamente alto provoca que el mecanismo de selección proporcional asigne un número grande de copias a estas supercadenas provocando así la convergencia temprana. La causa de este fenómeno está en la misma esencia del AG de seleccionar los mejores por la regla usual de selección (la ruleta, $p_{select} = f_i / \sum f$) para que pasen a formar parte de la generación siguiente. Lo más seguro es que el individuo más destacado de la población, aun cuando es mediocre, logre apropiarse de la próxima generación y por lo tanto no hay posibilidades de continuar la evolución. De esta forma se ha logrado la

convergencia sin asegurarse que el valor a donde se converge es en realidad un valor bueno (Goldberg, 1989).

Por otra parte, en las postrimerías de la búsqueda todos los individuos son demasiado "buenos", entonces la varianza entre los valores de adaptabilidad de las cadenas es pequeña y por tanto el esquema de selección proporcional asigna más o menos igual número de copias para todas las cadenas, la población se mantendrá por ese orden de bondad y será muy difícil buscar individuos aun mejores, salvo por un golpe de "suerte" logrado mediante la mutación o el cruzamiento.

Selección basada en el rango

Un enfoque para evitar los problemas de la selección proporcional antes mencionados es el de ignorar la función objetivo actual y usar un procedimiento de ranqueo. Se clasifican las cadenas de acuerdo a sus valores de adaptabilidad. Los padres son seleccionados usando la siguiente distribución de probabilidad:

$$p[k] = 2k/n(n+1)$$

donde $[k]$ es la k-ésima cadena ordenada en forma ascendente y n es el tamaño de la población. La mejor cadena de la población, la $[n]$, tiene una probabilidad de ser seleccionada igual a $2/(n+1)$, el doble que la media cuya probabilidad es de $1/n$. Algunos especialistas en el tema han generalizado este método de ranqueo y argumentan el hecho de preferir el ranqueo al escalado de la función objetivo.

Selección de torneo

Una alternativa que combina la idea de ranqueo con el mecanismo de selección de la ruleta es la selección de torneo. Cada elemento de la muestra se toma eligiendo el mejor de los individuos de un conjunto de z elementos tomados al azar de la población base, esto se repite k veces hasta completar la muestra. El parámetro z suele ser un entero pequeño comparado con el tamaño de la población base, normalmente 2 ó 3 (Pérez, 1996).

Técnica del resto estocástico

A cada individuo x_i se le asignan directamente $\lfloor p_i \cdot k \rfloor$ puestos en la muestra. Seguidamente los individuos se reparten los puestos vacantes en función de sus puntuaciones. El reparto suele hacerse por el método de la ruleta y entonces se le dice muestreo estocástico por restos.

Para el ejemplo propuesto, a x_4 le corresponde un puesto en la muestra, el individuo que ocupe el otro puesto se elige por el método de la ruleta, se genera un número aleatorio simple $r = 0.39874$ y se compara con las puntuaciones acumuladas q_i, en este caso el puesto vacante le corresponde a x_2 dado que $q_1 < r < q_2$.

Existe una variante en la que el reparto de las vacantes se hace por muestreo directo; en esta variante no hay ninguna intervención del azar, no es un método de muestreo estocástico sino directo, por este motivo se le llama muestreo determinista por restos, para distinguirlo del descrito anteriormente.

Elitismo

El método más utilizado para mejorar la convergencia de los AG es el elitismo. Consiste básicamente en seleccionar una élite de r miembros de entre los mejores de una población e incorporarlos directamente a la población de la siguiente generación, sin pasar por los mecanismos de cruzamiento y mutación. Esto garantiza que los mejores individuos de una población sobrevivan de generación en generación. Comúnmente el tamaño de la élite r es bastante pequeño (1 ó 2 para $n = 50$).

Bajo ciertas condiciones muy generales la introducción del elitismo garantiza la convergencia teórica al óptimo global; en la práctica mejora la velocidad de convergencia de los AG cuando la función de evaluación es unimodal (no existen óptimos locales), sin embargo la velocidad de convergencia empeora con funciones fuertemente multimodales.

Con tamaños de población pequeños se consiguen efectos similares a los del elitismo introduciendo reinicializaciones periódicas en los AG, cada vez que el AG converge se salvan los mejores individuos, se reinicializan los demás y se vuelve a comenzar. La reinicialización tiene efectos beneficiosos sobre las prestaciones del método debido a que introduce diversidad, requisito especialmente crítico en los AG con poblaciones pequeñas.

Escalado

Una de las maneras de controlar la diversidad de las aptitudes es definiendo una función de aptitud neta, distinta de la función de evaluación. Esto es lo que se conoce como escalado de la función de evaluación. El escalado no usará

directamente la evaluación de la función objetivo para calcular las probabilidades, la transformará de forma que se lleve a otra escala. Mediante el escalado se trata de evitar un fenómeno con causas opuestas al anterior, pero de similares efectos: es el fenómeno ya descrito de la convergencia prematura hacia superindividuos.

En último término se trata de controlar los efectos de la presión selectiva amortiguándolos en las fases tempranas de la evolución y acentuándolos en las fases tardías.

Esencialmente se usan cuatro mecanismos de escalado: el escalado lineal, el truncamiento sigma, el escalado potencial y los métodos de escalado basados en el orden; siendo el escalado lineal uno de los más simples y poderosos (Pérez, 1996).

2.1.5.4 Cruzamiento

El proceso de reproducción solo garantiza la supervivencia de los mejores pero no hace que la generación cambie. La repetición de este proceso solo lleva a un reforzamiento de la presencia del mejor y a la desaparición de los peores.

La obtención de individuos nuevos a partir de los que existen es una de las características más interesantes e importantes en el trabajo con los AG. A este proceso se le llama cruzamiento, a semejanza del proceso natural.

Este operador permite el cambio de código genético entre los individuos, introduciendo así instancias de nuevas combinaciones de genes, facilita al AG una búsqueda con eficiencia en espacios de varias dimensiones.

Para realizar el cruzamiento es necesario determinar el punto que definirá que parte de cada cadena de los padres irá a cada hijo. El número J que se usará para indicar el punto de cruce en cada cadena de longitud N estará entre 1 y N, significando que cada cadena se divide en dos y se intercambian sus partes:

- una desde el gen 1 hasta el gen anterior a J
- otra desde el gen J hasta el gen N

Ejemplo:

Padres: 1 0 **1** 0 1 1 1 1 1 **0** 1 1 1 0

Hijos: 1 0 0 1 1 1 0 1 1 1 0 1 1 1

Si el número generado es 1, la primera porción de la cadena quedará vacía lo cual indica que en realidad no se está realizando un intercambio de información, pues

cada hijo será exactamente igual a uno de sus padres. Esto no puede considerarse un error pero sí es importante tenerlo en cuenta en el funcionamiento del AG.

Teniendo esto en cuenta se trabaja con un parámetro denominado Probabilidad de Cruzamiento, el cual se corresponde con la probabilidad de que el número aleatorio sea mayor que 1. Esta probabilidad es la de que en realidad se creen hijos nuevos, por lo que debe tener un valor alto para lograr una mayor introducción de soluciones nuevas del problema en cuestión.

De esta forma para que el AG realice el cruzamiento tiene que generar dos números aleatorios: uno para determinar si ocurrirá o no el cruzamiento en función de la probabilidad de cruzamiento definida y otro para, si debe ocurrir el cruzamiento, seleccionar cuál será el punto de la cadena donde ocurrirá el cruce. En este caso, el segundo número estará entre 2 y N.

La selección del punto de cruce deberá hacerse para cada pareja. Lo mismo ocurre con la decisión de si ocurrirá o no cruzamiento entre un par de individuos, lo cual, como es lógico deberá decidirse antes de seleccionar el punto de cruce.

El cruzamiento no tiene necesariamente que provocar la obtención de un individuo mejor, su misión principal es la exploración de un nuevo camino.

Alternativas del cruce clásico

Se han propuesto muchas alternativas que mejoran, generalmente, las prestaciones del cruce clásico. Las alternativas más importantes son:

- Cruce multipunto: Consiste en cruzar los individuos en torno a dos o más puntos. El cruzamiento de múltiples puntos trata cada cadena como un anillo de bits dividido por k puntos de cruce en k segmentos. Los segmentos alternados son intercambiados entre el par de cadenas a entrecruzar (Gálvez, 1998). Este cruce mejora la capacidad de procesamiento de los individuos a costa de perder velocidad de convergencia.
- Cruce segmentado: Es una versión del cruce multipunto que permite la introducción de variabilidad en el número de puntos de cruce. Consiste en reemplazar el número fijo de puntos de cruce por una probabilidad de segmentación p_{seg} que da cuenta de la posibilidad de que se produzca un cruce al llegar a cierto punto de la cadena. Así se trata de corregir la asimetría con respecto a la mutación (Pérez, 1996).

- Cruce uniforme: Para cada bit del primer descendiente se decide, con probabilidad p_{unif}, de qué progenitor heredará su valor en esa posición. El segundo descendiente recibe el correspondiente gen del otro progenitor. Dado que el cruce uniforme intercambia más bien genes que bloques constructivos, se usa para combinar atributos específicos, con independencia de la posición en que han sido codificados. En algunos problemas esta capacidad compensa el riesgo adicional que se introduce al descomponer bloques constructivos. No obstante se recomienda utilizar el cruce uniforme solo en casos concretos en los que haya motivos fundados para hacerlo.

Otros mecanismos que se pueden añadir a cualquiera de los operadores de cruce son: los cruces anulares, cruces externos, cruces con barajado, cruces con adaptación y los cruces solitarios.

2.1.5.5 Mutación

La mutación es un proceso similar al biológico en el cual un gen de un individuo muta o cambia a otro valor posible.

Este proceso es muy importante ya que puede ser que un individuo malo tuviera alguna característica muy buena. Cuando este individuo pasa por el proceso de reproducción existe una alta probabilidad de que sea eliminado y por lo tanto se pierda esa característica deseable. La recuperación de esta característica puede ser prácticamente imposible a través de los otros mecanismos genéticos.

La casualidad, que a veces permite hallazgos importantes que por los caminos lógicos podría demorarse mucho, es simulada a través de la mutación.

El proceso es muy simple y solo consiste en determinar cuál es el gen que mutará en el individuo. La selección de este gen se realiza de forma aleatoria. Una vez que se obtiene solo es necesario cambiar el valor que está en esa posición de la cadena por otro posible.

Es importante tener el control de la mutación ya que estos cambios pueden tener tanto un efecto conveniente como perjudicial; para ello se utiliza un parámetro llamado Probabilidad de Mutación, el cual servirá para determinar si un individuo mutará o no (Gálvez, 1998).

La mutación es un operador de importancia teórica secundaria, lo que se manifiesta en su baja probabilidad de aplicación en comparación con el cruce. Ello no resta para que en la práctica sea un operador imprescindible, de hecho, se ha

comprobado que un AG puede funcionar sin realizar cruces, pero no sin realizar mutaciones (Pérez, 1996).

2.1.5.6 Criterios de terminación

Hasta el momento se ha aceptado por defecto como criterio de terminación de los AG el máximo número de iteraciones. Se admite que dada la hipótesis de la construcción por bloques y una adecuada población inicial, los mejores individuos de la población se acercarán cada vez más al óptimo. Es decir, se asume implícitamente que al llegar a cierto valor de generaciones el algoritmo ha convergido a efectos prácticos. Esta manera de determinar la convergencia es excesivamente arbitraria en muchos casos, lo que lleva a plantear otros criterios de terminación.

Tradicionalmente se distinguen dos tipos de criterios de terminación.

Criterios de terminación enfocados al costo

Abreviadamente, criterios de "tipo MAX". Son los que limitan a priori el número máximo de iteraciones o el máximo número de evaluaciones, habitualmente usados cuando la función de evaluación es complicada. Son los más sencillos, pero no siempre tienen significado, ya que normalmente no hay ninguna razón para dar a dichos máximos uno u otro valor. En tal circunstancia o bien ocurrirá que el AG no esté lo suficientemente cerca del óptimo al terminar o bien consumirá recursos computacionales inútilmente, estando ya lo suficientemente cerca del óptimo.

Criterios de terminación enfocados a la calidad

También se les conoce como criterios incrementales de terminación o abreviadamente criterios de tipo "TOL". Se hace parar al algoritmo una vez que se hayan superado ciertos requisitos de convergencia, independientemente de la iteración en que se encuentre. Dependiendo del criterio práctico que se utilice para evaluar la convergencia se dividen en dos tipos:

a) Enfocados a la estructura: Se evalúa la convergencia de la población verificando la cantidad de genes que han convergido. Se considera que un gen del genotipo ha convergido cuando cierto porcentaje de la población (alrededor del 90%) tiene los alelos iguales en dicho gen (o parecidos, si la representación no es binaria). La búsqueda termina cuando el número de genes que ha

convergido excede cierto porcentaje (alrededor del 95%) del total de genes de que consta el genotipo.

b) Enfocados al contenido: Miden el progreso hecho por el algoritmo en cierto número de iteraciones; si es menor que cierta tolerancia TOL, que es un parámetro del método, la búsqueda ha terminado. Habitualmente se mide el progreso hecho por el algoritmo como el incremento relativo de la aptitud total de la población; también se puede usar la aptitud máxima o la mínima, según sea el problema.

La elección de unos valores u otros para las tolerancias es delicada, no solo depende del problema, sino que también influye la suerte que se haya tenido en la ejecución del AG. Además, una pequeña reducción de las tolerancias permitidas puede conllevar a un tiempo de ejecución mucho más grande.

Habitualmente se usa un criterio de terminación híbrido que consta de una tolerancia y un mecanismo de tipo MAX para cuando se alargue demasiado la búsqueda (Pérez, 1996).

2.1.5.7 Tratamiento de las restricciones

Una de las limitaciones del AG básico es que carece de mecanismos para considerar las posibles restricciones o ligaduras que pudieran imponerse a la búsqueda. El problema en concreto consiste en que para poder usar la representación binaria se debe codificar sobre un superconjunto del dominio del problema $\mathrm{X} \supseteq X$.

El problema surge debido a la rigidez de la representación, no se puede esperar que el espacio de búsqueda del AG, X, coincida con el dominio del problema X; razonablemente existirán en el espacio de búsqueda puntos no factibles que habrá que tratar de alguna manera.

Tradicionalmente se han empleado diferentes tipos de técnicas para resolver este problema, cada una con sus ventajas e inconvenientes.

Técnicas de penalización

Consisten en resolver directamente el problema penalizando a los individuos no factibles. En los AG se lleva a cabo generando individuos sin tener en cuenta su factibilidad reduciendo la aptitud de los que resulten ser no factibles. La única exigencia de estas técnicas es que debe poderse medir de alguna manera el grado

de no factibilidad para así dar mayor penalización a los puntos más alejados del dominio del problema.

Las funciones de penalización muy severas no son recomendables en problemas en que hay muchas probabilidades de generar individuos no factibles. Esto se debe a que no solo se corre el riesgo que el AG gaste la mayor parte de su tiempo en procesar individuos no factibles, sino que además es muy probable que cuando se encuentre un individuo factible, este destaque ostensiblemente sobre los demás por el mero hecho de ser el único factible y provoque una convergencia prematura.

Por otra parte, las funciones de penalización muy flexibles tienen el riesgo de dejar prosperar a individuos que aun siendo no factibles tengan una aptitud neta mayor que otros factibles.

Técnicas de reparación o corrección

Consisten en habilitar un procedimiento para corregir cualquier solución no factible que se genere. Los inconvenientes de estas técnicas residen en que suele ser difícil encontrar un procedimiento de corrección lo suficientemente general y si se encuentra, normalmente consume muchos recursos de computación. Además, tal procedimiento es necesariamente específico para cada problema.

Técnicas de decodificación

Consisten en realizar la codificación de modo tal que resulte muy poco probable, idealmente imposible, generar soluciones no factibles, dicho de otro modo, se trata de modificar X a través de la codificación para que se aproxime lo máximo posible a X. Normalmente, implica la modificación de los operadores genéticos para que conserven la factibilidad de los individuos.

Los inconvenientes de estas técnicas son básicamente los mismos que los de las técnicas de reparación.

Paradigma del recuerdo del comportamiento

Bajo este nombre se conoce a una técnica de consideración de las restricciones, específicamente desarrollada para los AG, que consiste básicamente en lo siguiente: se encadenan dos AG, en el primero la función de aptitud mide exclusivamente el grado de satisfacción de las restricciones de modo parecido a como lo haría una función de gratificación; el segundo AG toma como población inicial la población final del primero, la cual constituye un recuerdo de las restricciones, y evoluciona según la aptitud neta del problema que se ha

modificado para que devuelva cero cuando no se satisfagan las restricciones (Pérez, 1996).

2.1.5.8 Asignación de los parámetros de control

Para que un Algoritmo Genético pueda funcionar se deben dar valores apropiados a un conjunto de parámetros, entre los que destacan el tamaño de la población, la longitud de los individuos, las probabilidades de mutación y cruce, el máximo número de iteraciones, la precisión o tolerancia y otros específicos de cada método como por ejemplo el parámetro z en la selección de torneo.

No existen criterios definitivos para dar valores a esos parámetros, por lo que esta tarea depende en gran medida del buen criterio del programador. En principio, los Algoritmos Genéticos están reputados como técnicas de búsqueda paramétricamente robustas, es decir, funcionan, mejor o peor, pero funcionan en un amplio rango de valores de sus parámetros. Sin embargo, las diferencias de eficiencia en la práctica son lo suficientemente espectaculares como para solicitar una mayor atención en dicha elección.

Se ha realizado una gran cantidad de pruebas sobre una amplia variedad de problemas de búsqueda y optimización con la intención de determinar rangos de buen funcionamiento para los parámetros del algoritmo. Aunque los resultados medidos en términos de prestaciones son dependientes en mayor o menor medida del problema en particular, se pueden extraer las siguientes conclusiones generales:

- El tamaño de la población varía habitualmente entre 50 y 100 individuos; valores menores suelen plantear graves problemas de convergencia prematura y valores mayores requieren un gran esfuerzo computacional sin obtener mejoras apreciables.
- El tamaño de los individuos suele venir dado como una consecuencia de los criterios de representación y codificación del problema. El único consejo que cabe dar es el de no alargar innecesariamente dicha cantidad salvo si se tiene la garantía de que al introducir redundancias se van a obtener prestaciones muy superiores.
- Las mejores prestaciones se obtienen en general con tasas de cruce que varían entre el 20% y el 60% y tasas de mutación entre el 0.1% y el 5%, aunque no es raro ver otros valores (Levine, 1995). Incrementando la probabilidad de

cruzamiento se incrementa la recombinación de bloques de construcción pero también se incrementa la destrucción de buenas cadenas. Por otro lado, incrementando la probabilidad de mutación se tiende a transformar la búsqueda genética en una búsqueda aleatoria, introduciendo pérdidas en el material genético (Gálvez, 1998).

- El máximo número de iteraciones depende de la precisión especificada y varía mucho de un problema a otro; sirven como valores de orientación 50 para problemas de evaluación compleja y 1000 para problemas de evaluación sencilla. En cuanto a la tolerancia de error se hace notar que está fuertemente condicionada por la representación elegida y por la "rugosidad" de la función de evaluación. Para problemas reales de ingeniería se usan valores entre 10^{-2} y 10^{-5}.

La robustez paramétrica garantiza el buen funcionamiento del Algoritmo Genético clásico con esos valores para una gran variedad de problemas de optimización; pero cuando el problema no está dentro de esa gran variedad lo común es que una parametrización como la anterior proporcione mediocres prestaciones (Pérez, 1996).

Es necesario tener en cuenta la interacción entre los diferentes operadores genéticos. El propio problema de seleccionar el juego de parámetros óptimos para el Algoritmo Genético puede convertirse en un problema de optimización no lineal complejo (Gálvez, 1998).

Ajuste automático de los parámetros de un Algoritmo Genético

Posiblemente el procedimiento más práctico y que mejor se adapta a la filosofía de la computación evolutiva es el método de los parámetros evolutivos.

La idea básica consiste en someter a los parámetros de aplicación de los operadores genéticos a un proceso de evolución análogo al de los AG, con la idea final de utilizar con más frecuencia los operadores que hagan un buen trabajo. A cada operador se le asigna una aptitud operacional en función del incremento de aptitud que proporcione a la población su aplicación en un instante dado y se actualiza periódicamente.

El método de los parámetros evolutivos realiza un ajuste dinámico de los parámetros de un AG de manera que en cada momento se tiene presumiblemente la mejor parametrización posible. Esto presenta el inconveniente de que un

operador que sea importante únicamente en las fases finales de la evolución puede perderse en las primeras etapas, para evitarlo se debe dar a los operadores con una probabilidad nula de aplicación una aptitud no nula que les permita volver a funcionar cuando el momento sea propicio. Este método es también muy útil para probar nuevos operadores genéticos, pues en el propio proceso de evolución se irán desechando los que sean inservibles (Heitkotter y Beasley, 1996).

2.1.6 Implementaciones de un Algoritmo Genético

Existen varias formas de trabajo de los Algoritmos Genéticos, cada una basada en una metáfora distinta de la Naturaleza.

Algoritmos Genéticos generacionales

Es semejante a la forma de reproducción de los insectos, donde generación pone huevos, se aleja geográficamente o muere y es sustituida por una nueva. En tal modelo se realizan cruces en un conjunto de individuos y los descendientes son colocados en otro. Al final de la fase reproductiva se elimina la generación anterior y se utiliza la nueva. Este modelo también es conocido como AG canónico.

Algoritmos Genéticos de estado fijo

Utilizan el esquema generacional de los mamíferos y otros animales de vida larga, en el cual coexisten padres y sus descendientes, lo que permite que los hijos sean educados por sus progenitores, pero también que a la larga se genere competencia entre ellos.

En este modelo no solo se deben seleccionar los dos individuos padres, sino también cuáles de la población anterior serán eliminados para dar espacio a los descendientes.

Algoritmos Genéticos paralelos

Parte de la metáfora biológica que motivó a utilizar la búsqueda genética consiste en que es inherentemente paralela, ya que al evolucionar se recorren de forma simultánea muchas soluciones, cada una representada por un individuo de la población. Sin embargo, es muy común en la naturaleza que no solo exista una población evolucionando, sino varias, normalmente aisladas de forma geográfica, que originan respuestas diferentes a la presión evolutiva. Esto trae consigo dos modelos que tienen en cuenta tal variación y utilizan no una población como los anteriores, sino múltiples concurrentemente.

a) Modelos de islas: Una población de individuos se divide en subpoblaciones que evolucionan independientemente como un Algoritmo Genético normal. En ocasiones ocurren migraciones entre ellas, lo que les permite intercambiar material genético. Con la migración, se puede explotar las diferencias en las subpoblaciones. Tal variación representa una fuente de diversidad genética. Sin embargo, si un gran número de individuos emigra en cada generación ocurre una mezcla global y se eliminan las diferencias locales. Por otro lado, si la migración no es frecuente es probable que se produzca convergencia prematura en las subpoblaciones.

b) Modelo celular: Coloca cada individuo en una matriz y solo podrá reproducirse con los otros que tenga a su alrededor, por lo que escoge al azar o al mejor adaptado. El descendiente pasará a ocupar una posición cercana.

   No hay islas en este modelo, pero hay efectos potenciales similares. Si se asume que el cruce está restringido a individuos adyacentes, dos de ellos separados por muchos espacios estarán tan separados como si estuvieran en dos islas. Esta forma se conoce como aislamiento por distancia (Águila, 2001).

2.2 Aprendizaje Reforzado

El Aprendizaje Reforzado (AR) ó Reinforcement Learning (RL) en inglés, es un conjunto de técnicas diseñadas para dar solución a problemas de decisión secuencial o control óptimo, cuya base son los procesos de decisión markovianos. Los procesos markovianos son procesos estocásticos de decisión que se basan en la propiedad de que la acción a tomar en un estado determinado, en un instante determinado, depende solo del estado en que se encuentre el sistema en el momento de tomar la decisión.

La clave del AR está en la interacción continua entre un agente y su ambiente. El agente ejercita acciones de control que influyen en el estado del proceso y recibe una señal de recompensa o penalización, dependiendo del estado resultante de cada acción tomada. De este modo adquiere la experiencia necesaria para mejorar su comportamiento en el futuro.

El aprendizaje ha sido el factor clave de los sistemas inteligentes ya que tiene como meta lograr que los robots realicen tareas sin la necesidad de una programación explícita de ellas (Brooks, 1991).

El problema de aprendizaje en los robots es un caso especial del problema de

aprendizaje de máquinas. Esta rama de la Inteligencia Artificial (IA) pretende reemplazar la programación explícita de las máquinas por la enseñanza (Koening y Simmons, 1998).

La enseñanza en las máquinas significa la transmisión de conocimientos, la explicación de alguna tarea o cualquier otra forma de instrucción (Arkin, 1998).

La investigación en máquinas de aprendizaje se divide en dos áreas: Aprendizaje Supervisado y Aprendizaje No Supervisado. Las metodologías de tipo supervisado a su vez se dividen en Aprendizaje Inductivo y Aprendizaje Basado en Explicaciones. Del tipo no supervisado son el Aprendizaje Reforzado y el Aprendizaje Evolutivo.

2.2.1 Elementos del Aprendizaje Reforzado

Los elementos esenciales de un sistema de AR son:

El agente

Su función es aprender una conducta mediante la interacción con el ambiente para alcanzar una meta. El objetivo que tiene establecido es maximizar el total de recompensas adquiridas del ambiente. Determina la acción a ejecutar de acuerdo a la información concerniente al estado actual, proporcionada por el ambiente.

El ambiente

Es el sistema externo en el que un agente está "insertado", y puede percibir y actuar en el mismo. Cada sistema de Aprendizaje Reforzado aprende una representación de estado-acción por interacciones de prueba y error en un ambiente. Este ambiente debe ser por lo menos parcialmente observable por el sistema. En (Russell y Norving, 1995) se relacionan los posibles ambientes que un agente puede enfrentar (estáticos, dinámicos, semidinámicos, accesibles, parcialmente accesibles, continuos, episódicos, etc.)

Los estados

Son una representación de la condición actual del ambiente en que se ubica el problema. La condición actual del ambiente puede ser modificada por la ejecución de las acciones.

Las acciones

Son el conjunto de alternativas que se pueden tomar. El problema consiste en determinar cuál es la mejor acción que se debe seleccionar para cada uno de los estados posibles.

La política

Es la asociación de los estados que son percibidos del ambiente y las acciones que deben ser ejecutadas. La conducta del agente queda establecida por la política, por lo cual se dice que es el núcleo del agente. La política determina el aprendizaje de un agente de acuerdo a la conducta que va tomando en un espacio de tiempo. En general, la política debe ser estocástica (Sutton y Barto, 1998), y es la conducta actual en el proceso de aprendizaje, pues describe un mapa de situaciones percibidas y acciones a ejecutar (Martin, 1998).

En algunos casos la política puede ser una tabla de consulta o una función simple, y en otros puede implicar el cálculo extenso, como un proceso de búsqueda.

La función de recompensa inmediata

Define el objetivo en un problema de Aprendizaje Reforzado. Esto se realiza mediante la asociación de (estado, acción) con un número escalar nombrado recompensa inmediata. Este valor se expide al agente desde el ambiente y muestra qué tan bueno fue el resultado de la acción ejecutada con respecto a la meta global del agente. La función de recompensa define las acciones que son buenas y malas para un agente; no puede ser cambiada por el agente, pero puede servir, sin embargo, como una base para cambiar la política.

La función de valor

Cada par (estado, acción) es asociado al valor total de la recompensa acumulada que se obtiene al realizar la acción según el estado en que se encuentre. Orienta sobre qué tan bueno ha sido para el agente ejecutar la acción, cada vez que se origina dicho estado.

Mientras que una función de recompensa indica lo que está bien en un sentido inmediato, una función de valor especifica lo que está bien a largo plazo. A grandes rasgos, el valor de un estado es la cantidad de recompensa que un agente puede esperar acumular sobre el futuro que comienza desde ese estado.

Por ejemplo, un estado siempre podría ceder una recompensa inmediata baja, pero podría tener un valor alto porque es con regularidad seguido de otros estados que ceden recompensas altas, o a la inversa podría pasar lo mismo.

Las recompensas son en cierto modo primarias, mientras que los valores, como predicciones de recompensas, son secundarios. Sin recompensas no podría haber valores, y el único objetivo de estimar valores es conseguir más recompensa. Las

recompensas son básicamente dadas directamente por el ambiente, pero los valores deben ser estimados en cada secuencia de observaciones que un agente hace de por vida.

En cada paso de decisión se establece una nueva función de valor para las acciones disponibles. Una función de valor de las acciones permite determinar una política.

El modelo del ambiente

Es de alguna manera algo que imita el comportamiento del ambiente, por ejemplo, dado un estado y una acción el modelo debe predecir el siguiente estado y la siguiente recompensa. Los modelos son usados para la planificación, por la cual se puede decidir el curso de una acción considerando futuras situaciones posibles antes de que ellas realmente sean experimentadas (Sutton y Barto, 1998).

2.2.2 Métodos de selección de acciones

La regla de selección de acción más simple consiste en seleccionar, entre todas las acciones posibles, la acción con el valor de acción estimado más alto, o sea, seleccionar en el paso t la acción a^* tal que $Q_t(a^*) = \max_a Q_t(a)$. Este método, conocido como *greedy*, explota el conocimiento actual del agente para maximizar la recompensa inmediata. De esta manera el agente no pierde tiempo en probar acciones aparentemente inferiores para ver si realmente las puede mejorar.

Una alternativa para este método es comportarse como tal la mayor parte del tiempo y con una baja probabilidad ε explorar, seleccionando una acción aleatoria, independientemente de los valores de acción estimados. A esta probabilidad ε se le conoce como índice de exploración, en los primeros episodios este índice debe ser alto para que se exploren bastantes acciones y va disminuyendo a medida que transcurren los episodios para poder explotar el conocimiento adquirido. Este método se conoce como *ε-greedy*.

La ventaja de los métodos *ε-greedy* sobre los *greedy* depende del proceso en cuestión. Por ejemplo, si se toma una varianza grande en la recompensa, se necesitaría más exploración para encontrar la acción óptima y el primer método resultaría relativamente mejor que el segundo. Por el contrario, si la varianza fuera cero los métodos *greedy* conocerían el valor verdadero de cada acción luego de examinarla una vez y tendrían un mejor desempeño, pues encontrarían más rápido la acción óptima, no exploran. A pesar de esto, incluso en el caso determinista,

existe una amplia ventaja en la exploración. El Aprendizaje Reforzado requiere un balance entre exploración y explotación.

Selección de la acción softmax

Aunque la selección de la acción por el método *ε-greedy* es una efectiva manera de balancear la exploración y la explotación, presenta el inconveniente de elegir por igual entre todas las acciones cuando explora. Esto significa que existe la posibilidad de escoger la acción que aparentemente produzca una menor recompensa. En problemas donde la acción aparentemente peor es realmente la de menor valor, el método podría alcanzar resultados insatisfactorios. Una solución es variar la probabilidad de las acciones como una función gradual del valor estimado. La acción de mayor valor sigue teniendo la mayor probabilidad de selección, pero las restantes son ponderadas y ordenadas de acuerdo a sus valores estimados. A estas variantes se le conoce reglas softmax de selección de acciones.

Los métodos softmax más comunes usan una distribución de Gibbs o Boltzmann. Seleccionan una acción a en el paso t con probabilidad:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

donde τ es un parámetro positivo llamado temperatura. Altas temperaturas provocan que las acciones sean casi equiprobables. Bajas temperaturas causan diferencias más notables en la probabilidad de selección de las acciones que difieren en sus valores estimados.

No está claro cuál de los dos métodos es mejor, depende de la tarea y del programador. Ambos métodos tienen un solo parámetro que debe ser fijado. Muchos usuarios prefieren fijar el parámetro ε, puesto que fijar el parámetro τ requiere algún conocimiento de los valores probables de las acciones.

2.2.3 El problema del Aprendizaje Reforzado

En este epígrafe se describe el problema que define el dominio del Aprendizaje Reforzado; cualquier método que lo resuelva se considera un método de Aprendizaje Reforzado. Además, se introducen los elementos claves de la estructura matemática que fundamenta el problema.

2.2.3.1 Interfaz agente - ambiente

El agente y el ambiente interactúan en cada uno de los intervalos discretos de tiempo $t = 1,2,3...$. En cada paso de tiempo el agente recibe cierta representación referente al estado del ambiente $s_t \in S$, donde S es el conjunto de estados posibles, y sobre esa base selecciona una acción $a_t \in A(s_t)$, donde $A(s_t)$ es el conjunto de acciones disponibles en el estado s_t. Un intervalo de tiempo después, en parte como una consecuencia de su acción, el agente recibe una recompensa numérica $r_{t+1} \in \Re$, y se encuentra en un nuevo estado, s_{t+1}. En la figura 2.2 se muestra un diagrama de interacción entre el agente y el ambiente.

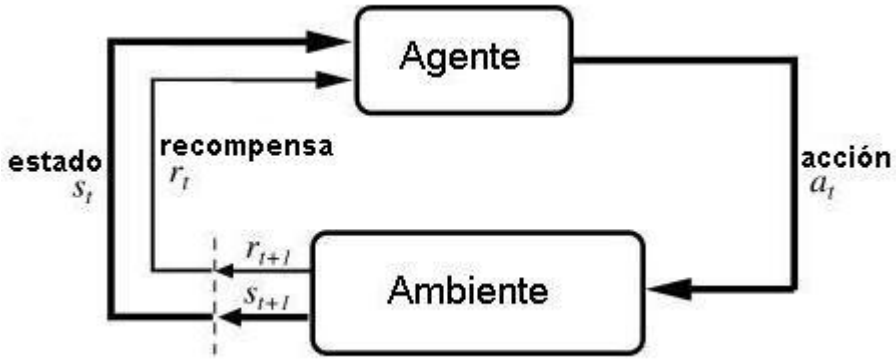


Figura 2.2. Interacción entre el agente y el ambiente. (Sutton y Barto, 1998).

En cada intervalo de tiempo el agente pone en práctica una correlación de estados a probabilidades de seleccionar cada acción posible. Esta correlación es llamada la política del agente y es denotada por π_t, donde $\pi_t(s,a)$ es la probabilidad de que $a_t = a$ si $s_t = s$. Los métodos de Aprendizaje Reforzado especifican como el agente cambia su política a consecuencia de su experiencia.

Para una gran cantidad de problemas es posible probar, bajo condiciones no muy fuertes que existe una política óptima estacionaria y determinista, de hecho basta que S y A sean finitos; en tales casos la política del agente no depende del tiempo y $\pi(s,a)$ denota simplemente la acción a correspondiente al estado s.

El objetivo del agente es maximizar la cantidad total de recompensa obtenida en un horizonte de tiempo, que puede ser finito o infinito (Sutton y Barto, 1998).

El ambiente por lo general es no determinista, ya que elegir la misma acción en el mismo estado en dos ocasiones diferentes puede resultar en diferentes estados siguientes y/o diferentes valores de refuerzo (Kaelbling et al., 1996).

2.2.3.2 La propiedad de Markov

En el marco del AR el agente toma sus decisiones basándose en una señal del ambiente llamada estado del ambiente. En este epígrafe se define formalmente una propiedad del ambiente y sus señales de estados llamada propiedad de Markov.

La propiedad de Markov establece que las transiciones de los estados y los valores de recompensa dependen solamente del estado actual y de la acción actual, no dependen de estados, acciones o recompensas previas (Cichoz, 1995).

Ciertamente la señal de estado debería incluir sensaciones inmediatas, pero puede contener mucho más que eso. Las representaciones del estado pueden ser versiones altamente procesadas de sensaciones originales, o pueden ser complejas estructuras construidas a través del tiempo desde la sucesión de las sensaciones.

Lo que se quiere, idealmente, es una señal de estado que resuma sólidamente las sensaciones pasadas, de forma tal que toda la información relevante se retenga. Esto normalmente requiere más sensaciones inmediatas, pero nunca la historia completa de todas las sensaciones pasadas. Una señal de estado que tenga éxito en retener toda la información relevante se dice ser de Markov, o que tiene la propiedad de Markov.

Ahora se define formalmente la propiedad de Markov para el problema de Aprendizaje Reforzado. Para mantener las matemáticas simples se asume que hay un número finito de estados y valores de recompensa. Estos permiten trabajar en términos de sumas y probabilidades en lugar de integrales y densidades de probabilidad, el argumento puede ser fácilmente ampliado para incluir estados continuos y recompensas. Considérese cómo un ambiente podría responder en el tiempo $t+1$ a la acción tomada en el tiempo t. En el caso más general, esta respuesta puede depender de todo lo que ha sucedido anteriormente y la dinámica se define por la especificación de la distribución de probabilidad completa:

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\} \qquad 2.1$$

para todo s', r y todos los valores posibles de los acontecimientos pasados: $s_t, a_t, r_t, ..., r_1, s_0, a_0$. Si la señal de estado tiene la propiedad de Markov, entonces la respuesta del ambiente en $t+1$ depende solo de las representaciones del estado y

de la acción en t, en tal caso la dinámica del ambiente puede ser definida especificando solamente:

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\} \qquad 2.2$$

para todos s', r, s_t y a_t. En otras palabras, una señal de estado tiene la propiedad de Markov, y es un estado de Markov, si y solo si 2.2 es igual a 2.1 para todo s', r, y sucesos $s_t, a_t, r_t, ..., r_1, s_0, a_0$. En este caso se dice que el ambiente y el proceso de manera general, tienen la propiedad de Markov.

Si un ambiente tiene la propiedad de Markov entonces su dinámica de un paso facilita, dados el estado y la acción actual, la predicción del próximo estado y la siguiente recompensa esperada.

Se puede demostrar que iterando esta ecuación se puede predecir todos los futuros estados y recompensas esperadas a partir del conocimiento del estado actual, tal como sería posible dada la historia completa hasta el paso actual. De esta manera los estados de Markov proporcionan la posible mejor base para elegir las acciones.

Incluso, cuando la señal de estado no es de Markov, es todavía apropiado pensar en el estado del Aprendizaje Reforzado como una aproximación a un estado de Markov. En particular, siempre se quiere que el estado sea una buena base para predecir futuras recompensas y para la selección de acciones.

Las representaciones del estado de Markov no se aplican solamente dentro del Aprendizaje Reforzado, están presentes también en la mayoría de las aplicaciones de la Inteligencia Artificial (Sutton y Barto, 1998).

2.2.3.3 Procesos markovianos de decisión

Un proceso del Aprendizaje Reforzado que satisface la propiedad de Markov es llamado proceso de decisión de Markov, *Markov Decision Process* (MDP) en inglés. Si los espacios de estado y acción son finitos, entonces es llamado proceso de decisión de Markov finito (MDP finito). Los MDPs finitos son particularmente importantes para la teoría del Aprendizaje Reforzado, pues constituyen lo necesario para entender el 90 % del Aprendizaje Reforzado moderno.

Un MDP finito es definido por un conjunto de estados, un conjunto de acciones y por la dinámica de un paso de tiempo del ambiente. Dados cualquier estado y acción, s y a, la probabilidad de cada posible siguiente estado, s', es:

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

que son llamadas probabilidades de transición. Similarmente, dados cualquier estado y acción actuales s y a, unidos a cualquier siguiente estado s', el valor esperado de la próxima recompensa es:

$$R_{ss'}^{a} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$$

Las cantidades $P_{ss'}^{a}$ y $R_{ss'}^{a}$ detallan completamente los aspectos más importantes de la dinámica de un MDP finito, solo se pierde la información relacionada con la distribución de las recompensas alrededor del valor esperado (Sutton y Barto, 1998).

El modelo formal de MDP quedaría compuesto por (Morales, 2004):

- Un conjunto de estados S.
- Un conjunto de acciones A.
- Probabilidades de transición:

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

- Recompensa:

$$R_{ss'}^{a} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$$

2.2.3.4 Función de valor

Casi todos los algoritmos de Aprendizaje Reforzado están basados en la estimación de funciones de valor, funciones de estados, o pares de estado-acción, que estiman qué tan bueno es para el agente estar en el estado dado o qué tan bueno es el desempeño al ejecutar una acción determinada en dicho estado.

Es importante recordar que una política π es un mapeo de cada estado, $s \in S$, y acción, $a \in A(s)$, a la probabilidad $\pi(\mathrm{s},\mathrm{a})$ de tomar la acción a en el estado s. Informalmente, el valor de un estado s bajo la política π, denotado por $V^{\pi}(s)$, es la recompensa esperada cuando se parte de s y se sigue la política π. Para los MDP, se define $V^{\pi}(s)$ como:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\} \qquad 2.3$$

donde $E_{\pi}\{\}$ representa el valor esperado dado que el agente sigue la política π; $0 \le \gamma \le 1$ es una constante conocida como factor de descuento, que determina el

valor relativo de las recompensas inmediatas y retardadas; y t es cualquier paso de tiempo.

A la función V^{π} se le denomina función estado-valor para la política π. Similarmente se define el valor de tomar la acción a en el estado s bajo una política π, denotado por $Q^{\pi}(s,a)$, como la ganancia a partir de s, tomando la acción a, y siguiendo la política π:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\} \qquad 2.4$$

Las funciones de valor V^{π} y Q^{π} pueden ser estimados de la experiencia.

Una propiedad fundamental de las funciones de valor que se usa a lo largo de los métodos de AR y Programación Dinámica, es que satisfacen una particular relación recursiva. Para cualquier política π y cualquier estado s, se obtiene la siguiente condición de consistencia entre el valor de s y el valor de sus posibles estados sucesores:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t \mid s_t = s\} \\ &= E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\} \\ &= E_{\pi}\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s\} \qquad 2.5 \\ &= \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_{t+1} = s'\}] \\ &= \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')] \end{aligned}$$

Está implícito que la acción a se toma del conjunto $A(s)$, y los estados siguientes s' se toman del conjunto S ó S^+ (es el conjunto S más un estado terminal si el problema es episódico). La ecuación anterior es conocida como la ecuación de Bellman para V^{π} y expresa la relación entre el valor de un estado y el valor de los estados sucesores (Sutton y Barto, 1998).

2.2.4 Métodos de solución al problema de Aprendizaje Reforzado

Existen tres formas principales de resolver los problemas de Aprendizaje Reforzado. Cada uno de ellos tiene sus fortalezas y debilidades y aunque pueden emplearse para resolver el problema completo, se diferencian en diversas maneras

respecto a su eficiencia y velocidad de convergencia, en los requisitos que exigen para la resolución y en la manera en que se lleva a cabo dicha resolución. A continuación se mencionan:

- Los métodos basados en Programación Dinámica: Estos métodos tienen un desarrollo matemático bien conocido, pero requieren de un modelo del entorno completo y ajustado a la realidad.
- Los métodos de Monte Carlo: No requieren tal modelo y son simples conceptualmente, pero no se pueden emplear para obtener soluciones iterativas paso a paso.
- Los métodos de Diferencias Temporales son en realidad una combinación de ideas extraídas de los métodos de Monte Carlo y de la Programación Dinámica. Estos métodos tampoco necesitan un modelo y pueden calcularse de manera incremental, aunque su análisis es más complejo.

En los siguientes epígrafes se profundizará en estos tres métodos y en cómo pueden combinarse para obtener las ventajas de cada uno de ellos.

2.2.4.1 Métodos de Programación Dinámica

El término de Programación Dinámica (PD) se refiere a una colección de algoritmos que pueden ser usados para calcular las políticas óptimas dado un modelo perfecto del ambiente como un proceso de decisión de Markov. Los algoritmos clásicos de PD son limitados en utilidad dentro del Aprendizaje Reforzado, pues requieren un completo modelo del ambiente y un alto costo computacional.

A pesar de sus limitaciones tienen una gran importancia teórica, ya que suministran las bases esenciales para entender los restantes métodos, incluso muchos de ellos intentan alcanzar sus mismos resultados con menor costo computacional y sin la asunción de un modelo perfecto del ambiente.

Para un mejor análisis de este método se asume el ambiente como un proceso de decisión de Markov finito. Los conjuntos de estados y acciones S y $A(s)$, para $s \in S$, son finitos y su dinámica viene dada por el conjunto de probabilidades de transición $P_{ss'}^{a} = \Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$ y recompensas esperadas $R_{ss'}^{a} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$, para todo $s \in S$, $a \in A(s)$ y $s' \in S^{+}$. Aunque las ideas de la programación dinámica pueden ser aplicadas a problemas con

espacios de estado y acción continuos, las soluciones exactas solo son posibles en casos especiales.

La idea principal de la Programación Dinámica y del Aprendizaje Reforzado en general, es el uso de las funciones de valor para planificar y estructurar la búsqueda de buenas políticas. Seguidamente se demuestra cómo los métodos de PD pueden ser usados para calcular las funciones de valor definidas en los epígrafes anteriores. Se pueden obtener las políticas óptimas a partir de las funciones de valor óptimas V^* ó Q^*, las cuales satisfacen las ecuaciones de optimalidad de Bellman:

$$V^*(s) = \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\}$$
$$= \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')]$$

ó

$$Q^*(s,a) = E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\}$$
$$= \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \max_{a'} Q^*(s', a')]$$

Si se conoce el modelo del ambiente, es decir, las transiciones de probabilidad $P^a_{ss'}$ y los valores esperados de recompensas $R^a_{ss'}$, las ecuaciones de optimalidad de Bellman nos representan un sistema de $|S|$ ecuaciones y $|S|$ incógnitas.

Primeramente se calcula la función de valor V^π dada una política arbitraria π:

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\}$$
$$= E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \mid s_t = s\}$$
$$= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\}$$
$$= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]$$

donde $\pi(s,a)$ es la probabilidad de tomar la acción a en el estado s bajo la política π.

Se pueden hacer aproximaciones sucesivas evaluando $V_{k+1}(s)$ en términos de $V_k(s)$:

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]$$

En la figura 2.3 se define un algoritmo iterativo de evaluación de políticas.

Entrada: La política π a ser evaluada.

Inicializar $V(s) = 0$ **para todo** $s \in S^+$

Repetir

$\Delta \leftarrow 0$

Para cada $s \in S$

$v \leftarrow V(s)$

$$V(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$$

$\Delta \leftarrow \max(\Delta, | v - V(s) |)$

hasta que $\Delta < \theta$ **(número pequeño positivo)**

retornar $V \approx V^*$

Figura 2.3. Algoritmo iterativo de evaluación de la política. (Sutton y Barto, 1998)

Una de las razones para calcular la función de valor de una política es tratar de encontrar mejores políticas. Dada una función de valor para una política dada se puede probar una acción $a \neq \pi(s)$ y ver si su $V(s)$ es mejor o peor que el $V^{\pi}(s)$.

En lugar de hacer un cambio en un estado y ver el resultado, se pueden considerar cambios en todos los estados considerando todas las acciones de cada estado, seleccionando la mejor.

Luego se calcula una nueva política $\pi'(s) = \text{argmax}_a Q^{\pi}(s,a)$ y se continúa hasta que no se mejore.

Esto sugiere, partir de una política π_0 y calcular la función de valor V^{π_0}, con la cual encontrar una mejor política π_1 y así sucesivamente hasta converger a π^* y V^*. A este procedimiento se le llama iteración de políticas y viene descrito en la figura 2.4.

Uno de los problemas de la iteración de políticas es que cada iteración involucra una evaluación de políticas que requiere recorrer todos los estados varias veces. Sin embargo, el paso de evaluación de política se puede truncar de varias formas, sin perder la garantía de convergencia. Una de ellas es pararla después de recorrer una sola vez todos los estados. A esta forma se le llama iteración de valor. En particular se puede escribir combinando la mejora en la política y la evaluación de la política truncada como sigue:

$$V_{k+1} = \max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V_k(s')]$$

1. Inicializar:
$V(s) \in \Re$ **y** $\pi(s) \in A(s)$ **arbitrariamente** $\forall s \in S$

2. Evaluación de la política:
Repite
$\Delta \leftarrow 0$
Para cada $s \in S$
$v \leftarrow V(s)$
$V(s) = \sum_{s'} P_{ss'}^{\pi(s)} [R_{ss'}^{\pi(s)} + \gamma V(s')]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
hasta que $\Delta < \theta$ **(número pequeño positivo)**

3. Mejora de la política:
$pol\_estable \leftarrow true$
Para cada $s \in S$ **:**
$b \leftarrow \pi(s)$
$\pi(s) \leftarrow \arg\max_a \sum_{s'} P_{ss'}^{a} [R_{ss'}^{a} + \gamma V(s')]$
si $b \neq \pi(s)$ **, entonces** $pol\_estable \leftarrow false$
si $pol\_estable$ **, entonces** $stop$ **, sino ir a 2.**

Figura 2.4. Algoritmo de iteración de la política. (Sutton y Barto, 1998).

Se puede ver como expresar la ecuación de Bellman en una regla de actualización. Es muy parecido a la regla de evaluación de políticas, solo que se evalúa el máximo sobre todas las acciones, figura 2.5.

Inicializar $V(s) = 0$ **para todo** $s \in S^+$
Repetir
$\Delta \leftarrow 0$
Para cada $s \in S$
$v \leftarrow V(s)$
$V(s) = \max_a \sum_{s'} P_{ss'}^{a} [R_{ss'}^{a} + \gamma V(s')]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
hasta que $\Delta < \theta$ **(número pequeño positivo)**

retornar una política determinista π **tal que:**
$\pi(s) = \arg\max_a \sum_{s'} P_{ss'}^{a} [R_{ss'}^{a} + \gamma V(s')]$

Figura 2.5 Algoritmo de iteración del valor. (Sutton y Barto, 1998).

Para espacios muy grandes, el hecho de ver todos los estados puede ser costoso computacionalmente. Una opción es hacer estas actualizaciones al momento de

estar explorando el espacio, y por lo tanto determinando sobre qué estados se hacen las actualizaciones.

Estos métodos tienen la propiedad de actualizar los estimados de los valores de estados basándose en los estimados de los valores de estados sucesivos. A esta propiedad se le conoce como *bootstrapping.* Muchos métodos de AR usan esta propiedad, aun cuando no requieren un modelo exacto y completo del ambiente.

Aunque los métodos de Programación Dinámica no son prácticos para problemas muy largos, comparados con otros métodos de solución de MDPs son realmente eficientes. Si se ignoran algunos detalles técnicos, el tiempo que toman estos métodos para encontrar una política óptima, en el peor de los casos, es polinomial respecto al número de estados y acciones. Un método de PD garantiza encontrar una política óptima en un tiempo polinomial, aun cuando el total de políticas deterministas es m^n, donde n y m representan la cantidad de estados y de acciones respectivamente. En tales casos este método es exponencialmente más rápido que cualquier búsqueda directa en el espacio de políticas, pues la búsqueda directa tendría que examinar exhaustivamente cada política para alcanzar los mismos resultados.

En la práctica, los métodos de PD pueden ser usados para resolver MDPs con millones de estados. La iteración de la política y la iteración del valor son ampliamente usadas. Estos métodos convergen frecuentemente más rápido que como señalan sus tiempos de ejecución teóricos en los peores casos, particularmente si se inicializan con buenas funciones de valor o políticas (Sutton y Barto, 1998).

2.2.4.2 Métodos de Monte Carlo

Los métodos de Monte Carlo (MC), a diferencia de los métodos de Programación Dinámica, no requieren un conocimiento completo del ambiente. Estos métodos aprenden o determinan las funciones de valor con base en una serie de experiencias denominadas episodios. Cada episodio consiste de un cierto número de pasos de decisión a partir de una función de valor de las acciones y bajo una determinada política.

La idea básica en que se apoya este método es que el valor de las acciones, definido como el valor esperado de la recompensa acumulada para las acciones

aplicadas en un estado determinado, puede ser calculado directamente como el promedio de las recompensas inmediatas, figura 2.6.

Repetir
  Generar un episodio usando π
  Para cada estado s **en ese episodio**
    $R \leftarrow$ recompensa después de la primera ocurrencia de s
    Añadir R **a** $recomp(s)$
    $V(s) \leftarrow \text{promedio}(recomp(s))$

Figura 2.6 Algoritmo de Monte Carlo para estimar V^{π}. (Sutton y Barto, 1998)

Para estimar el valor de los pares estado-acción Q^{π}, se corre el peligro de no ver todos los pares, por lo que se busca mantener la exploración. Lo que normalmente se hace es considerar solamente políticas estocásticas que tienen una probabilidad diferente de cero para seleccionar todas las acciones.

Con Monte Carlo se puede alternar entre evaluación y mejoras en base a cada episodio. Una vez terminado el episodio se utiliza la secuencia de estados, acciones y recompensas observadas para estimar la siguiente función de valor y evaluar la política, la cual se actualiza para todos los estados visitados en el episodio, figura 2.7.

Repetir
  Generar un episodio usando π **con exploración**
  Para cada par (s,a) **en ese episodio**
    $R \leftarrow$ recompensa después de la primera ocurrencia de (s,a)
    Añadir R **a** $recomp(s,a)$
    $Q(s,a) \leftarrow \text{promedio}(recomp(s,a))$
  Para cada s **en el episodio**
    $\pi(s) \leftarrow \text{argmax}_a Q(s,a)$

Figura 2.7 Algoritmo de Monte Carlo. (Sutton y Barto, 1998)

Estos métodos se basan en la experiencia que se tiene después de la ejecución de un conjunto de acciones. Aprenden la función de valor a partir de su interacción con el ambiente. Por tanto, no requieren una especificación de la dinámica del ambiente. Otra diferencia importante con respecto a los métodos de PD, es que el cálculo del valor de las acciones para un estado determinado no depende de ningún otro estado.

Las propiedades de convergencia de los métodos de Monte Carlo para el Aprendizaje Reforzado no están del todo claras y su efectividad en la práctica se ha experimentado bastante poco. La importancia principal es su simplicidad y su estrecha relación con otros métodos (Sutton y Barto, 1998).

2.2.4.3 Métodos de Diferencias Temporales

Un elemento central y novedoso dentro del Aprendizaje Reforzado lo constituye sin dudas el aprendizaje por Diferencia Temporal (DT). Este método es una combinación de las ideas de Monte Carlo y Programación Dinámica. Al igual que en los métodos de Monte Carlo, los métodos de DT pueden aprender directamente de la experiencia natural, sin un modelo de la dinámica del ambiente; y como en los métodos de Programación Dinámica actualizan los estimados basándose en parte en otros estimados adquiridos, sin esperar por el resultado final (*bootstrapping*).

Los métodos de Diferencias Temporales, al igual que los de Monte Carlo, usan la experiencia para resolver el problema de estimar los valores de las funciones de valor generadas. Dada alguna experiencia, siguiendo una política π, ambos métodos actualizan su estimación de V^{π}. Si un estado no terminal s_t se visita en el tiempo t, ambos métodos actualizan su estimación $V(s_t)$ con base en lo que sucede después de la visita. Los métodos de MC esperan hasta que termine el episodio para determinar la recompensa asociada al valor de las acciones ejecutadas en cada visita a s_t. Una vez determinada esta recompensa, se utiliza para estimar $V(s_t)$. Por otro lado, los métodos de DT necesitan esperar solo hasta el próximo paso. En el tiempo $t+1$ se hace una actualización usando la recompensa observada r_{t+1} y la estimación de $V(s_{t+1})$. El método más simple de DT, conocido como DT (0), es:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

donde $0 \leq \alpha \leq 1$ es la velocidad de aprendizaje, mientras más alto sea el valor más rápido ocurre el aprendizaje.

Al igual que los métodos de PD, los métodos de DT se consideran como métodos *bootstrapping*, pues basan su actualización en una estimación ya existente. De los resultados en epígrafes anteriores se tiene:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} \qquad 2.6$$

$$= E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\}$$

$$= E_{\pi}\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s\} \qquad 2.7$$

$$= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\}$$

Los métodos de Monte Carlo usan un estimado de 2.6 como un objetivo, mientras que los métodos de Programación Dinámica usan un estimado de 2.7. El objetivo de MC es un estimado porque el valor esperado en 2.6 no se conoce, usa una simulación de la recompensa en lugar de la verdadera recompensa esperada. El objetivo de la PD es un estimado, no debido a los valores esperados que son suministrados completamente por un modelo del ambiente, sino porque $V^{\pi}(s_{t+1})$ no se conoce y en su lugar se usa $V_t(s_{t+1})$. El objetivo de DT es un estimado por dos razones: muestrea los valores esperados en 2.7 y usa el estimado actual V_t en lugar del verdadero V^{π}. De esta manera los métodos de DT combinan las simulaciones de MC con el *bootstrapping* de los métodos de PD. El algoritmo procedural para DT (0) se muestra en la figura 2.8.

Inicializar $V(s)$ **arbitrariamente,** π **la política a evaluar**
Repetir (para cada episodio):
  Inicializar s
  Repetir (para cada paso del episodio):
    $a \leftarrow$ **acción dada por** π **para** s
    Realizar la acción a; observar la recompensa r**, y el siguiente estado** s'
    $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
    $s \leftarrow s'$
  hasta que s **sea terminal.**

Figura 2.8 Algoritmo DT(0). (Sutton, 1998)

Actualmente los métodos de Diferencias Temporales son los más usados dentro del Aprendizaje Reforzado. Esto se debe a su gran simplicidad y a la ventaja de que pueden ser empleados de manera incremental, permiten que el agente aprenda *on-line* con un mínimo costo computacional y además convergen más rápido con mejores predicciones.

Estos métodos dan lugar a algoritmos iterativos muy sencillos, aspecto que favorece su implementación. Dos de los algoritmos más utilizados son SARSA y Q-Learning (Sutton y Barto, 1998).

2.2.5 Algoritmos de solución

En la sección anterior se analizaron los tres métodos fundamentales para resolver el problema del Aprendizaje Reforzado, por lo que en este apartado se profundizará en los algoritmos que derivan de la predicción de Diferencias Temporales.

2.2.5.1 Algoritmo SARSA

El algoritmo SARSA usa los métodos de predicción de Diferencias Temporales para el problema de control. Al igual que en los métodos de Monte Carlo, se enfrenta a la necesidad de emplear exploración y explotación.

Este algoritmo es *on-policy* pues estima el valor de la política mientras la usa para el control. Se trata de mejorar la política que se usa para tomar decisiones.

El primer paso es aprender una función de acción-valor en lugar de una función estado-valor. En particular, para un método *on-policy* se necesita estimar $Q^{\pi}(s,a)$ para la política de comportamiento actual π, y para todos los estados y acciones, s y a respectivamente. Esto se puede hacer usando esencialmente lo mismo que se usa en los métodos de Diferencias Temporales para obtener V^{π}.

Un episodio consiste de una secuencia alterna de estados y pares estado-acción. En las secciones anteriores se consideraron las transiciones de estado a estado obteniendo los valores de estado. Ahora se consideran las transiciones de un par estado-acción a otro par estado-acción obteniéndose el valor de los pares. Formalmente estos casos son idénticos: ambos son cadenas de Markov con un procedimiento de recompensa.

Los teoremas que aseguran la convergencia de los valores de estado en el algoritmo DT(0), también se aplican a este algoritmo para los valores de acciones:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Esta actualización se hace después de cada transición desde un estado no terminal s_t. Si s_{t+1} es terminal entonces $Q(s_{t+1}, a_{t+1})$ se define como cero. Esta regla usa cada elemento del quíntuplo de eventos $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$, completando una transición de un par estado-acción al siguiente. Este quíntuplo da origen al

nombre SARSA para el algoritmo. La figura 2.9 muestra la forma general de este algoritmo.

Inicializar $Q(s,a)$ **arbitrariamente**
Repetir (para cada episodio)
  Inicializar s
  Escoger a **de** s **usando una política derivada de** Q **(ej: *ε-greedy*)**
  Repetir (para cada caso del episodio)
    Tomar una acción a **, observar** r **,** s'
    Escoger a' **de** s' **usando una política derivada de** Q **(ej: *ε-greedy*)**
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'; a \leftarrow a';$
  hasta que s **sea terminal.**

Figura 2.9 Forma general del algoritmo SARSA. (Sutton y Barto, 1998)

Las propiedades de convergencia del algoritmo SARSA dependen de la naturaleza de dependencia de la política sobre Q. SARSA converge con probabilidad 1 a una política óptima y a una función óptima de acción-valor siempre y cuando todos los pares de estado-acción se visiten un número infinito de veces.

2.2.5.2 Algoritmo Q-Learning

Uno de los avances más importantes en el Aprendizaje Reforzado fue el desarrollo de un algoritmo *off-policy* de control de Diferencias Temporales conocido como Q-Learning (Watkins, 1989).

Los algoritmos *off-policy* usan la política y el control de forma separada. La estimación de la política puede ser por ejemplo *greedy* y la política de comportamiento puede ser *ε-greedy*. O sea que la política de comportamiento está separada de la política que se quiere mejorar (Morales, 2004).

Los métodos clásicos de Q-Learning utilizan una matriz como soporte para la representación de los valores de Q. A cada par estado-acción se le asigna un valor de Q o acción de utilidad, que representa un elemento en la matriz cuya entrada es el correspondiente valor aproximado de $Q(s,a)$. Este valor es un estimado de la suma de los futuros valores de refuerzo recibidos a partir del estado s, tomando la acción a y siguiendo una política *greedy* con respecto a la función Q. (Cichoz, 1995).

En su forma más simple un paso del algoritmo se define por:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \qquad 2.8$$

Esta regla actualiza cada par estado-acción maximizando todas las posibles acciones en el estado siguiente. El algoritmo se muestra de forma procedural en la figura 2.10.

Inicializar $Q(s,a)$ **arbitrariamente**
Repetir (para cada episodio)
  Inicializar s
  Repetir (para cada caso del episodio)
    Escoger a **de** s **usando una política derivada de** Q **(ej: *ε-greedy*)**
    Tomar una acción a**, observar** r**,** s'
    Escoger a' **de** s' **usando una política derivada de** Q **(ej: *ε-greedy*)**
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s';$
  hasta que s **sea terminal.**

Figura 2.10 Algoritmo Q-Learning. (Sutton y Barto, 1998)

En este caso la función de acción-valor observada, Q, se aproxima directamente a Q^*, la función de acción-valor óptima es independiente de la política seguida. La política mantiene un efecto en la determinación de qué par estado-acción será visitado y actualizado. Sin embargo, todo lo que se requiere es que cada par continúe siendo actualizado. Este es un requerimiento mínimo para cualquier método interesado en encontrar un comportamiento óptimo en el caso general (Sutton y Barto, 1998).

Kaelbling plantea que cuando los valores de Q se encuentren cerca de converger a lo valores óptimos es apropiado para el agente adoptar una política *greedy*, tomando en cada situación la acción con el valor más alto de Q. Sin embargo, mientras dura el aprendizaje, es necesario enfrentarse a la difícil disyuntiva exploración versus explotación, para la cual no existe una aproximación lo suficientemente formal en el caso general. De los métodos Q-Learning se dice que son "insensibles a la exploración", es decir, que los valores de Q convergen a los valores óptimos, independientemente de cómo se comporte el agente mientras se recolectan los datos (con tal de que todos los pares estado-acción sean examinados lo suficiente), esto significa que pesar de que el asunto exploración-

explotación debe ser dirigido en el método, los detalles de la exploración no afectan la convergencia del algoritmo de aprendizaje. El hecho de que este método pueda ser utilizado para cualquier MDP del que no se disponga de un modelo exacto, lo convierte en uno de los algoritmos libres del modelo más populares para el Aprendizaje Reforzado; aunque hay que señalar que en ocasiones puede converger muy lentamente.

CAPITULO III. MODULO DE ASIGNACION

En este capítulo se describe la implementación de ambas técnicas de Inteligencia Artificial para dar solución óptima o cercana a la óptima al modelo matemático Özsu, con un costo computacional relativamente bueno y en un espacio de tiempo aceptable.

Se muestran las consideraciones y variaciones impuestas al modelo para una factible implementación y las pruebas realizadas al software con los resultados obtenidos.

3.1 Consideraciones realizadas al modelo de asignación

El modelo descrito en el Capítulo I sobre la asignación de datos en la red es muy general, para poder lograr una implementación computacional del mismo hay que hacer un conjunto de consideraciones que se describirán a continuación.

3.1.1 Condición de réplica

El modelo de la forma que está planteado siempre tiene como solución óptima la matriz nula, o sea la no asignación de fragmentos en la red, lo que no tiene sentido desde el punto de vista práctico. Teniendo esto en cuenta se agrega una restricción de réplica la cual establece que cada fragmento es ubicado en al menos un sitio. La condición de réplica queda modelada de la siguiente manera:

$$\sum_{\forall F_j \in F} x_{jk} \geq 1, \quad \forall s_k \in S$$

donde j y k son los índices de los fragmentos y los sitios respectivamente.

3.1.2 Tratamiento de restricciones

Con el objetivo de lograr un mayor desempeño y por ser el espacio libre para el almacenamiento una característica muy variable de los sitios, se asume que habrá espacio suficiente para almacenamiento de la base de datos, por tanto no se tiene en cuenta la restricción de almacenamiento. En cuanto a la restricción de tiempo de respuesta, también es desechada por no contar con suficiente información en un diseño partiendo de cero, esta será tenida en cuenta en futuras implementaciones de un modelo de reubicación, teniendo en cuenta parámetros de desempeño de las aplicaciones.

3.1.3 Informaciones adicionales

El asistente elaborado para dar solución al problema de la asignación será insertado en una herramienta de ayuda al diseño de bases de datos distribuidas. Esta herramienta debe mantener una gran cantidad de información detallada, descriptores o metadatos referentes a los datos que conforman cada base de datos objeto de diseño; sobre las aplicaciones que usan los datos para dar solución a los casos de uso o solicitudes de usuarios; sobre los sitios que intervendrán en el almacenamiento de los datos y el procesamiento de las solicitudes; y sobre la red que soporta la conexión entre sitios y el intercambio entre ellos a fin de efectuar sus funciones. Toda esta variedad de información es almacenada en el catálogo del sistema, que no es más que una base de datos relacional, autodescriptiva por naturaleza, que mantiene la herramienta para sus propios fines internos.

En el trabajo con el catálogo se usan métodos y lógica de acceso propios para almacenar y recuperar rápida y eficientemente la información que se necesita en la realización de las tareas para las cuales han sido concebidas las herramientas. Las tablas que componen dicho catálogo solo son usadas internamente por las herramientas, compartiendo por esta vía toda la información relevante para el proceso de diseño, de esta forma no son de interés para los diseñadores como usuarios finales del sistema.

Para el correcto funcionamiento del asistente se asume que toda la información necesaria para dar solución al modelo matemático ha sido captada en etapas anteriores del diseño y debidamente almacenada en el catálogo, el cual debe ubicarse en el mismo directorio del programa. El anexo 1 muestra una descripción más detallada del catálogo de la herramienta.

3.2 Aspectos generales del diseño y la implementación del asistente

El asistente se programó en el lenguaje de programación C sharp, que está incluido en el paquete Microsoft Visual Studio 2005. Este lenguaje de programación está diseñado para construir una amplia variedad de aplicaciones que se ejecutan en el entorno .NET.

3.2.1 Diagrama de clases

La figura 3.1 muestra el diagrama estático de clases del módulo desarrollado en este trabajo. En el anexo 2 se muestra con más detalle el diseño de cada una de las clases.

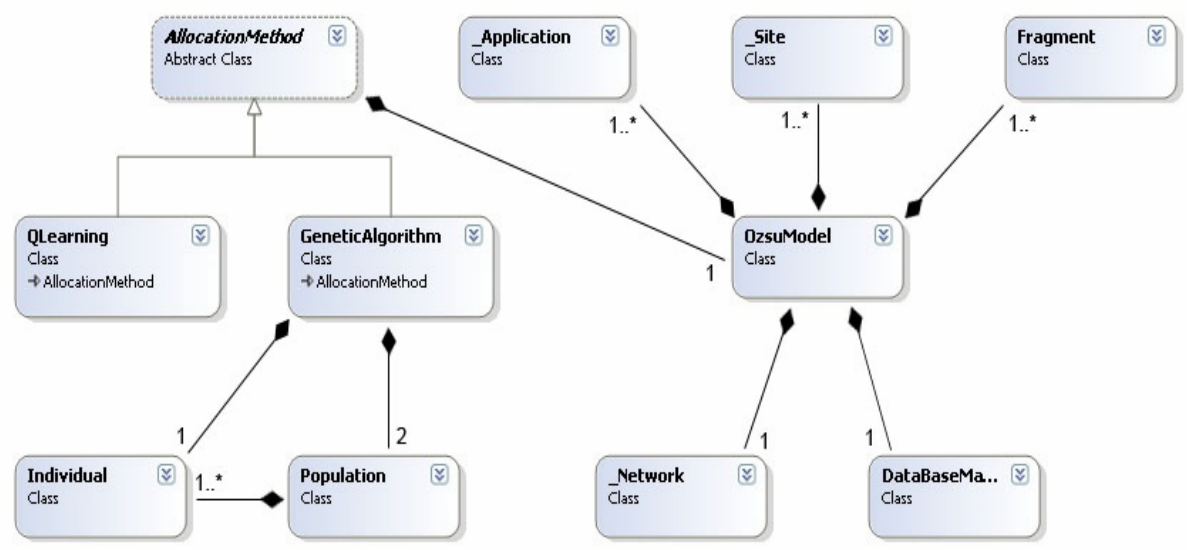


Figura 3.1 Diagrama de clases

3.2.2 Definición de las clases

Las principales clases de este proyecto son: *GeneticAlgorithm*, *QLearning* y *OzsuModel*. Las dos primeras imitan el comportamiento de los Algoritmos Genéticos y el algoritmo Q-Learning respectivamente y son las encargadas de todo el proceso de optimización del modelo Özsu, representado por la tercera.

Las demás clases sirven de complemento a las primeras, cumpliendo determinadas funciones dentro del proyecto. A continuación se describen las más importantes.

La clase DataBaseManagement

Esta clase es la encargada de interactuar con la base de datos que representa al catálogo. A través de los métodos públicos *LoadData* y *SaveData*, lee toda la información que necesitan los algoritmos para resolver el modelo del problema y salva el resultado de la asignación en el catálogo de la aplicación. Para ello se vale de una variedad de clases que usan la interfaz de programación OLE DB y que pertenecen al namespace System.Data.OleDb. Esta interfaz llevó a la práctica los conceptos teóricos de la estrategia Universal de Acceso a Datos (UDA) de Microsoft, permitiendo el acceso a cualquier tipo datos tanto relacional como no relacional.

La clase OzsuModel

Esta clase representa al modelo matemático Özsu que se quiere optimizar. Cuenta con arreglos de instancias de las clases *Site*, *Fragment* y *Application* y con una

instancia de la clase *Network*. Cada una de estas clases contiene la información que le corresponde dentro del modelo matemático.

Otro atributo importante perteneciente a esta clase es una matriz que toma valores binarios, declarada de tipo *char*, que representa la asignación de los fragmentos en los sitios.

Por último, cuenta con una instancia de la clase *DataBaseManagement* para interactuar con el catálogo.

De los métodos de la clase, *TotalCost* es el más importante, pues se encarga de calcular el costo total de la asignación de los fragmentos en los sitios, según la matriz binaria de asignación. Otros métodos a destacar son *ProcessingConstraint* y *ReplicationConstraint* que verifican si la asignación actual cumple con las restricciones de procesamiento y de réplica respectivamente.

La clase AllocationMethod

Esta es una clase abstracta que cuenta con una instancia de la clase *OzsuModel*. De ella heredan las clases *GeneticAlgorithm* y *QLearning*. Una instancia de esta clase dirige el funcionamiento del problema. En tiempo de ejecución y en dependencia del método seleccionado por el usuario, se inicializa con cualquiera de los constructores de las clases derivadas y comienza el proceso de optimización, invocando al método *Execute* correspondiente al algoritmo seleccionado, haciendo uso del polimorfismo.

3.2.2.1 Implementación del Algoritmo Genético: la clase GeneticAlgorithm

La clase *GeneticAlgorithm* está compuesta por un arreglo de tamaño dos de instancias de la clase *Population* el cual representa dos poblaciones: la población actual y la población de la generación anterior. El mejor individuo de la población actual está representado por una instancia de la clase Individual.

Además cuenta con atributos que definen los parámetros de control del algoritmo, tales como el tamaño de la población, el número máximo de generaciones, la tolerancia, y las probabilidades de cruzamiento y mutación.

Para dar solución al problema se implementó un Algoritmo Genético generacional, donde la población nueva reemplaza totalmente a la antigua población.

Definición del cromosoma

El cromosoma usado es una matriz representada por valores de tipo *char* que toma valores binarios. Con esto se logra que su representación esté lo más cerca

posible de la solución real. Las columnas del cromosoma representan el índice de los sitios y las filas representan el índice de los fragmentos; la intersección de la fila i con la columna j significa que el fragmento i está almacenado en el sitio j.

Generación de la población inicial

Se generó la población inicial de forma parcialmente aleatoria. Se inicializa la matriz que representa el cromosoma de cada individuo de modo que los fragmentos se ubiquen en un solo sitio, escogido aleatoriamente. De esta forma se garantiza que la asignación inicial cumpla con la condición de réplica. Se tomaron 99 individuos como tamaño de la población.

Función de evaluación

Para asignar la adaptabilidad de cada individuo al medio, se evalúa en la función de costo del modelo planteado anteriormente. Si el individuo no cumple con alguna de las restricciones se penaliza, asignándole un valor de adaptabilidad cualitativamente muy bajo, (como se está minimizando este valor sería muy alto cuantitativamente).

Selección

Debido a su simplicidad y a la esencia no determinista del Algoritmo Genético, se usa el mecanismo se selección estocástico de torneo, en este caso de tamaño dos. Esta alternativa combina la idea del ranqueo de la población con el método de selección de la ruleta, evitando la convergencia temprana. Consiste en seleccionar de manera aleatoria dos individuos de la población y escoger el que tenga mayor adaptabilidad.

Cruzamiento

El cruzamiento se lleva a cabo si al generar un número aleatorio es menor o igual que la probabilidad de cruce. El cruce se realiza de manera análoga al método tradicional, se selecciona una fila de cruce de forma aleatoria, las filas que ocupan posiciones inferiores o igual a la fila de cruce se mantienen igual y las filas que ocupan posiciones superiores se intercambian entre ambos individuos. Se tomó 0.98 como probabilidad de cruzamiento.

Mutación

La mutación se lleva a cabo si al generar un número aleatorio es menor o igual que la probabilidad de mutación. Semejante al método tradicional, se selecciona de

forma aleatoria un elemento de la matriz del cromosoma del individuo y su valor se sustituye por su complemento. Se tomó 0.2 como probabilidad de mutación.

Condición de parada

Para detener el algoritmo se usa un criterio de terminación híbrido que consta de un mecanismo de tipo MAX y cierta tolerancia de convergencia. El mecanismo de tipo MAX fija un número máximo de generaciones, en este caso se escogió 80.

En el proceso de evaluación de la población se calcula la suma de las adaptaciones de los individuos no penalizados. Si la adaptabilidad del mejor individuo dividida entre el promedio de adaptabilidad de los individuos no penalizados es mayor que la tolerancia, se detiene el algoritmo. Para este problema se fijó la tolerancia en 0.995.

El método OneEpoch

Este método lleva a cabo una generación completa del algoritmo. En él se realizan los procesos de selección, cruce y mutación. La figura 3.2 muestra el seudocódigo asociado al método.

```
new_population[0] ← bestIndividual    // Elitismo
i ← 1
Mientras i < popSize do
    child1 ← new_population[i]
    child2 ← new_population[i + 1]
    parent1 ← SelectIndividual()
    parent2 ← SelectIndividual()
    if AppliesOperator(crossProbability) then
        CrossOver(parent1, parent2, child1, child2)
    else
        child1 ← parent1
        child2 ← parent2
    if AppliesOperator(mutProbability) then
        Mutate(child1)
    if AppliesOperator(mutProbability) then
        Mutate(child2)
    i ← i + 2
generation ← generation + 1
Evaluate()
```

Figura 3.2 Método *OneEpoch* de la clase *GeneticAlgorithm*

Para evitar la convergencia temprana se pone en práctica la variante del elitismo, donde el mejor individuo se copia para la siguiente generación, garantizando que si en alguna de ellas se alcanza el óptimo, no se pierda en los procesos de cruzamiento o mutación.

El método Execute

Este método inicia el proceso de optimización del Algoritmo Genético. Luego de completarse el ciclo generacional se copia la matriz que representa el cromosoma del mejor individuo de la población correspondiente a la última generación, en la matriz de solución del modelo; y se determina la calidad de la solución en dependencia si el individuo está penalizado o no. La figura 3.3 muestra el seudocódigo asociado al método.

```
Inicializar current _ population aleatorio
Evaluar current _ population
Mientras generacion < max Generation
    OneEpoch()
if bestIndividual penalizado
   goodSolution ← false
else
   goodSolution ← true
```

Figura 3.3 Método *Execute* de la clase *GeneticAlgorithm*

3.2.2.2 Implementación del algoritmo Q-Learning: la clase QLearning

El algoritmo implementado para dar solución al problema de asignación cuenta con las características que distinguen a un sistema de Aprendizaje Reforzado. A continuación se explica como se modeló cada uno de los elementos del algoritmo.

El agente

Su función es aprender a ubicar los fragmentos de la base de datos en los sitios de la red, de manera tal que se minimicen los costos de procesamiento y almacenamiento en un espacio de tiempo razonable.

El ambiente

Está constituido por el catálogo de la aplicación. Allí se almacena toda la información que se necesita en el modelo matemático, la cual se utiliza para estimar la función de costo y así ubicar los fragmentos de forma óptima o cercana a la óptima, cumpliendo con las restricciones del modelo.

Los estados

Están representados por cada par fragmento – sitio e indican la condición actual del ambiente en que se ubica el problema, es decir, si el fragmento se ubica o no en el sitio correspondiente.

Las acciones

Las posibles acciones a tomar por el agente son dos: *sí*, en caso de que el fragmento se ubique en el sitio correspondiente al par, según el estado en cuestión; o *no* en caso contrario.

La función de valor

Está representada por la matriz Q [estados, acciones], en la cual se acumulan los valores de recompensa que se han obtenido durante la interacción del agente con el ambiente. Luego de acumular suficiente conocimiento, su contenido le permite determinar a un agente si es factible o no, ubicar un fragmento en el sitio correspondiente.

Entre otros atributos, la clase *QLearning* cuenta con dos matrices. La primera es una matriz declarada de tipo *char* que toma valores binarios, esta matriz tiene la misma estructura que la matriz de solución del modelo y precisamente representa la mejor solución que se ha obtenido hasta el momento. La segunda matriz representa la tabla Q que almacena el conocimiento adquirido por el agente, las filas de esta matriz representan los estados del algoritmo y las columnas las acciones.

Además cuenta con atributos que definen los parámetros de control del algoritmo, tales como la cantidad de episodios, la velocidad de aprendizaje, el factor de descuento y el factor de exploración.

Inicialización del algoritmo

Antes de comenzar a ejecutar el ciclo de episodios del algoritmo, se necesita inicializar la matriz solución de la clase *OzsuModel* y la matriz Q. En el primer caso se tiene acceso a la matriz a través de una instancia de la clase *OzsuModel* que pertenece a la clase *AllocationMethod* y que es heredada por la clase *QLearning*. Para inicializarla se escoge una solución donde se ubica cada fragmento en un solo sitio, escogido de manera aleatoria. De esta forma se obtiene una solución inicial que cumpla con la condición de réplica. La segunda matriz se inicializa con todos sus valores en cero.

La política

Al encontrarse en un estado se debe seleccionar la acción que se va a tomar de acuerdo a la política seguida por el agente, basándose en la información almacenada en la tabla Q. En este caso se hace con una política ε-greedy; es decir, en la tabla Q [Estados] [Acciones] se busca con una probabilidad (1- ε) el menor valor de las posibles acciones para este estado, donde ε es la constante de exploración. Así mismo, se elige una acción aleatoria de las posibles acciones con probabilidad ε. En los primeros episodios este índice es alto (alrededor de un 95%) para que se exploren bastantes acciones y va disminuyendo a medida que transcurren los episodios, para poder explotar el conocimiento adquirido.

Si la acción seleccionada es cero, (indica que el fragmento no se ubica en el sitio correspondiente) y como consecuencia, el fragmento no queda ubicado en ningún sitio, se fuerza a tomar la acción uno; de esta forma se logra que en cada paso del algoritmo la matriz de solución cumpla con la condición de réplica. En la implementación del algoritmo este proceso lo realiza el método *SelectAction*.

La función de recompensa inmediata

La recompensa inmediata es el costo de ubicar el fragmento correspondiente al estado actual en los sitios donde se encuentra asignado, para todas las aplicaciones, utilizando la función de costo del modelo matemático. Si producto de la acción tomada se excede la capacidad de procesamiento del sitio, se penaliza el hecho de haberla escogido, tomando como recompensa inmediata el doble del costo de la asignación del fragmento en todos los sitios donde se encuentra. En la implementación del algoritmo este proceso lo realiza el método *Reward*.

Actualización de la matriz de solución

La matriz de solución del modelo se actualiza tomando la acción correspondiente al menor valor en la matriz Q para cada estado. Si la ubicación final de un fragmento incumple con la condición de réplica se le asigna el sitio correspondiente a la menor diferencia entre sus acciones en la tabla Q.

El método Execute

Este método inicia el proceso de optimización del algoritmo Q-Learning. En cada episodio se recorren todos los estados actualizando la matriz Q con la fórmula 2.8. Luego se actualiza la matriz de ubicación del modelo matemático y si cumple con

la restricción de procesamiento se evalúa en la función de costo total del modelo, si es menor que la obtenida hasta el momento, se almacena como la mejor solución. Luego de completarse el ciclo de episodios se copia la matriz que almacena la mejor asignación en la matriz de solución del modelo; y se determina la calidad de la solución. La figura 3.4 muestra el seudocódigo asociado al método.

Inicializar $Q(s,a)$ **en cero**
Inicializar $Sol(frg, site)$ **aleatorio**
Repetir (para cada episodio)
  Repetir (para cada estado s **del episodio)**
    $a \leftarrow SelectAction$ **usando una política *ε-greedy***
    $r \leftarrow \text{Re}\,ward(s,a)$
    $s' \leftarrow NextState(s)$
    $a' \leftarrow MinAction(s')$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$
  UpdateSolutionMatrix()
  (...) **//Actualización de los parámetros del algoritmo**
  if $Sol(frg, site)$ **cumple restricción de procesamiento**
    if $\cos t(Sol(frg, site)) < \cos t(btSol(frg, site))$ **then**
      $\cos t(btSol(frg, site)) \leftarrow \cos t(Sol(frg, site))$
si $Sol(frg, site)$ cumple las restricciones **then**
  $goodSolution \leftarrow true$
else
  $goodSolution \leftarrow false$

Figura 3.4 Método *Execute* de la clase *Q-Learning*

3.3 Descripción del asistente

Para ejecutar los algoritmos el asistente recibe como entrada la información almacenada en el catálogo. Luego de seleccionar el algoritmo mediante el cual se pretende dar solución al problema, se lleva a cabo el proceso de optimización. Finalmente se muestra la asignación alcanzada o un mensaje en caso de que no se haya encontrado una solución que cumpla con las restricciones del modelo matemático. El anexo 3 muestra la interfaz visual del asistente.

3.4 Casos de prueba

Se generaron mediante un programa casos de prueba aleatorios, que dados los rangos de valores entre los cuales se desea que se encuentren el número de fragmentos, de sitios y de aplicaciones, obtenga los datos necesarios para caracterizar una instancia del problema. En los tres primeros casos se encontró el óptimo (en milisegundos) por un método exacto de enumeración total. En los casos restantes el valor es el más pequeño que se ha obtenido en las etapas de prueba del software. El anexo 4 muestra los casos de prueba.

3.5 Resultados experimentales

En esta sección se presentan los resultados obtenidos en los experimentos realizados al software con el objetivo de evaluar la calidad de la solución y el tiempo de procesamiento.

Las pruebas fueron realizadas en una computadora con una Unidad de Procesamiento Central (CPU) Intel Pentium III, a 1000MHz de velocidad; con 256Mb de memoria RAM y sistema operativo Microsoft Windows XP Professional Service Pack 2.

Se ejecutó 20 veces cada algoritmo para los diferentes casos de prueba. Se determinó para cada caso el promedio de las soluciones, las veces que alcanzó el óptimo, la peor solución, el tiempo que tardó en obtener la solución (en segundos) y el porciento de desviación estándar del promedio con respecto al óptimo. El anexo 5 muestra los resultados de los experimentos realizados.

Las figuras 3.5 y 3.6 muestran las gráficas del comportamiento de cada algoritmo tomando en cuenta el porciento de desviación estándar del promedio de las soluciones respecto al óptimo, en la primera, y el tiempo de ejecución de cada algoritmo, en la segunda, para los diferentes casos de prueba.

Para instancias pequeñas y medianas del problema ambos algoritmos se comportan de manera similar en cuanto a la calidad de la solución. El algoritmo Q-Learning consume un menor tiempo de ejecución independientemente del tamaño del problema, pero a medida que este aumenta va perdiendo calidad en las soluciones, no así el Algoritmo Genético que mantiene cierta estabilidad en el porciento de desviación estándar respecto al óptimo.

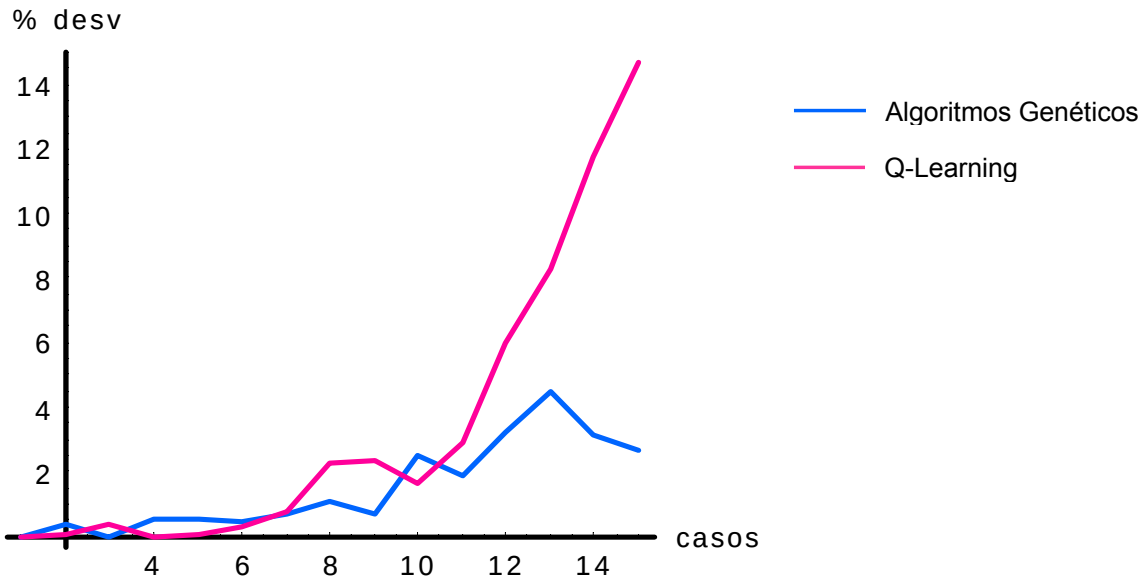


Figura 3.5 Comportamiento de los algoritmos teniendo en cuenta la desviación estándar

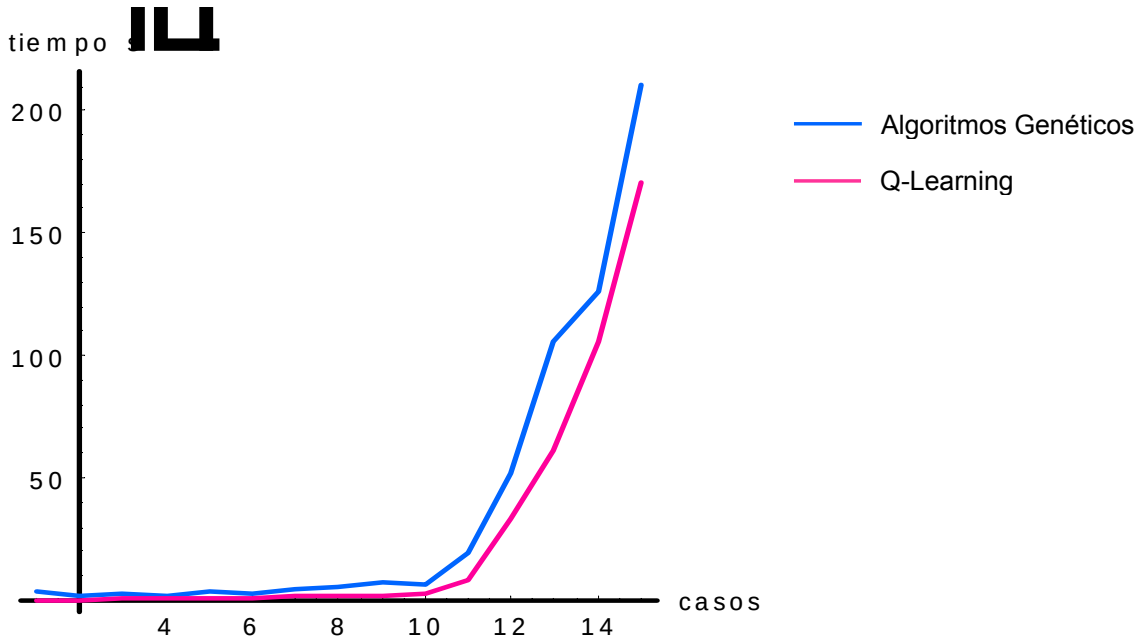


Figura 3.6 Comportamiento de los algoritmos teniendo en cuenta el tiempo de ejecución

CONCLUSIONES

1. Se realizó un estudio de la asignación de datos en la red según el modelo matemático Özsu y de las técnicas de Algoritmos Genéticos y Aprendizaje Reforzado para dar solución al mismo. El problema de dar solución al modelo de optimización pertenece a la clase NP-Completo, por lo que se justifica la aplicación de los métodos heurísticos para superar el alcance limitado de los métodos exactos.
2. Se diseñaron e implementaron los módulos que utilizan los Algoritmos Genéticos y el algoritmo Q-Learning para solucionar el problema de asignación de fragmentos, en un tiempo aceptable con un costo computacional considerablemente bueno.
3. Mediante las pruebas realizadas y los resultados obtenidos se demostró que ambas técnicas de Inteligencia Artificial son factibles de aplicar en la solución del problema de ubicación de fragmentos en el diseño de bases de datos distribuidas.
4. A medida que aumenta el tamaño del problema, la solución del mismo mediante los Algoritmos Genéticos tiende a ser mejor que la solución mediante el algoritmo Q-Learning, que resulta un método mucho más efectivo para instancias pequeñas y medianas del problema debido a la precisión de la solución y a la rapidez con que se obtiene.
5. Finalmente, se elaboró un asistente para llevar a cabo el diseño de la asignación, que será insertado en una herramienta general de diseño de bases de datos distribuidas.

RECOMENDACIONES

A pesar de que ambas técnicas de Inteligencia Artificial resultaron factibles de aplicar para resolver el problema de ubicación en el diseño de bases de datos distribuidas, se considera oportuno recomendar:

1. Ajustar los parámetros: cantidad de generaciones y tamaño de la población, en el Algoritmo Genético y cantidad de episodios en el algoritmo Q-Learning de acuerdo al tamaño del problema.

   Estos parámetros influyen en gran medida en el desempeño y la fortaleza de ambos algoritmos. Asignarles un valor independientemente del tamaño de la instancia podría ocasionar convergencia prematura o representar pobremente el espacio de soluciones si el valor es demasiado pequeño; por otro lado, si es demasiado grande seguramente desperdiciará recursos computacionales sin obtener mejoras apreciables. Por eso es conveniente obtener un mecanismo que adapte tales valores a las circunstancias particulares del problema.
2. Mezclar ambas técnicas para dar solución al problema.

   Debido a la rapidez y a la efectividad que demostró el algoritmo Q-Learning ante las instancias pequeñas del problema y al beneficio que puede ocasionar el hecho de elegir una población inicial factible y/o próxima al óptimo en el Algoritmo Genético, se puede aplicar el algoritmo Q-Learning sin considerar réplicas, con esto se logra disminuir considerablemente el espacio de soluciones incluso para los problemas grandes, y de esta manera obtener una población inicial del Algoritmo Genético que contenga la mayor cantidad posible de puntos factibles.

BIBLIOGRAFIA

Alander, J., (1992) *On optimal population size of genetic algorithms.* Proc. CompEuro 92, IEEE Computer Society Press.

Águila, L., (2001) *Aplicación de los algoritmos genéticos a la asignación de fragmentos en bases de datos distribuidas.* Tesis de licenciatura. Santa Clara, Departamento de Computación, Universidad Central de Las Villas.

Arkin, C., (1998) *Behavior-Based Robotics*. Ed. MIT Press; Cambridge, Massachusetts.

Artiles, M., (2001) *Herramienta para la ayuda a la fragmentación horizontal*. Tesis de licenciatura. Santa Clara, Departamento de Computación, Universidad Central de Las Villas.

Back, T., (1995) *Generalized convergence models for tournament - and (μ,λ) - selection*. Proc. 6th Intl. Conf. on Genetic Algorithms and their Applications.

Blickle, T. y L. Thiele, (1995) *A mathematical analysis of tournament selection*. Proc. 6th Intl. Conf. On Genetic Algorithms and their Applications.

Brooks, A., (1991) *Intelligence without reason.* MIT Press A.I. Memo No. 1293. USA.

Chakravarthy, S. et al., (1992) *An Objective Function for Vertically Partitioning Relations in Distributed Databases and its Analysis*, University of Florida Technical Report UF-CIS-TR-92-045.

Chakravarthy, S. et al., (1993) *A Formal Approach to the Vertical Partitioning Problem in Distributed Database Design, in Parallel and Distributed Information Systems (PDIS-2)*, San Diego.

Cichoz, P., (1995) *Truncating Temporal Differences: On the Efficient Implementation of TD for Reinforcement Learning*. Journal of Artificial Intelligence Research. Warsaw University of Technology, Warsaw, Poland.

Davis, L., (1991) *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.

Gálvez, D., (1998) *Algoritmos Genéticos*. Monografía para la especialización de Inteligencia Artificial, Universidad Cooperativa de Colombia.

Goldberg, D., (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company.

González, J. et al. (2001) *Agente Inteligente de Aprendizaje Reforzado aplicado al diseño de bases de datos distribuidas.* Departamento de Sistemas y Computación, Instituto Tecnológico Ciudad Madero.

Heitkotter, J. y D. Beasley, (1996) *The Hitch -Hiker's Guide to Evolutionary Computation.*

Hernández, A., (2005) *Agente de Aprendizaje Reforzado aplicado al Problema de Asignación en Bases de Datos Distribuidas*. Tesis de licenciatura. Santa Clara, Departamento de Computación, Universidad Central de Las Villas.

Kaelbling, L.; Littman, M. y A. Moore, (1996) *Reinforcement Learning: A Survey*. Brown University, USA.

Koenig, S. y G. Simmons, (1998). *Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models*. AAAI Press/MIT Press. Cambridge, Massachusetts.

Levine, D. (1995) *User's Guide to the PGAPack Parallel Genetic Algorithm Library*.

Martin, M., (1998) *Reinforcement Learning for embedded agents facing complex tasks* Ph D, Thesis, Universidad Politécnica de Cataluña.

Mei, H. y O. Sheng, (1992) *A Semantic Based Methodology for Integrated Computer-Aided Distributed Database Design*, Proc. 25th Hawaii International Conference on System Sciences, vol. 3 pp. 288- 299.

Morales, E., (2004) *Aprendizaje por Refuerzo*. Departamento de Computación. División de Ingeniería y Ciencias. Tecnológico de Monterrey (ITESM). Campus Cuernavaca.

Morell, A., (1999) *Un enfoque a la fragmentación vertical en bases de datos distribuidas.* Tesis de Maestría. Santa Clara, Departamento de Computación, Universidad Central de Las Villas.

Özsu, M y P. Valduriez, (1991) *Principles of Distributed Database Systems*. Prentice Hall.

Özsu, M., (1995) “Changing Infrastructure - New Demands on Distributed Data Management”*. International Conference on Computer Communications and Networks*.

Özsu, M y P. Valduriez, (1999) *Principles of Distributed Database Systems*. Second Edition. Prentice Hall.

Pérez, A., (1996) *Una introducción a la computación evolutiva.*

Rodríguez, A., (1998) *Bases de datos en ambientes distribuidos*. Especialización en Ciencias Electrónicas e Informática. Santa Clara, Departamento de Computación, Universidad Central de Las Villas.

Russell, S. y P. Norvig, (1995) *Artificial Intelligence, a modern approach*. Prentice Hall, New Jersey.

Sutton, R. y A. Barto, (1998) *Reinforcement Learning. An Introduction*. Ed. MIT Press, Cambrige, Massachusetts.

Watkins, C., (1999) *Learning from Delayed Rewards*. Ph. D. Thesis, King’s College, Cambridge.

X, Lin.; Orlowska, M. y Y. Zhan, (1998) *On Data Allocation with the Minimum Overall Communication Cost in Distributed Database Design*. Proc. Of ICCI'93, pp. 539-544.

Anexo 1. Descripción del catálogo del asistente

Tabla: LogicalFragments — Fragmentos lógicos de la BD

Atributo	Tipo	Descripción
IdLogFragm	Auto numérico	Identificador del fragmento lógico
FragmentName	Text[25]	Nombre del fragmento lógico
Cardinality	Integer	Cardinalidad del fragmento lógico

Tabla: PhysicalFragments — Fragmentos físicos de la BD

Atributo	Tipo	Descripción
IdPhysicalFragment	Auto numérico	Identificador del fragmento físico
IdLogFragm	Integer	Identificador del fragmento lógico
SiteName	Text[25]	Nombre del sitio donde fue ubicado
PhysicalFragmName	Text[25]	Nombre del fragmento físico

Tabla: Sites — Sitos de la red

Atributo	Tipo	Descripción
SiteName	Text[25]	Nombre del sitio
IP	Text[20]	IP del sitio
TotalDiskSpace	Double	Capacidad de disco duro
FreeDiskSpace	Double	Espacio libre de disco duro
ReadAccessTime	Double	Costo de procesar una unidad de trabajo
WriteAccessTime	Double	Costo de almacenar un bloque de datos
MaxAnswerTime	Double	Tiempo de respuesta máximo admisible

Tabla: GlobalAttributes — Atributos globales de las relaciones

Atributo	Tipo	Descripción
IdAtt	Auto numérico	Identificador del atributo
IdType	Integer	Identificador del tipo del atributo
AttName	Text[25]	Nombre del atributo
Pkey	Yes/No	Si es llave primaria
Fkey	Yes/No	Si es llave foránea
AttSize	Integer	Longitud del atributo

Tabla: Applications — Aplicaciones

Atributo	Tipo	Descripción
IdApp	Auto numérico	Identificador de la aplicación
MaxAnswerTime	Double	Tiempo de respuesta máximo admisible
ExecuteTime	Double	Tiempo de ejecución

Tabla: FragmentAttributes — Atributos de los fragmentos

Atributo	Tipo	Descripción
IdLogFragm	Integer	Identificador fragmento lógico
IdGlobalAtt	Integer	Identificador del atributo global

Tabla: App_LogicalFragment — Caracterización de las aplicaciones

Atributo	Tipo	Descripción
IdApp	Integer	Identificador de la aplicación
IdLogfrag	Integer	Identificador del fragmento lógico
ReadAccessCount	Integer	Accesos de lectura que la aplicación realiza sobre el fragmento durante su ejecución
WriteAccessCount	Integer	Accesos de actualización que la aplicación realiza sobre el fragmento durante su ejecución
Selectivity	Integer	Selectividad del fragmento lógico respecto a la aplicación

Tabla: AccessTime — Costo de comunicación entre los sitios

Atributo	Tipo	Descripción
FromStation	Text[25]	Nombre del primer sitio
ToStation	Text[25]	Nombre del segundo sitio
AverageTime	Double	Tiempo de acceso entre los sitios (x1024)

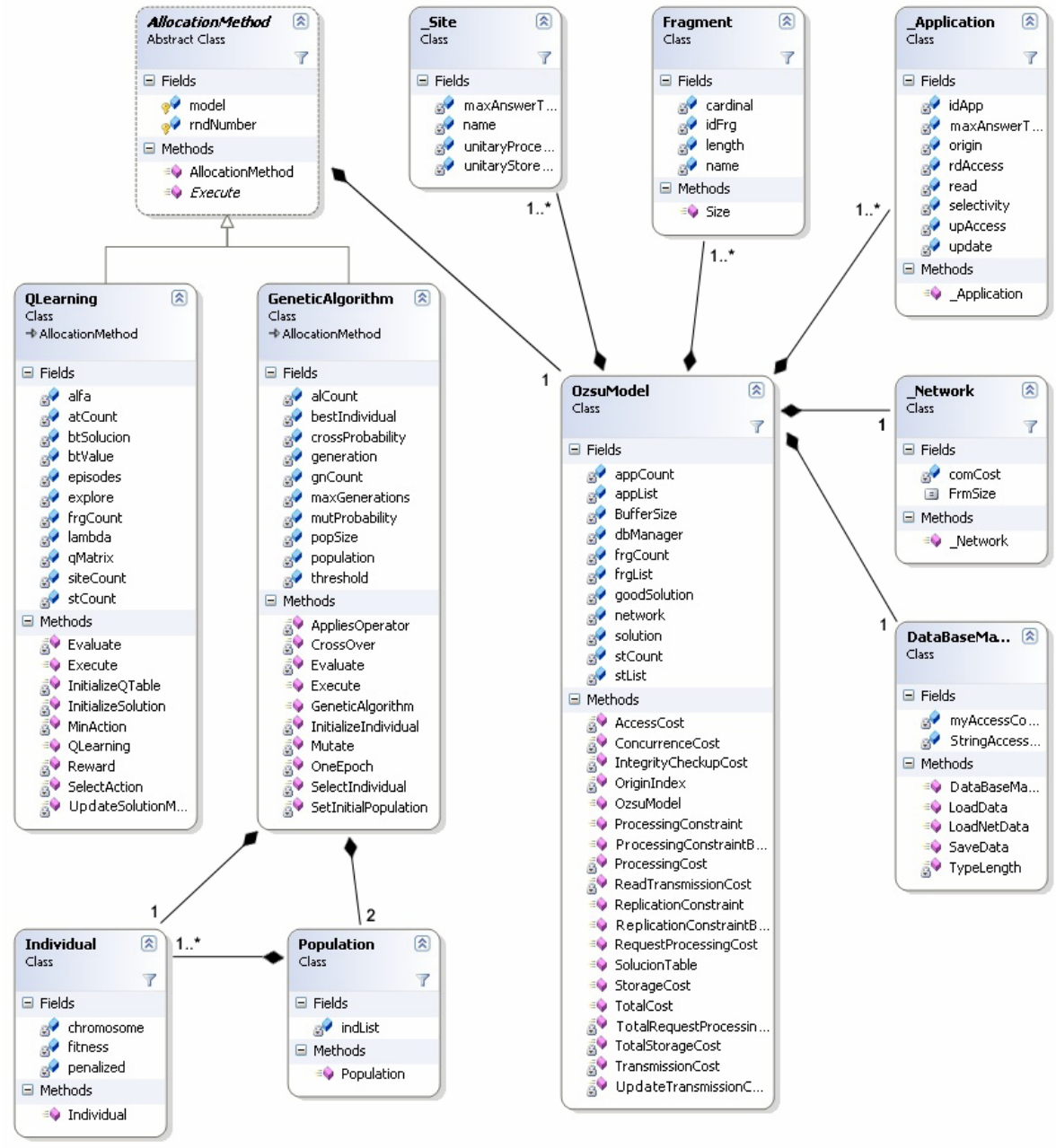
Tabla: App_Site — Muestra los sitios donde se originan las aplicaciones

Atributo	Tipo	Descripción
IdApp	Integer	Identificador de la aplicación
SiteName	Text[25]	Nombre del sitio de origen de la aplicación

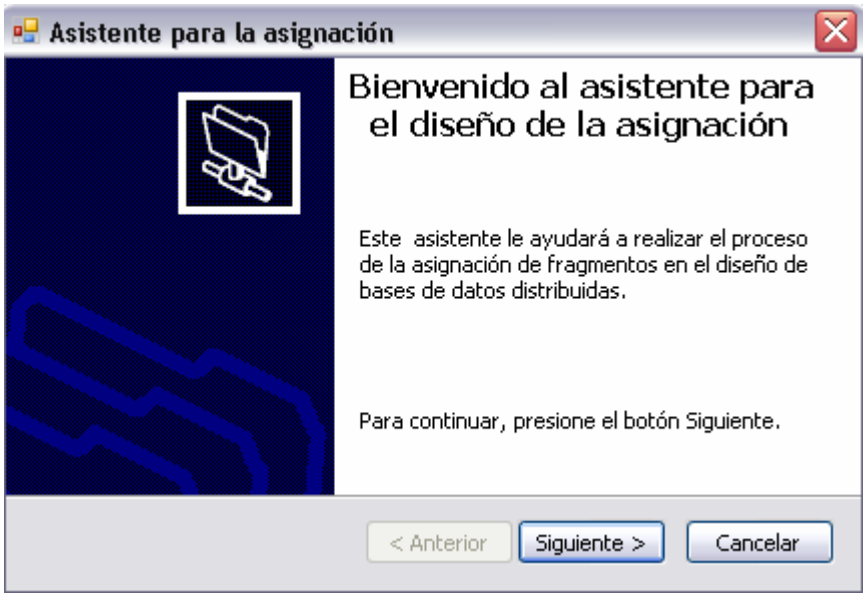
Tabla: DataTypes — Tipos de datos de SQL Server

Atributo	Tipo	Descripción
IdType	Auto numérico	Identificador del tipo de dato
Type	Text[30]	Nombre del tipo de dato

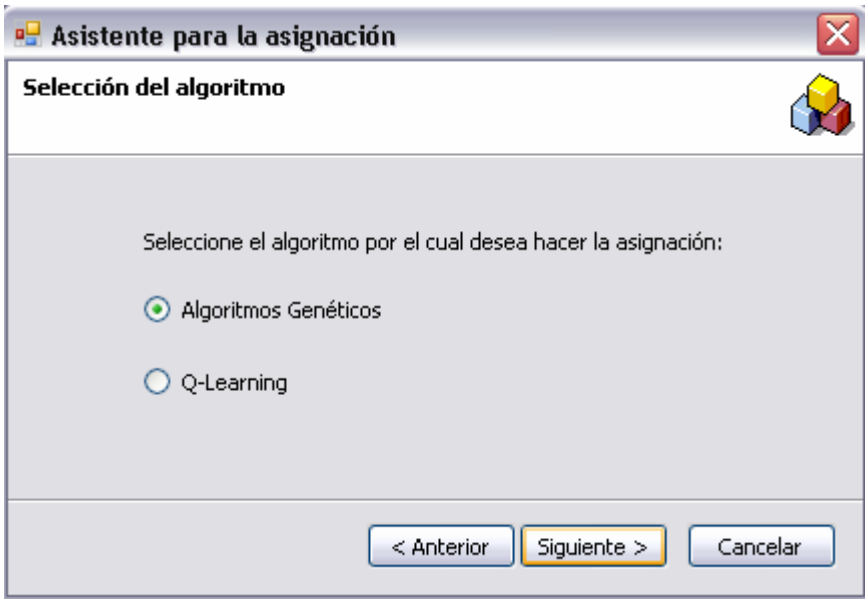
Anexo 2. Diagrama de clases



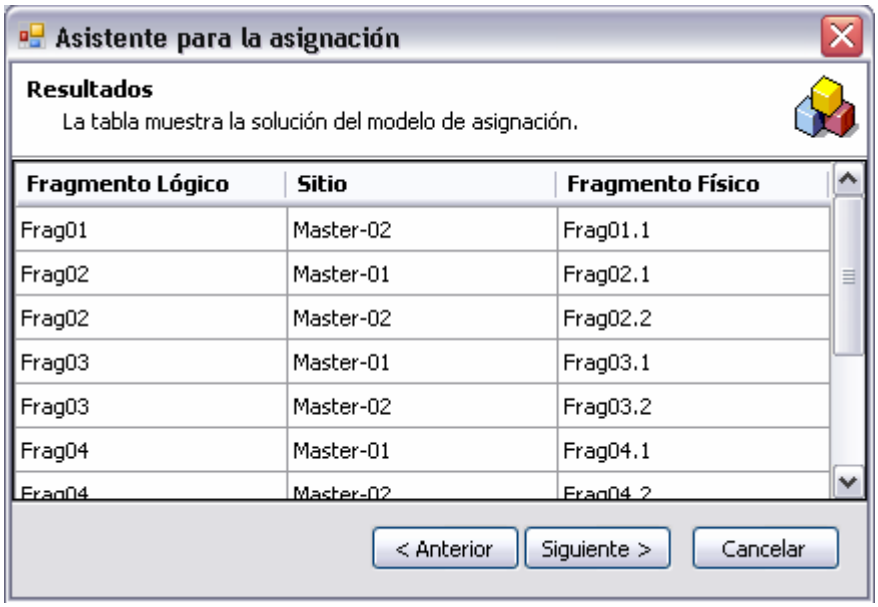
Anexo 3. Interfaz visual del asistente



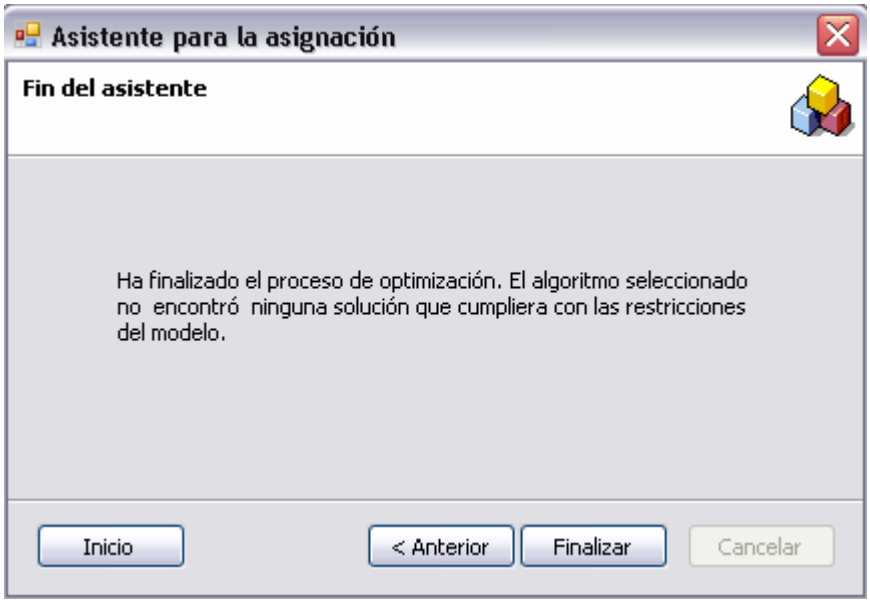
Ventana de presentación.



Ventana de selección del algoritmo.



En esta ventana se muestra el resultado de la asignación.



Esta ventana se muestra en caso de no hallarse una solución que cumpla con las restricciones del modelo.

Anexo 4. Casos de prueba

Caso	Fragmentos	Sitios	Aplicaciones	Óptimo (ms)
1	6	2	22	61104,85
2	7	4	5	13549,53
3	3	10	6	4479,58*
4	17	3	8	52659,82*
5	24	2	10	102502,55*
6	16	3	22	54498,22*
7	8	6	16	71426,56*
8	7	11	5	26087*
9	11	8	7	40975,05*
10	25	7	5	51282,82*
11	14	9	29	252440,2*
12	24	16	16	198615,14*
13	27	18	18	369647,17*
14	30	20	20	339398,45*
15	36	24	24	549275,72*

* No se conoce el óptimo con exactitud

Anexo 5. Resultados de los experimentos realizados al software

		AG					QL				
Caso	**Óptimo**	**AVE**	**VO**	**max**	**t (s)**	**% desv**	**AVE**	**VO**	**max**	**t (s)**	**% desv**
1	61104,85	61104,85	20	61104,85	3,42	0	61106,27	19	61133,38	0,25	0
2	13549,53	13599,85	16	13961,25	1,65	0,37	13559,16	17	13613,77	0,24	0,07
3	4479,58*	4479,58	20	4479,58	2,5	0	4496,48	19	4817,7	0,63	0,38
4	52659,82*	52951,11	11	53809,06	1,88	0,55	52659,82	20	52659,82	0,67	0
5	102502,55*	103040,35	1	104292,49	3,6	0,52	102571,99	5	102913,56	0,66	0,07
6	54498,22*	54726,56	13	55288,28	2,86	0,42	54651,85	15	55285,4	0,67	0,28
7	71426,56*	71918,28	6	72957,96	4,86	0,69	71962,17	4	72621,84	1,75	0,75
8	26087*	26371,27	2	26987,04	5,85	1,09	26672,93	1	27983,13	2,12	2,25
9	40975,05*	41251,02	1	41613,12	7,49	0,67	41922,69	0	43213,53	1,67	2,31
10	51282,82*	52540,99	0	53964,49	6,38	2,53	52080,81	2	52861,03	2,4	1,64
11	252440,2*	257096,76	2	261547,33	19,49	1,84	259728,43	1	266100,01	8,37	2,89
12	198615,14*	204943,25	2	212613,52	52,15	3,19	210416,34	0	223640,47	33,55	5,94
13	369647,17*	386241,44	0	406948,68	105,65	4,49	400167,30	0	426956,78	60,91	8,26
14	339398,45*	349973,94	1	361746,39	125,8	3,12	379218,21	0	403171,71	105,53	11,73
15	549275,72*	563707,64	1	580360,95	210,79	2,63	629684,21	0	688023,4	170,2	14,64

* No se conoce el óptimo con exactitud