SciVerse ScienceDirect
Disponible en www.sciencedirect.com

Revista Iberoamericana de Automática e Informática industrial 10 (2013) 344–355

www.elsevier.es/RIAI

Algoritmo genético permutacional para el despliegue y la planificación de sistemas de tiempo real distribuidos

Ekain Azketa[a,*], J. Javier Gutiérrez[b], Marco Di Natale[c], Luís Almeida[d], Marga Marcos[e]

[a] *IK4-Ikerlan Centro de Investigaciones Tecnológicas, Área de Tecnologías de Software, Mondragón, España*
[b] *Universidad de Cantabria, Grupo de Computadores y Tiempo Real, Santander, España*
[c] *Scuola Superiore Sant'Anna, Real-Time Systems Laboratory, Pisa, Italia*
[d] *Universidade do Porto, Departamento de Engenharia Eletrotécnica e de Computadores, Oporto, Portugal*
[e] *Universidad del País Vasco, Departamento de Ingeniería de Sistemas y Automática, Bilbao, España*

Resumen

El despliegue y la planificación de tareas y mensajes en sistemas de tiempo real distribuidos son problemas NP-difíciles (*NP-hard*), por lo que no existen métodos óptimos para solucionarlos en tiempo polinómico. En consecuencia, estos problemas son adecuados para abordarse mediante algoritmos genéricos de búsqueda y optimización. En este artículo se propone un algoritmo genético multiobjetivo basado en una codificación permutacional de las soluciones para abordar el despliegue y la planificación de sistemas de tiempo real distribuidos. Además de desplegar tareas en computadores y de planificar tareas y mensajes, este algoritmo puede minimizar el número de computadores utilizados, la cantidad de recursos computacionales y de comunicaciones empleados y el tiempo de respuesta de peor caso medio de las aplicaciones. Los resultados experimentales muestran que este algoritmo genético permutacional puede desplegar y planificar sistemas de tiempo real distribuidos de forma satisfactoria y en tiempos razonables.

Palabras Clave:
Sistemas de tiempo real, Algoritmos de planificación, Algoritmos genéticos, Optimizaciones multiobjetivo.

1. Introducción

Un sistema de tiempo real distribuido está basado en aplicaciones compuestas por tareas y mensajes con restricciones temporales. Las tareas se ejecutan en computadores conectados a redes de comunicación deterministas a través de las que se transmiten los mensajes. Los sistemas de control de entornos como la industria, automoción, aviónica, etc. habitualmente son sistemas de tiempo real distribuidos.

El despliegue y la planificación de sistemas de tiempo real distribuidos son problemas complejos. Por una parte, hay que definir las aplicaciones y sus requisitos funcionales y no funcionales, como por ejemplo las restricciones temporales. Por otra parte, ha de diseñarse la arquitectura física de modo que las aplicaciones puedan ejecutarse cumpliendo todas las restricciones. La ejecución de las aplicaciones en la arquitectura física conlleva algunas otras dificultades, como el establecimiento del despliegue y la planificación de las tareas en los computadores y los mensajes en las redes, es decir, dónde y cuándo se ejecuta cada tarea y se transmite cada mensaje.

El despliegue y la planificación de tareas y mensajes en sistemas de tiempo real distribuidos son problemas NP-difíciles (*NP-hard*) (Tindell et al., 1992), por lo que no existen métodos óptimos para solucionarlos en tiempo polinómico. Esta complejidad hace del despliegue y la planificación de sistemas de tiempo real distribuidos problemas adecuados para abordarse mediante algoritmos genéricos de búsqueda y optimización.

En este artículo se propone un algoritmo genético multiobjetivo basado en una codificación permutacional de las soluciones para acometer el diseño de sistemas de tiempo real distribuidos. Además de desplegar tareas en computadores y planificar tareas y mensajes, este algoritmo puede minimizar el número de computadores utilizados, la cantidad de recursos computacionales y de comunicaciones empleados y el tiempo de respuesta de peor caso medio de las aplicaciones.

La estructura de este artículo es la siguiente. En la sección 2 se discuten los trabajos relacionados. En la sección 3 se describe el modelo de sistema. En la sección 4 se exponen los objetivos y la definición de los problemas abordados. En la sección

* Autor en correspondencia
Correos electrónicos: eazketa@ikerlan.es (Ekain Azketa), gutierjj@unican.es (J. Javier Gutiérrez), marco.dinatale@sssup.it (Marco Di Natale), lda@fe.up.pt (Luís Almeida), marga.marcos@ehu.es (Marga Marcos)

http://dx.doi.org/10.1016/j.riai.2013.05.006

5 se detalla el algoritmo genético propuesto, y su evaluación se expone en la sección 6. Finalmente, en la sección 7 se resumen las conclusiones obtenidas y se esboza el trabajo futuro.

2. Trabajos relacionados

En (Tindell et al., 1992) se propone *simulated annealing* (SA) (Kirkpatrick, 1984) para solucionar el despliegue de tareas y la planificación de tareas y mensajes mediante *rate monotonic scheduling* pero sin minimizar el número de computadores usados. En (Di Natale and Stankovic, 1995) usan SA para crear planificaciones en sistemas de tiempo real distribuidos con el objetivo de minimizar el *jitter* de las tareas y cumplir el plazo de las tareas y de los flujos de extremo a extremo. En (Dick and Jha, 2002) y (Shang et al., 2007) proponen un algoritmo SA paralelo recombinativo (PRSA) para seleccionar los recursos hardware necesarios, desplegar tareas en computadores y crear planificaciones en sistemas de tiempo real con los objetivos de minimizar el precio y la energía consumida. Sin embargo, estos dos últimos trabajos están más enfocados a arquitecturas *system-on-chip* que a sistemas de tiempo real distribuidos.

Entre los basados en métodos de programación matemática (Minoux, 1986), el trabajo en (Hladik et al., 2008) propone *constraint satisfaction programming* (CSP) (Tsang, 1993) para desplegar y planificar tareas en sistemas de tiempo real distribuidos conectados por un bus CAN. En (Metzner and Herde, 2006) se usa una variante de CSP denominada SAT para el despliegue y la planificación de tareas mediante *deadline monotonic scheduling*. El trabajo en (Davare et al., 2007) propone la programación geométrica (GP) (Boyd et al., 2007) para optimizar el periodo de las tareas y los mensajes, que ya están desplegados y tienen periodos y plazos. En (Zheng et al., 2007) se propone *mixed-integer linear programming* (MILP) (Schrijver, 1998) para desplegar tareas en computadores, empaquetar señales en mensajes y asignar prioridades fijas a tareas y mensajes en sistemas de tiempo real distribuidos conectados mediante redes CAN y con el objetivo de minimizar los tiempos de respuesta de peor caso. Ninguno de estos trabajos citados aborda la minimización del número de computadores usados.

Los trabajos en (Porto and Ribeiro, 1995; Porto et al., 2000) proponen *tabu search* (TS) (Glover, 1986) para desplegar tareas en computadores y crear planificaciones no expulsoras en sistemas de tiempo real distribuidos con el objetivo de minimizar el tiempo total de la planificación pero no el número de computadores usados. En (Chen and Lin, 2000) se usa TS para desplegar tareas en computadores con el objetivo de minimizar el tráfico de red y el número de computadores usados pero sin llevar a cabo una planificación explícita.

En (Pop et al., 2003a) se proponen heurísticos (Pearl, 1984) para desplegar tareas en computadores y planificar tareas y mensajes en sistemas de tiempo real distribuidos que combinan los paradigmas *time-triggered* y *event-triggered*. También han desarrollado heurísticos para desplegar y planificar tareas con el objetivo de maximizar la extensibilidad del sistema (Pop et al., 2002). En (Pop et al., 2003b) se presentan algunos heurísticos para sintetizar, analizar y optimizar el despliegue de tareas en computadores, la asignación de políticas de planificación, la asignación de prioridades a tareas y la asignación de ventanas temporales de comunicación a computadores. El modelo de sistema combina simultáneamente los paradigmas *time-triggered* y *event-triggered* tanto en computadores como en redes. Sin embargo, ninguno de estos trabajos citados aborda la minimización del número de computadores usados.

En (Mitra and Ramanathan, 1993) y (Monnier et al., 1998) se propone un algoritmo genético (GA) (Holland, 1975) para desplegar tareas en computadores y crear planificaciones no expulsoras en sistemas de tiempo real distribuidos. En (Samii et al., 2009) se usa un GA para asignar prioridades fijas a tareas y prioridades fijas e identificadores de trama a mensajes en sistemas de tiempo real distribuidos conectados por redes FlexRay, pero no abordan el despliegue de tareas. En (Hamann et al., 2006) se propone un GA multiobjetivo para asignar prioridades fijas a tareas y ventanas temporales a mensajes en sistemas de tiempo real distribuidos conectados por redes TDMA. Una característica distintiva de este trabajo es que una solución candidata puede estar compuesta por varios cromosomas que codifican diferentes elementos y/u objetivos. Sin embargo, tampoco se aborda el despliegue de tareas.

En (Azketa et al., 2011b) y en (Azketa et al., 2011a) se obtienen buenos resultados para la asignación de prioridades y la optimización del tráfico del bus CAN en sistemas de tiempo real distribuidos, respectivamente, mediante la aplicación de algoritmos genéticos con variantes de la codificación permutacional propuesta en el presente artículo.

3. Modelo de sistema

La arquitectura física está compuesta por múltiples computadores $Px \in P$ $(x = 1, 2, ..., Q_P)$ que pueden ser heterogéneos en su núcleo y velocidad de computación. Cada computador puede proveer a las tareas diferentes recursos hardware y/o software, tales como sensores, actuadores, programas, bibliotecas, etc. A los computadores se les puede configurar un límite máximo de utilización de los recursos computacionales $U_{Px}^{max} \in \mathbb{R}$ $(0 \leq U_{Px}^{max} \leq 1)$. Los computadores están conectados a una red de comunicación N con una velocidad de transmisión BR concreta. A la red se le puede configurar un límite máximo de utilización de los recursos de comunicaciones $U_N^{max} \in \mathbb{R}$ $(0 \leq U_N^{max} \leq 1)$.

La arquitectura lógica esta compuesta por tareas $t_i \in t$ $(i = 1, 2, ..., Q_t)$, cada una de las cuales puede tener un periodo de activación T_{t_i}. Dependiendo de los requisitos hardware y/o software de la tarea y los recursos hardware y/o software de los computadores, una tarea tiene un conjunto concreto de computadores candidatos $P_{cand}^{t_i}$ donde puede ser desplegada. Por ejemplo, si una tarea requiere una biblioteca de programa concreta, la tarea no podrá ser desplegada en los computadores que no dispongan de dicha biblioteca. El conjunto de tareas desplegadas en el computador Px se expresa como t_{Px}. En cada computador candidato, una tarea t_i tiene un tiempo de ejecución de peor caso $C_{t_i}^{Px}$ y un tiempo de ejecución de mejor caso $C_{t_i}^{Px,b}$. Se dice que un computador es prescindible si todas las tareas para las que es un computador candidato tienen más de un computador candidato. El conjunto de computadores prescindibles se

representa como P_{dis}. Una tarea desplegada y priorizada tiene un tiempo de respuesta de peor caso R_{t_i} y un tiempo de respuesta de mejor caso $R^b_{t_i}$, que pueden calcularse con las técnicas de análisis apropiadas.

La arquitectura lógica también esta compuesta por mensajes $m_j \in m$ ($j = 1, 2, ..., Q_m$), cada uno de los cuales tiene una longitud de peor caso L_{m_j}, una longitud de mejor caso $L^b_{m_j}$, y puede tener un periodo de activación T_{m_j}. Los tiempos de transmisión de peor caso C_{m_j} y mejor caso $C^b_{m_j}$ de un mensaje se calculan haciendo la división de sus longitudes de peor y mejor caso con la velocidad de transmisión BR de la red de comunicación.

Un mensaje tiene una tarea transmisora Tx_{m_j} y una o más tareas receptoras Rx_{m_j}. Si las tareas transmisora y receptora están desplegadas en el mismo computador, el mensaje se envía a través de un mecanismo de memoria compartida cuyo tiempo de transmisión se considera despreciable. Si no fuese despreciable, podría tenerse en cuenta en los tiempos de ejecución de la tarea emisora o receptora del mensaje. Un mensaje desplegado y priorizado tiene un tiempo de respuesta de peor caso R_{m_j} y un tiempo de respuesta de mejor caso $R^b_{m_j}$, que pueden calcularse con las técnicas de análisis apropiadas.

El sistema puede tener elementos heredados que podrían ser subsistemas ya desarrollados que se quieren integrar como parte de otros nuevos, pero que tienen características que no se pueden cambiar, como por ejemplo la vinculación a un procesador o una red, o la asignación de ciertas prioridades. Un algoritmo genético de síntesis permite optimizar todo el sistema aunque algunas características o parámetros no se puedan cambiar.

Un *step* τ_i es una actividad genérica que puede ser una tarea o un mensaje. El conjunto de tareas y mensajes crean un grafo dirigido en el que hay múltiples rutas o flujos de extremo a extremo $\Gamma_l \in \Gamma$ ($l = 1, 2, ..., Q_\Gamma$). Un flujo de extremo a extremo puede ser un *step* $\Gamma_l = [\tau_o]$ o una secuencia ordenada de tareas comunicantes $\Gamma_l = [\tau_o, ..., \tau_p]$ que comenzando desde τ_o llega hasta τ_p atravesando tareas que reciben un mensaje de su predecesora y lo transmiten a su sucesora. Más de un flujo de extremo a extremo puede empezar y/o terminar en la misma tarea, y una tarea y/o mensaje puede pertenecer a más de un flujo de extremo a extremo. Un flujo de extremo a extremo se activa con un periodo T_{Γ_l} y puede tener un plazo al final de D_{Γ_l}. Si las tareas y los mensajes de un flujo de extremo a extremo no tienen definido un periodo, se considera que se activan con el periodo del flujo de extremo a extremo si solo pertenecen a uno de ellos o todos los flujos a los que pertenecen tienen el mismo periodo. El plazo puede ser mayor que el periodo, por lo que en cada instante puede haber pendientes varias instancias del flujo de extremo a extremo. Un flujo de extremo a extremo desplegado y priorizado tiene un tiempo de respuesta de peor caso R_{Γ_k} y otro de mejor caso $R^b_{\Gamma_l}$, que pueden ser calculados con las técnicas de análisis apropiadas. A un flujo de extremo a extremo se le puede configurar una utilización temporal máxima $UR^{max}_{\Gamma_l} \in \mathbb{R}$ que permite controlar su holgura temporal mínima. $UR^{max}_{\Gamma_l}$ es el valor máximo que puede limitarse por configuración para $R_{\Gamma_l}/D_{\Gamma_l}$ y su rango es

$$\frac{\sum_{\forall \tau_i \in \Gamma_l} \min_{\forall \Gamma_k, \tau_i \in \Gamma_k} \left[C^b_{\tau_i}\right]}{D_{\Gamma_l}} \leq UR^{max}_{\Gamma_l} \leq 1$$

La arquitectura lógica soporta la planificación por prioridades fijas y la planificación cíclica ordinal. Si se utiliza planificación por prioridades fijas, se asignan prioridades a las tareas y los mensajes. Si se usa planificación cíclica ordinal, se ordena la activación de las tareas y los mensajes.

La arquitectura lógica es independiente del modelo de activación, por lo que el algoritmo genético propuesto soporta los paradigmas *event-triggered*, *time-triggered* y mixto. En el modelo de activación *event-triggered*, los flujos de extremo a extremo se activan con periodos definidos, las tareas se activan con la recepción de mensajes, y los mensajes se activan con la finalización de sus tareas transmisoras. Cuando todos los flujos de extremo a extremo son secuencias lineales de tareas y mensajes *event-triggered* y basados en prioridades fijas, la planificabilidad del sistema puede analizarse mediante el método holístico (Tindell and Clark, 1994) o con las técnicas basadas en *offsets* (Palencia and Harbour, 1998, 1999), en combinación con métodos de análisis recientes para redes de comunicación basadas en prioridades fijas (Azketa et al., 2012a). Por el contrario, en el paradigma *time-triggered*, las tareas y los mensajes se activan con periodos definidos, independientemente de la llegada de los mensajes a recibir o de la finalización de la ejecución de las tareas transmisoras. El análisis de sistemas *time-triggered* es más simple que el de los *event-triggered* debido a que el tiempo de respuesta de peor caso de las tareas y los mensajes se calcula sin tener en cuenta el *jitter* de activación heredado de las actividades previas (Di Natale et al., 2007). El modelo de activación mixto combina tareas y mensajes *event-triggered* y *time-triggered*, y su planificabilidad puede analizarse con técnicas como la expuesta en (Di Natale et al., 2007).

4. Objetivos y definición del problema

El despliegue y la planificación de un sistema de tiempo real distribuido se lleva a cabo con el objetivo de cumplir (1) los requisitos funcionales, de forma que el comportamiento del sistema se ajuste a las características de funcionamiento especificadas, y (2) los requisitos no funcionales, de modo que la operación del sistema se corresponda con las restricciones impuestas. Los requisitos no funcionales más habituales de un sistema de tiempo real distribuido son el *rendimiento*, el *costo* y la *extensibilidad*, que se emplean como métricas para cuantificar la calidad de las soluciones obtenidas para los problemas de despliegue y planificación.

En sistemas de tiempo real, el principal requisito no funcional de rendimiento está relacionado con aspectos temporales. El rendimiento temporal es mayor cuantos más plazos se cumplan y menores sean los tiempos de respuesta de peor caso (Zheng et al., 2007).

El requisito no funcional de costo está relacionado con el precio monetario del sistema, típicamente en términos de número de elementos físicos utilizados. En un sistema de tiempo real

distribuido, la métrica de costo es mejor cuanto menor sea la cantidad de computadores y dispositivos de red utilizados (Dick and Jha, 2002; Shang et al., 2007).

El requisito no funcional de extensibilidad está relacionado con la capacidad del sistema de aceptar adiciones futuras, típicamente de extensiones en los elementos lógicos existentes (Zhu et al., 2009) y/o incorporaciones de nuevos elementos lógicos (Pop et al., 2002). En un sistema de tiempo real distribuido, la métrica de extensibilidad es mejor cuanto menor sea la utilización de los recursos de los computadores y redes, y menores sean los tiempos de respuesta de peor caso con respecto a sus plazos, ya que así la plataforma tiene más recursos y más holgura temporal para aceptar nuevas tareas y mensajes.

El despliegue y la planificación son problemas de búsqueda y optimización cuyos objetivos son maximizar las métricas de rendimiento, costo y extensibilidad del sistema de tiempo real distribuido resultante. En general, un problema de búsqueda y optimización consiste en la asignación de valores a algunas variables de forma que se cumplan todas las restricciones y algunos objetivos se maximicen o minimicen. El algoritmo genético permutacional propuesto realiza la búsqueda optimizada de los siguientes parámetros:

- *Despliegue:* Desplegar cada tarea en uno de sus computadores candidatos.
- *Planificación:* Asignar a cada tarea y mensaje una prioridad de ejecución/transmisión o un orden de activación.

La búsqueda optimizada de los anteriores factores se lleva a cabo cumpliendo las siguientes restricciones:

- *Despliegue:* No rebasar el límite de utilización máxima de los recursos computacionales de los computadores y los recursos de comunicación de la red.
- *Planificación:* Cumplir los plazos de los flujos de extremo a extremo.

La búsqueda optimizada del despliegue y la planificación tiene los siguientes objetivos:

- *Despliegue:* Minimizar el número de computadores y las utilizaciones de computadores y redes.
- *Planificación:* Minimizar los tiempos de respuesta de peor caso de los flujos de extremo a extremo.

5. Algoritmo genético permutacional

5.1. Introducción

Un algoritmo genético (Holland, 1975) es un método probabilístico genérico de búsqueda y optimización basado en evolucionar un conjunto inicial de soluciones candidatas, denominado población de individuos, hacia mejores soluciones a lo largo de generaciones de poblaciones mediante mecanismos bioinspirados como la herencia, la selección natural, la adaptación al medio, el cruzamiento y la mutación.

Inicialmente, se crea una población de individuos de forma aleatoria o guiada. En cada una de las generaciones del algoritmo genético se evalúa la aptitud de los individuos. La aptitud denota el grado de adaptación de cada individuo, lo que en términos de búsqueda y optimización significa cómo de bien soluciona el problema cada solución candidata. En virtud de la ley de selección natural, los individuos más adaptados tienen mayores probabilidades de reproducirse y crear descendencia. En un algoritmo genético, el citado fenómeno se traduce en que las mejores soluciones candidatas tienen más posibilidades de cruzarse con otras para generar nuevas soluciones candidatas, que conforman una nueva generación heredera de algunas de las características de las soluciones cruzadas. Además, las nuevas soluciones candidatas pueden sufrir mutaciones aleatorias a fin de impedir parecidos excesivos entre sí. Las soluciones candidatas de la nueva generación es evalúan, seleccionan, cruzan y mutan para crear otra generación. La nueva generación se emplea en la siguiente iteración del algoritmo, y así sucesivamente hasta alcanzar la condición de parada, que puede ser la creación de un número máximo de generaciones, el alcance de un nivel de aptitud suficientemente satisfactorio por parte de alguna solución candidata, o la ejecución del algoritmo durante un tiempo máximo, por mencionar algunos criterios empleados habitualmente. Los algoritmos genéticos requieren definir y/o configurar determinados parámetros, siendo los más relevantes el tamaño de la población, los tipos y las probabilidades de aplicación de los operadores de selección, cruzamiento y mutación, la función de evaluación de la aptitud de las soluciones candidatas, el método de reemplazo de las poblaciones antiguas por las nuevas y el criterio de finalización.

Un algoritmo genético tiene algunas ventajas destacables. El manejo simultáneo de múltiples y posiblemente muy distintas soluciones candidatas le confiere la capacidad de explorar extensas áreas del espacio de búsqueda de soluciones, que puede ser no lineal. Asimismo, como la mayoría de las técnicas de búsqueda y optimización basadas en principios aleatorios, tiene la habilidad de escapar de los óptimos locales. Además, un algoritmo genético puede optimizar varios objetivos diferentes al mismo tiempo, y estos pueden ser añadidos con cierta facilidad mediante la introducción de nuevos factores a la función de evaluación de la aptitud de las soluciones candidatas. Cabe destacar también que su ejecución puede ser fácilmente paralelizable para la adecuada explotación de las hoy en día comunes arquitecturas multinúcleo y multiprocesador. Como principal inconveniente de los algoritmos genéticos destaca su coste en términos de tiempo de cómputo. Sin embargo, en los experimentos que presentamos al final de este artículo puede apreciarse que el mencionado coste es razonable y está justificado por la calidad de las soluciones que se alcanzan.

En las siguientes secciones se presentan algunos de los aspectos más relevantes relacionados con los algoritmos genéticos, como la codificación, las operaciones de creación de la población inicial, cruzamiento, mutación, agrupamiento, corrección, la función de aptitud y el flujo de ejecución del propio algoritmo genético.

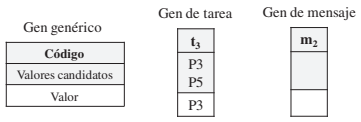

Figura 1: Diferentes tipos de genes.

t_1	t_5	t_7	t_2	t_3	t_6	t_4	m_1	m_3	m_4	m_2
P1	P1	P2	P2	P3 P5	P3 P5	P4 P5				
P1	P1	P2	P2	P3	P3	P4				

Figura 2: Cromosoma de ejemplo.

5.2. Codificación

Una solución candidata es un conjunto de valores de las variables del problema. En un algoritmo genético una variable se codifica mediante un elemento denominado gen y una solución candidata mediante una cadena de genes llamada cromosoma.

Normalmente, los genes se codifican mediante cadenas de símbolos binarios con valores cambiantes y posiciones fijas en el cromosoma. Sin embargo, existen otros tipos de codificaciones, tales como la permutacional (Davis, 1991), utilizada para solucionar problemas de ordenación de conjuntos de elementos. A diferencia de la codificación binaria, la permutacional está basada en un conjunto de genes con valores fijos y posiciones cambiantes en el cromosoma. Algoritmos genéticos basados en la codificación permutacional se han aplicado satisfactoriamente en la resolución de algunos problemas NP-complejos clásicos, como el *Travelling Salesman Problem* y el *Job-Shop Scheduling Problem* (Garey et al., 1976).

En este artículo se propone una nueva codificación híbrida valor-permutacional para representar las soluciones candidatas del problema de despliegue y planificación de sistemas de tiempo real distribuidos. Cada variable del problema tiene un gen asociado. Como puede verse en la figura 1, la codificación híbrida valor-permutacional se basa en un gen compuesto por tres campos: *Código*, *Valor* y *Valores candidatos*. *Código* es un campo fijo que almacena el nombre de la variable. Cada variable tiene un *Código* diferente. *Valor* es un campo potencialmente cambiante que representa el despliegue de la variable asociada al gen. Y *Valores candidatos* es un campo fijo que contiene la lista de valores que el campo *Valor* puede adquirir.

Algunos genes representan tareas y otros mensajes. En los genes de tareas el valor es uno de los computadores candidatos de la tarea. En los genes de mensajes el campo de valores candidatos y el de valor están vacíos. Adicionalmente, la posición relativa del gen en el cromosoma con respecto a otros genes con el mismo valor denota la prioridad o el orden de activación de la variable asociada. Es decir, cuanto más a la izquierda esté situado el gen, mayor es la prioridad de la variable asociada.

La figura 2 muestra un ejemplo de cromosoma. Como puede verse, el valor de t_1, t_2, t_5 y t_7 no puede cambiar porque sólo tienen un valor candidato, mientras que el de t_3, t_4 y t_6 sí puede hacerlo. Por otro lado, en el computador *P1*, la tarea t_1 está más a la izquierda que t_5 y por tanto tiene mayor prioridad. Y lo mismo ocurre en el resto de computadores. De igual forma, el mensaje m_1 tiene mayor prioridad que m_3, m_4 y m_2. En la solución candidata representada por este cromosoma no hay ninguna tarea desplegada en *P5*, dando lugar a la no utilización de este computador y a la consecuente minimización del número de computadores usados del máximo de 5 a 4.

Esta novedosa codificación híbrida valor-permutacional tiene ventajas destacables. En primer lugar, el valor del gen es siempre uno entre los valores candidatos, y por lo tanto, representa siempre un despliegue válido. Además, la posición del gen en el cromosoma es única, y por tanto, la prioridad o el orden de activación es siempre diferente a las del resto de variables con el mismo valor. Adicionalmente, la codificación híbrida valor-permutacional puede gestionar elementos heredados, como tareas desplegadas en computadores concretos y ejecutadas con prioridades ya asignadas, o mensajes con prioridades ya asignadas. En resumen, la codificación híbrida valor-permutacional constituye una representación eficiente de la configuración del despliegue y la planificación que soporta la búsqueda y optimización simultánea en ambas dimensiones, simplificando el proceso de diseño de sistemas de tiempo real distribuidos.

5.3. Población inicial

La población inicial es el conjunto de soluciones candidatas que el algoritmo genético evoluciona. Habitualmente, la población inicial se crea de forma aleatoria. Esta forma de creación garantiza una alta probabilidad de que las soluciones candidatas sean significativamente diferentes entre sí, lo que contribuye a evitar la convergencia del algoritmo genético hacia óptimos locales. No obstante, es también común emplear métodos heurísticos para crear uno, algunos o todos los individuos de la población inicial. La adecuada utilización de esta técnica puede proporcionar soluciones candidatas parcialmente evolucionadas, lo que acelera el proceso de búsqueda.

El uso de métodos heurísticos es especialmente recomendable en la síntesis de sistemas de tiempo real distribuidos *event-triggered* y analizados con alguna técnica holística. La complejidad de este tipo de métodos de análisis suele acarrear tiempos de análisis largos, especialmente si se trata de sistemas de tiempo real distribuidos grandes y complejos. Además, el tiempo de análisis es más largo cuanto peor es las planificación. Así, el análisis holístico de sistemas de tiempo real distribuidos grandes con planificaciones aleatorias, que habitualmente son mediocres, puede ser computacionalmente muy costoso. Por tanto, cuando el algoritmo genético se enfrenta a un sistema para cuyo análisis se usa algún tipo de método holístico, es conveniente realizar una planificación pseudoaleatoria preliminar de la población inicial de soluciones candidatas en lugar de partir de una planificación completamente aleatoria. La planificación pseudoaleatoria puede hacerse mediante el heurístico HOPA (Gutiérrez and Harbour, 1995) como se muestra en (Azketa et al., 2011b). Es destacable que para aplicar HOPA la utilización de los computadores y redes no puede exceder del 100 %. Por tanto, si la carga de alguno de los computadores o redes es mayor que 100 %, se debe aplicar un heurístico previo que

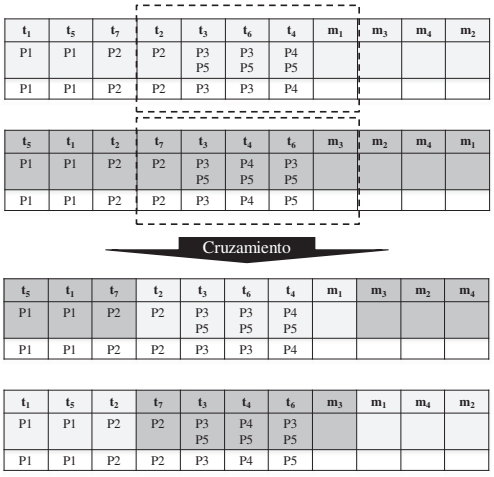

Figura 3: Operación de cruzamiento.

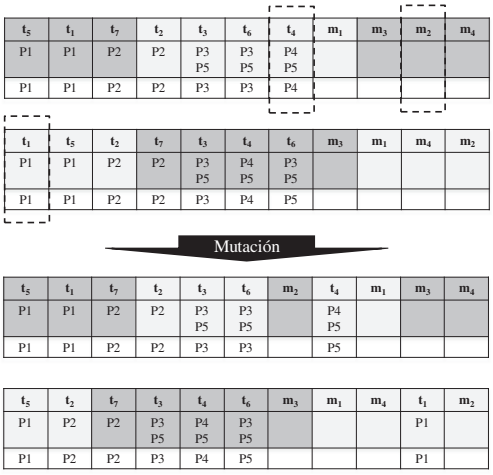

Figura 4: Operación de mutación.

mueva aleatoriamente tareas entre computadores hasta que la utilización de los computadores y redes no supere el 100 %.

Por el contrario, los métodos para analizar la planificabilidad de sistemas de tiempo real distribuidos *time-triggered* suelen ser más simples y requieren tiempos de ejecución más breves. Por lo tanto, cuando el algoritmo genético se enfrenta a un sistema de tiempo real distribuido *time-triggered*, es viable comenzar la evolución a partir de una población inicial de soluciones candidatas generadas de forma totalmente aleatoria.

5.4. Cruzamiento

Un operador genético de cruzamiento combina la información de dos cromosomas padres para crear dos cromosomas hijos que heredan parte de las soluciones de los padres.

En la literatura pueden encontrarse múltiples operadores de cruzamiento para algoritmos genéticos que emplean una codificación permutacional pura, los cuales son también válidos para la codificación híbrida valor-permutacional. En (Azketa et al., 2012b) se presenta un estudio empírico del rendimiento de los principales operadores de cruzamiento permutacionales en la asignación de prioridades en sistemas de tiempo real distribuidos. El trabajo concluye que los operadores con mejores resultados en la asignación de prioridades son *Position-based Crossover* (POS), *Order Crossover 1* (OX1) y *Cycle Crossover* (CX). Sin embargo, el presente problema incluye el despliegue de tareas además de la planificación. En este caso, algunos experimentos han revelado que *Order Crossover 3* (OX3) (Davis, 1991) obtiene los mejores resultados.

OX3 comienza seleccionando aleatoriamente dos puntos de corte en los cromosomas padre. El bloque de genes situado entre los dos puntos de corte del primer (segundo) cromosoma padre es directamente heredado por el primer (segundo) cromosoma hijo en la misma posición absoluta. El resto de genes se toman del otro padre en estricto orden de aparición y se insertan en el hijo comenzando desde el primer gen. OX3 explota la propiedad de que el orden de los genes es más importante que su posición absoluta. La figura 3 muestra la operación de cruzamiento con OX3 sobre dos posibles cromosomas de un escenario concreto y los cromosomas resultantes.

5.5. Mutación

La mutación es una operación que introduce cambios aleatorios o pseudoaleatorios en la información contenida en los genes para tratar de evitar que los individuos de una población se parezcan demasiado entre sí tras algunas generaciones (mantener la diversidad genética) o alternativamente también mejorar las soluciones aportadas por los individuos mediante cambios guiados (búsqueda local).

En este algoritmo genético se emplean operadores de mutación diferentes para cada uno de los dos tipos de información que un gen puede contener: posición, relacionado con la prioridad, y valor, relacionado con el despliegue. Para la mutación de posición se define un nuevo operador denominado *Guided Insertion Mutation* (GISM). Si existen flujos de extremo a extremo que no cumplen su plazo, de todas las tareas y mensajes de dichos flujos GISM selecciona aleatoriamente uno y lo desplaza un número aleatorio de posiciones hacia delante en el cromosoma, lo que potencialmente aumenta su prioridad. Si todos los flujos de extremo a extremo cumplen sus plazos, se aplica *Insertion Mutation* (ISM), que según (Azketa et al., 2012b) es eficaz en la asignación de prioridades en sistemas de tiempo real distribuidos. ISM selecciona un gen aleatoriamente y lo mueve a una posición aleatoria. Tanto GISM como ISM pueden aplicarse sobre los genes de tareas y mensajes.

En cuanto a la mutación de valor, se define *Guided Value Mutation* (GVM). Si existe un computador con sobreutilización de computación, selecciona aleatoriamente una tarea del computador con mayor sobreutilización y la despliega en su computador candidato con menor utilización y que tenga suficientes recursos para ejecutarla. Si no hay sobreutilización de computadores pero la red sufre sobreutilización de recursos de comunicación, selecciona aleatoriamente una tarea que envía un mensaje, selecciona aleatoriamente una tarea receptora de dicho mensaje y que tenga como candidato el computador donde está desplegada la tarea emisora del mensaje, y despliega

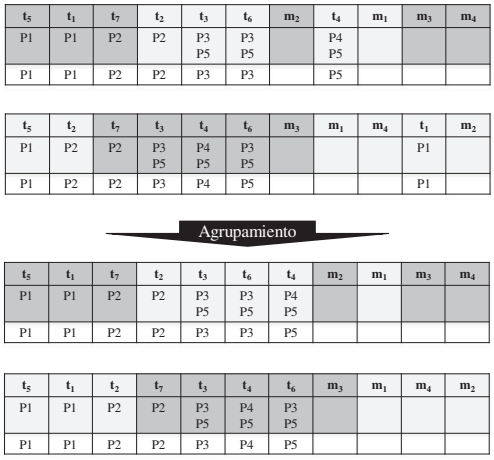

Figura 5: Operación de agrupamiento.

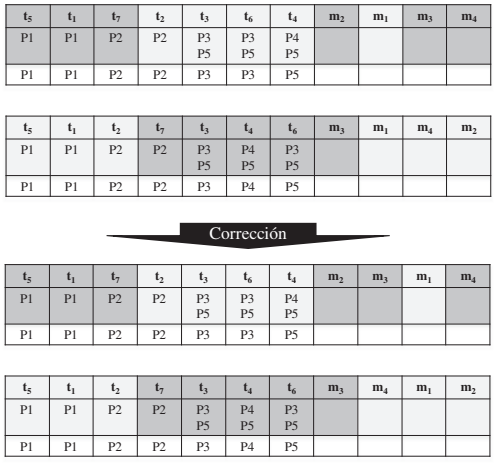

Figura 6: Operación de corrección.

la tarea receptora en el computador de la tarea emisora. Si no existe sobreutilización ni de computadores ni de la red, se selecciona aleatoriamente una tarea del computador prescindible con mayor utilización y se despliega en su computador candidato con menor utilización.

La figura 4 muestra la operación de mutación aplicada sobre los dos cromosomas resultantes del cruzamiento ilustrado en la figura 3. En el primer cromosoma, el gen t_4 sufre una mutación del tipo GVM y el gen m_2 sufre GISM. En el segundo cromosoma, al gen t_1 se le aplica ISM.

5.6. Agrupamiento

El agrupamiento se lleva a cabo con el objetivo de juntar los genes relacionados y crear así bloques de variables relacionadas. Esta operación incrementa la probabilidad del operador de cruzamiento de transmitir a los cromosomas hijos bloques de variables cuya relación hace que sea más probable que pertenezcan con los valores actuales a una solución del problema.

La operación de agrupamiento reordena los genes pero manteniendo el orden relativo de los mismos con respecto a otros genes con el mismo valor, ya que de lo contrario se alteraría la asignación de prioridad o el orden de activación llevado a cabo por el algoritmo. En primer lugar se agrupan los genes de las tareas desplegadas en el primer computador de la lista de computadores respetando su ordenación relativa. Tras esto, se lleva a cabo la misma operación con el siguiente computador de la lista, y así sucesivamente. Finalmente, se agrupan todos los mensajes manteniendo la ordenación relativa entre ellos. La figura 5 muestra la operación de agrupamiento ejecutada sobre ambos cromosomas.

5.7. Corrección

Cuando el algoritmo genético tiene que manejar tareas y mensajes con prioridades heredadas, las prioridades asignadas por el algoritmo a esas tareas o mensajes pueden ser incorrectas. En estos casos, los genes que representan tareas o mensajes con prioridades heredadas son movidos el mínimo número de posiciones para mantener la consistencia con sus prioridades heredadas.

Supongamos que en el sistema en el que se basan los cromosomas de la figura 5 el mensaje m_1 tiene una prioridad heredada por la cual solo un mensaje puede tener una prioridad inferior a él. Sin embargo, en el primer cromosoma resultante de la figura 5 puede observarse que m_3 y m_4 están situados más a la izquierda que m_1, y por tanto, tienen prioridades inferiores. En el segundo cromosoma ocurre esto mismo con m_4 y m_2. Como puede verse en la figura 6, la operación de corrección desplaza el mensaje m_1 una posición hacia la derecha en ambos cromosomas para guardar la consistencia con su prioridad heredada.

5.8. Aptitud

En un algoritmo genético la función de aptitud evalúa la calidad de las soluciones. Es decir, cómo de bien una solución candidata soluciona el problema. Una solución candidata es válida solo si cumple todas las restricciones. Por otra parte, a mayor grado de optimización, mejor es la solución candidata.

El despliegue y la planificación es un problema de búsqueda y optimización multiobjetivo porque comprende múltiples objetivos, como el aprovechamiento máximo de los recursos computacionales y de comunicaciones, el cumplimiento de los plazos de los flujos de extremo a extremo y la minimización del número de computadores usados, que tienen que ser abordados simultáneamente. Además, algunos de los objetivos mencionados tienen conflictos entre sí. Por ejemplo, la minimización del número de computadores usados puede causar la sobreutilización de los recursos computacionales de algunos computadores.

En problemas de búsqueda y optimización multiobjetivo, la calidad de una solución candidata puede representarse por medio de un vector que agrupa los factores que evalúan los diferentes objetivos. La calidad relativa entre pares de soluciones candidatas se cuantifica usando el concepto de *dominancia de Pareto*. Un vector multiobjetivo $u = (u_1, u_2, ..., u_k)$ domina a otro vector multiobjetivo $v = (v_1, v_2, ..., v_k)$ si y solo si u es mejor que v en un objetivo sin ser peor en todos los demás. Con-

siderando un problema de minimización, la notación formal de la dominancia de Pareto es

$$\forall i \in \{1,2,...,k\},\ u_i \leq v_i \bigwedge \exists i_0 \in \{1,2,...,k\} \mid u_{i_0} < v_{i_0}$$

Una solución candidata x^* es mejor que otra solución x si el vector multiobjetivo $u = f(x^*) = (u_1, u_2, ..., u_k)$ del primero domina al vector multiobjetivo $v = f(x) = (v_1, v_2, ..., v_k)$ del segundo. Además, una solución x^* es *Pareto-óptima* si y solo si no existe en todo el espacio de búsqueda ninguna otra solución x cuyo vector multiobjetivo $v = f(x) = (v_1, v_2, ..., v_k)$ domina el vector multiobjetivo $u = f(x^*) = (u_1, u_2, ..., u_k)$ del primero. La solución a un problema de búsqueda y optimización multiobjetivo es el conjunto de las soluciones candidatas no dominadas, denominado *frente de Pareto*.

En definitiva, en un algoritmo de búsqueda y optimización multiobjetivo, una solución es mejor que otra si la primera domina a la segunda. Dado que éste es el criterio para cuantificar la aptitud (capacidad de supervivencia) de una solución, se descartan las peores soluciones encontradas. Al final del proceso de búsqueda y optimización se obtiene un conjunto de soluciones en el que teóricamente ninguna es mejor que el resto, siendo responsabilidad del diseñador optar por la más adecuada.

El método de evaluación basado en vectores multiobjetivo permite añadir nuevos parámetros de búsqueda y optimización mediante la definición de sus correspondientes objetivos y su introducción directa en el vector. Por lo tanto, este método permite integrar diferentes objetivos de búsqueda y optimización y su extensión de forma sencilla.

El algoritmo genético propuesto la calidad de las soluciones candidatas está representada por un vector de tres objetivos minimizables, relacionados con la utilización de los recursos de procesamiento de los computadores y la red, el cumplimiento de plazos de los flujos de extremo a extremo y el número de computadores usados.

5.8.1. Objetivo de procesamiento

El objetivo de procesamiento O_C evalúa la utilización de los recursos computacionales y de comunicaciones de cada dispositivo de procesamiento. Dado que para ser válida una solución tiene que cumplir la restricción de no sobrepasar las utilizaciones de procesamiento máximas configuradas, el objetivo de procesamiento se considera un *objetivo de restricción*. No obstante, el objetivo de procesamiento puede ser optimizado en términos de minimización. La utilización de los recursos computacionales (comunicaciones) de las tareas (mensajes) desplegadas (transmitidos) en un computador (red) Ky es

$$U_{Ky} = \sum_{\forall \tau_h \ in \ Ky} \frac{C^{Ky}_{\tau_h}}{T_{\tau_h}}$$

donde τ_h es una tarea o mensaje y $C^{Ky}_{\tau_h}$ es su tiempo de ejecución o transmisión de peor caso en el computador o red Ky. El objetivo de procesamiento se calcula con la Ecuación (1).

$$O_C = \begin{cases} \sum_{\forall Ky} \dfrac{U_{Ky} - U^{max}_{Ky}}{|K|} & \forall Ky : U_{Ky} \leq U^{max}_{Ky} \\ \sum_{\forall Ky} max\left[0, \dfrac{U_{Ky} - U^{max}_{Ky}}{|K|}\right] & \exists Ky : U_{Ky} > U^{max}_{Ky} \end{cases} \quad (1)$$

Como puede observarse, O_C es la holgura relativa media de la utilización de procesamiento en los computadores y la red. Si $U_{Ky} > U^{max}_{Ky}$, se excede la utilización de procesamiento máxima configurada del dispositivo de procesamiento Ky. Si existe al menos un dispositivo de procesamiento cuya utilización máxima configurada es excedida, el objetivo de procesamiento es la suma de la holgura relativa de las utilizaciones de procesamiento de solo las que se exceden. Si $O_C > 0$, la solución candidata no cumple la restricción de la utilización máxima de procesamiento, por lo que es una solución candidata inválida.

Como se ha mencionado en la sección 3, el parámetro U^{max}_{Ky} puede configurarse para cada computador y red. Cuanto menor sea U^{max}_{Ky} la búsqueda de soluciones es más difícil, pero los recursos de procesamiento libres son mayores, y por tanto, la extensibilidad del sistema resultante también será mayor.

5.8.2. Objetivo de plazo

El objetivo de plazo O_D evalúa el cumplimiento de plazos de los flujos de extremo a extremo. Dado que para ser válida una solución tiene que cumplir los plazos de los flujos de extremo a extremo, el objetivo de plazo se considera un *objetivo de restricción*. No obstante, el objetivo de plazo puede ser optimizado en términos de minimización. El objetivo de plazo se calcula con la Ecuación (2).

$$O_D = \begin{cases} \sum_{\forall \Gamma_l} \dfrac{\frac{R_{\Gamma_l}}{D_{\Gamma_l}} - UR^{max}_{\Gamma_l}}{|\Gamma|} & \forall \Gamma_l : \dfrac{R_{\Gamma_l}}{D_{\Gamma_l}} \leq UR^{max}_{\Gamma_l} \\ \sum_{\forall \Gamma_l} max\left[0, \dfrac{\frac{R_{\Gamma_l}}{D_{\Gamma_l}} - UR^{max}_{\Gamma_l}}{|\Gamma|}\right] & \exists \Gamma_l : \dfrac{R_{\Gamma_l}}{D_{\Gamma_l}} > UR^{max}_{\Gamma_l} \end{cases} \quad (2)$$

Como puede observarse, O_D es la holgura relativa media de los tiempos de respuesta de peor caso con respecto a sus plazos. Si $R_{\Gamma_l}/D_{\Gamma_l} > UR^{max}_{\Gamma_l}$, el plazo del flujo de extremo a extremo Γ_l no se cumple. Si existe al menos un plazo que no se cumple, el objetivo de plazo es la suma de la holgura relativa de los tiempos de respuesta de peor caso de solo los que no cumplen plazos. Si $O_D > 0$, la solución candidata no cumple la restricción de plazo, por lo que es una solución candidata inválida. Cuanto menor es el objetivo de plazo, mayor es el rendimiento del sistema.

Como se ha mencionado en la sección 3, el parámetro $UR^{max}_{\Gamma_l}$ se puede configurar para cada flujo de extremo a extremo. Cuanto menor sea $UR^{max}_{\Gamma_l}$ la búsqueda de soluciones es más difícil, pero la holgura temporal de los flujos de extremo a extremo es mayor, y por tanto, la extensibilidad del sistema resultante también será mayor.

5.8.3. Objetivo de computadores

El objetivo de computadores O_P evalúa el número de computadores usados para desplegar las tareas. Una solución puede ser válida independientemente del número de computadores usados. Sin embargo, a menor número de computadores utilizados, mejor es la solución. Por lo tanto, el objetivo de computadores es considerado un *objetivo de optimización*. El objetivo de computadores se calcula con la Ecuación (3).

$$O_P = \sum_{\forall Px \in P_{dis}} \frac{w_1 \cdot \left\lceil \frac{|t_{Px}|}{|t|} \right\rceil + w_2 \cdot \frac{|t_{Px}|}{|t|}}{|P_{dis}|} \quad (3)$$

La Ecuación (3) incrementa el valor del objetivo con el número de computadores prescindibles usados y con el número de tareas en los computadores prescindibles usados. Como puede verse, el numerador del cociente es la suma ponderada de dos factores. El primer factor es 0 cuando no hay ninguna tarea desplegada en el computador correspondiente, mientras que de lo contrario es 1, incrementándose el valor del objetivo. El segundo factor incrementa el valor del objetivo con el número de tareas. La suma de los pesos tiene que ser 1, siendo $w_1 = 0{,}8$ y $w_2 = 0{,}2$ unos valores adecuados. Si $O_P = 0$, no se utiliza ningún computador prescindible, por lo que el objetivo de computadores alcanza su valor óptimo. Cuanto menor es el número de computadores usados mejor es la métrica de costo.

5.9. Objetivo total

El objetivo total O_T es el vector multiobjetivo que combina los tres objetivos parciales, como puede verse en la Ecuación 4.

$$O_T = (O_C, O_D, O_P) \quad (4)$$

El proceso de búsqueda trata de minimizar el objetivo total. Una solución candidata es válida si $O_C \leq 0$ y $O_D \leq 0$. Además, cuanto menores sean los objetivos parciales mejor es la solución candidata. El objetivo total O_T se usa para calcular las relaciones de dominancia entre las diferentes soluciones candidatas, siendo la aptitud de una solución inversamente proporcional al número de soluciones que la dominan.

5.10. Algoritmo

El algoritmo está basado en *Non-Dominated Sort Genetic Algorithm II* (NSGA-II) (Deb et al., 2002) pero presenta algunas diferencias que mejoran su rendimiento. Así, la ordenación secundaria basada en el parecido entre soluciones (*crowding*) se sustituye por una ordenación basada en el cumplimiento de los objetivos de restricción. Además, se añade una ordenación terciaria basada en los objetivos de optimización. A continuación se muestra el flujo del algoritmo propuesto:

1. Se crea una población inicial de N soluciones candidatas.
2. Se calculan los objetivos de las soluciones candidatas.
3. Se calcula la *no dominancia* de las soluciones candidatas, que obtiene el número de soluciones candidatas que dominan a cada solución candidata.
4. Se calculan la *no dominancia de restricción* y la *no dominancia de optimización* de las soluciones candidatas. La no dominancia de restricción es la no dominancia calculada teniendo en cuenta sólo los objetivos de restricción, que en este caso son el de procesamiento O_C y el de plazo O_D. La no dominancia de optimización es la no dominancia calculada teniendo en cuenta sólo los objetivos de optimización, que en este caso es el de computadores O_P.
5. La población se ordena en orden ascendente según la no dominancia. Dentro que cada nivel de no dominancia, las soluciones se ordenan en orden ascendente de no dominancia de restricción, y dentro que cada nivel de no dominancia de restricción, las soluciones se ordenan en orden ascendente de no dominancia de optimización. De esta forma, en primer lugar se sitúan las soluciones dominadas por 0 soluciones, después las dominadas por 1 solución, y así sucesivamente. Dentro de cada nivel de no dominancia, en primer lugar se sitúan las soluciones cuyos objetivos de restricción son dominados por 0 soluciones, después las soluciones cuyos objetivos de restricción son dominados por 1 solución, y así sucesivamente. Y dentro de cada nivel de no dominancia de restricción, en primer lugar se sitúan las soluciones cuyos objetivos de optimización son dominados por 0 soluciones, después las soluciones cuyos objetivos de optimización son dominados por 1 solución, y así sucesivamente.
6. Se asigna una aptitud a cada solución candidata de forma inversamente proporcional a la posición en la ordenación efectuada en el paso 5. Así, la solución candidata en la primera posición recibe la mayor aptitud, mientras que la situada en la última posición tiene la menor aptitud.
7. Se seleccionan pares de soluciones candidatas. A mayor aptitud, mayor es la probabilidad de ser seleccionado. Esos pares de soluciones candidatas se cruzan para crear N nuevas soluciones candidatas que forman una nueva población, que es unida con la anterior población.
8. Se ejecutan los pasos 2 y 3.
9. Se agrupan las soluciones candidatas en *frentes no dominados*: las soluciones candidatas dominadas por 0 soluciones se insertan en una primera lista, las soluciones candidatas dominadas por 1 solución se insertan en una segunda lista, y así sucesivamente.
10. A partir de la primera, se van cogiendo las listas hasta que el número de soluciones candidatas es igual o mayor que N. Las soluciones de las listas seleccionadas forman la nueva población y las listas no seleccionadas se eliminan.
11. Se ejecutan los pasos 4, 5 y 6.
12. Se cogen las primeras (mejores) N soluciones candidatas de la nueva población, y el resto son eliminadas.
13. Se replica la población. A cada solución candidata de la población replicada se le aplica mutación de posición, y se calculan los objetivos de todas las soluciones candidatas mutadas. Cada solución candidata mutada se compara con su correspondiente solución candidata original no mutada en términos de no dominancia primero, no dominancia de restricción después, y no dominancia de op-

timización finalmente. Se selecciona la mejor solución candidata de las dos y se inserta en la nueva población.

14. Sobre esta nueva población se ejecuta el mismo proceso del paso 13, pero aplicando mutación de valor en vez de mutación de posición.
15. Este proceso se repite a partir del paso 2 hasta que se cumple la condición de parada.

6. Evaluación

Esta sección presenta una evaluación comparativa del algoritmo genético permutacional y MILP. La principal razón para optar por MILP es que se ha aplicado satisfactoriamente para solucionar el problema del despliegue y la planificación de sistemas de tiempo real distribuidos similares a los basados en el modelo descrito en la sección 3. Por otra parte, MILP posee características teóricas muy interesantes, como la obtención garantizada de soluciones óptimas y la determinación de la distancia entre una solución cualquiera y la óptima. Además, se basa en la utilización de lenguajes formales para crear modelos matemáticos del problema, que pueden ser solucionados mediante diferentes algoritmos implementados en multitud de herramientas como por ejemplo SCIP (Achterberg, 2009).

A diferencia de lo que ocurre en otras áreas de investigación relacionadas con sistemas de tiempo real y en las que se aplican métodos de búsqueda y optimización, como por ejemplo el diseño de *systems-on-chip*, en el campo de los sistemas de tiempo real distribuidos no existen *benchmarks* reconocidos con los que evaluar de forma normalizada las técnicas de diseño propuestas. Por esta razón, la evaluación de la mayoría de los trabajos previos que han abordado este tema se basa en escenarios *ad-hoc* generados aleatoriamente. Sin embargo, los escenarios creados de forma aleatoria suelen presentar diferencias notables con respecto a los sistemas de tiempo real distribuidos implementados en aplicaciones reales. Por ello, consideramos más interesante realizar la evaluación sobre un sistema industrial real. Concretamente, tenemos acceso a la descripción de un sistema de tiempo real complejo de un automóvil, pero por razones de confidencialidad no podemos ofrecer una descripción cualitativa detallada. No obstante, sí podemos presentar una descripción cuantitativa del mismo.

El *Escenario1* (*Esc1*) tiene una arquitectura física compuesta por 32 computadores conectados a una red CAN con una velocidad de transmisión de 500 kb/s. La arquitectura lógica está compuesta por 82 tareas con tiempos de ejecución de peor caso de 1 ms y 270 mensajes cuyas longitudes se sitúan entre 16 y 64 bits. Los periodos de las tareas y mensajes están entre 8 ms y 1500 ms, siendo 10 ms, 50 ms y 100 ms los periodos más comunes. Muchas tareas reciben más de un mensaje, y muchos mensajes se transmiten a más de una tarea, por lo que el grafo de tareas que se crea es complejo. En este grafo hay 10 flujos de extremo a extremo compuestos por entre 7 y 11 tareas y mensajes. Algunos flujos de extremo a extremo empiezan y/o terminan en las mismas tareas. Los plazos de estos flujos oscilan entre 100 ms y 300 ms. Además, cada tarea y mensaje también es un flujo de extremo a extremo con un plazo igual a su periodo. Así, en total hay $10 + 82 + 270 = 362$ flujos. Las tareas situadas al inicio y/o al final de los flujos de extremo a extremo con más de una tarea solo tienen un computador candidato, mientras que el resto de tareas pueden ser desplegadas en cualquier computador. Este sistema esta basado en un modelo de activación mixto *time-triggered* y *event-triggered* planificado por prioridades fijas, y la técnica empleada para analizar su planificabilidad es la presentada en (Di Natale et al., 2007).

El algoritmo genético ha sido configurado con una población de 100 individuos. El método de selección es el torneo de dos individuos, consistente en optar por el mejor de dos individuos seleccionados aleatoriamente. El operador de cruzamiento es OX3 aplicado con una probabilidad de 1. Los operadores de mutación son GISM, ISM e GVM aplicados una vez sobre el cromosoma con una probabilidad de 1. El algoritmo genético permutacional se ejecuta múltiples veces durante 1500 segundos y se obtienen valores promedio de los objetivos y del número de computadores durante la evolución de la búsqueda y la optimización. La evaluación se ha llevado a cabo en un ordenador con procesador Intel i5 a 3.20 GHz, 16 GB de memoria RAM y Windows 7 Professional de 64 bits.

La experimentación ha revelado que el sistema *Esc1* es demasiado grande y complejo como para utilizar SCIP. El número inicial de variables y restricciones es 2763697 y 35738633 respectivamente, y durante el proceso de solución el enorme tamaño del problema hace que SCIP llegue a requerir más de 16 GB de memoria, lo que produce que el sistema operativo aborte la ejecución. Por el contrario, el algoritmo genético permutacional presenta un comportamiento muy satisfactorio. Esta afirmación puede ser corroborada mediante la figura 7, que muestra la evolución con respecto al tiempo (eje X) de los valores de los objetivos de las soluciones de *Esc1* encontradas por el algoritmo genético. Los valores de los objetivos corresponden al eje Y de la izquierda, mientras que el número de computadores está reflejado en el eje Y de la derecha.

Como puede observarse en la figura 7, el objetivo de plazo comienza en valores máximos. Esto es debido a que en alguna o varias de las ejecuciones el objetivo de plazo tiene valores máximos a causa de la sobreutilización de los recursos de algún computador y/o de la red. En este caso todas las mejores soluciones iniciales presentan sobreutilización de los recursos de comunicación de la red. En 280 segundos el objetivo de plazo deja de tener el valor máximo, lo que denota que en las soluciones promedio obtenidas en ese instante ningún computador ni la red sufren sobreutilización de recursos. Esto implica que en todas las ejecuciones se cumple la restricción del objetivo de procesamiento. Entre los segundos 280 y 430, el objetivo de plazo tiene valores positivos, por lo que la solución promedio no es planificable. A partir de 430 segundos, se cumplen las restricciones de procesamiento y plazo (tienen valores no positivos), por lo que las soluciones promedio obtenidas son válidas. Durante toda la evolución, el objetivo de computadores y el número de computadores muestran una tendencia descendente. En el instante en que se obtiene la primera solución válida (430 segundos), el número promedio de computadores está en torno a 16, pero en 1000 segundos esta cantidad se estabiliza en torno a poco más de 12 computadores.

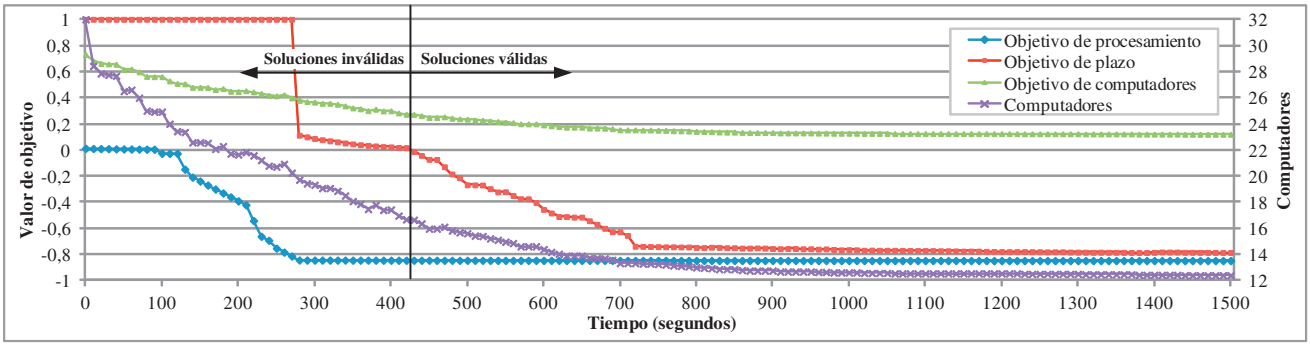

Figura 7: Evolución de las soluciones del algoritmo genético en *Esc1*.

Con el objetivo de obtener resultados de MILP, se ha definido el *Escenario 2* (*Esc2*), basado en un subconjunto de *Esc1*. *Esc2* está compuesto por 15 computadores, 26 tareas, 25 mensajes y 10 flujos de extremo a extremo, todos con las mismas características que los de *Esc1*.

Los problemas modelados como MILP se suelen someter a un preprocesado denominado *presolving* que detecta conflictos entre restricciones y elimina las variables y restricciones redundantes o prescindibles. Antes del *presolving*, el problema del despliegue y la planificación de *Esc2* tiene 36920 variables y 285216 restricciones, mientras que tras este proceso inicial que ha tardado 120 segundos el problema queda con 9770 variables y 84055 restricciones. La solución óptima, que tiene 8 computadores, se ha obtenido tras 4239 segundos de ejecución.

La evolución de las soluciones encontradas por el algoritmo genético permutacional en *Esc2* se muestran en la figura 8. Como puede observarse, el objetivo de procesamiento permanece invariable con valores negativos, y por tanto válidos, durante todo el proceso. El objetivo de plazo comienza con valores promedio negativos cercanos al cero, lo que puede ser reflejo de que en algunas ejecuciones el mejor valor inicial es planificable mientras que en la mayoría de ellas no lo es. En cualquier caso, la planificabilidad del sistema comienza a mejorar en seguida, tal y como muestra la línea descendente del objetivo de plazo. Finalmente, el objetivo de computadores comienza en un valor de 0,6 pero tarda tan solo 20 segundos en alcanzar un valor de 0, que es el óptimo. El número de computadores comienza en 15, pero alcanza el óptimo de 8 en 20 segundos, lo que supone un tiempo 2 órdenes de magnitud menor que el de MILP.

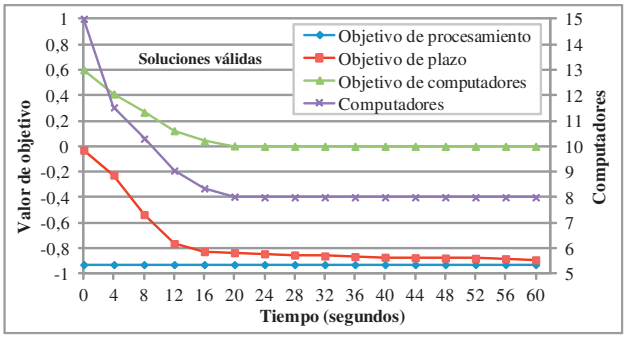

Figura 8: Evolución de las soluciones del algoritmo genético en *Esc2*.

7. Conclusiones y trabajo futuro

En este artículo se propone un algoritmo genético multiobjetivo basado en una codificación permutacional de las soluciones para abordar los problemas NP-difíciles del despliegue y la planificación de sistemas de tiempo real distribuidos. El modelo de sistema y algoritmo propuestos presentan un enfoque de estos problemas más detallado que los trabajos de la bibliografía existente.

La evaluación del algoritmo propuesto frente a MILP muestra la inviabilidad de aplicar este último método para abordar el diseño de sistemas complejos, mientras que el algoritmo genético permutacional obtiene soluciones satisfactorias en tiempos razonables.

Como trabajo futuro planteamos integrar el diseño de la topología de red. Esto requeriría nuevos genes relacionados con la topología y la definición de un nuevo objetivo que evaluara la calidad de este parámetro. El formato de los genes tiene suficiente genericidad como para permitir la definición de los genes relacionados con la topología, y el método de evaluación basado en vectores multiobjetivo permitiría agregar de forma sencilla un nuevo objetivo relacionado con la topología de red.

English Summary

Permutational genetic algorithm for the deployment and scheduling of distributed real time systems

Abstract

The deployment and scheduling of tasks and messages in distributed real-time systems are NP-hard problems, so there are no optimal methods to solve them in polynomial time. Consequently, these problems are suitable to be approached with generic search and optimisation algorithms. In this paper we propose a multi-objective genetic algorithm based on a permutational solution encoding for the deployment and scheduling

of distributed real-time systems. Besides deploying and scheduling tasks and messages, the algorithm can minimize the number of the used computers, the utilization of computing and networking resources and the average worst-case response times of the applications. The experiments show that this genetic algorithm can successfully synthesize complex distributed real-time systems in reasonable times.

Keywords: Real-time systems, Scheduling algorithms, Genetic algorithms, Multiobjective optimisations.

Referencias

Achterberg, T., 2009. SCIP: Solving Constraint Integer Programs. Mathematical Programming Computation 1 (1), 1–41.

Azketa, E., Gutiérrez, J. J., Palencia, J. C., Harbour, M. G., Almeida, L., Marcos, M., 2012a. Schedulability analysis of multi-packet messages in segmented CAN. In: Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation.

Azketa, E., Uribe, J., Marcos, M., Almeida, L., Gutiérrez, J., 2011a. Permutational genetic algorithm for fixed priority scheduling of distributed real-time systems aided by network segmentation. In: Proceedings of the 1st Workshop on Synthesis and Optimization Methods for Real-time Embedded Systems. pp. 13–18.

Azketa, E., Uribe, J., Marcos, M., Almeida, L., Gutiérrez, J., 2011b. Permutational genetic algorithm for the optimized assignment of priorities to tasks and messages in distributed real-time systems. In: Proceedings of the 8th IEEE International Conference on Embedded Software and Systems. pp. 958–965.

Azketa, E., Uribe, J. P., Marcos, M., Almeida, L., Gutiérrez, J. J., 2012b. An empirical study of permutational genetic crossover and mutation operators on the fixed priority assignment in distributed real-time systems. In: Proceedings of the IEEE International Conference on Industrial Technology. pp. 598–605.

Boyd, S., Kim, S., Vandenberghe, L., Hassibi, A., 2007. A tutorial on geometric programming. Optimization and Engineering 8 (1), 67–127.

Chen, W., Lin, C., 2000. A hybrid heuristic to solve a task allocation problem. Computers & Operations Research 27 (3), 287–303.

Davare, A., Zhu, Q., Di Natale, M., Pinello, C., Kanajan, S., Sangiovanni-Vincentelli, A., 2007. Period optimization for hard real-time distributed automotive systems. In: Proceedings of the 44th Annual Design Automation Conference. pp. 278–283.

Davis, L., 1991. Handbook of genetic algorithms. Arden Shakespeare.

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6 (2), 182–197.

Di Natale, M., Stankovic, J. A., 1995. Applicability of simulated annealing methods to real-time scheduling and jitter control. In: Proceedings of the 16th IEEE Real-Time Systems Symposium. pp. 190–199.

Di Natale, M., Zheng, W., Pinello, C., Giusto, P., Sangiovanni-Vincentelli, A., 2007. Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems. In: Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium. pp. 293–302.

Dick, R., Jha, N., 2002. MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17 (10), 920–935.

Garey, M., Johnson, D., Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research, 117–129.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13 (5), 533–549.

Gutiérrez, J. J., Harbour, M. G., 1995. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In: Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems. pp. 124–132.

Hamann, A., Jersak, M., Richter, K., Ernst, R., 2006. A framework for modular analysis and exploration of heterogeneous embedded systems. Real-Time Systems 33 (1), 101–137.

Hladik, P., Cambazard, H., Déplanche, A., Jussien, N., 2008. Solving a real-time allocation problem with constraint programming. Journal of Systems and Software 81 (1), 132–149.

Holland, J., 1975. Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor.

Kirkpatrick, S., 1984. Optimization by simulated annealing: Quantitative studies. Journal of Statistical Physics 34 (5), 975–986.

Metzner, A., Herde, C., 2006. RTSAT - An optimal and efficient approach to the task allocation problem in distributed architectures. In: Proceedings of the 27th IEEE Real-Time Systems Symposium. pp. 147–158.

Minoux, M., 1986. Mathematical programming: theory and algorithms. John Wiley & Sons.

Mitra, H., Ramanathan, P., 1993. A genetic approach for scheduling non-preemptive tasks with precedence and deadline constraints. In: Proceedings of the Hawaii International Conference on System Sciences. Vol. 26. pp. 556–556.

Monnier, Y., Beauvais, J., Deplanche, A., 1998. A genetic algorithm for scheduling tasks in a real-time distributed system. In: Proceedings of the 24th Euromicro Conference. Vol. 2. pp. 708–714.

Palencia, J., Harbour, M., 1998. Schedulability analysis for tasks with static and dynamic offsets. In: Proceedings of the 19th IEEE Real-Time Systems Symposium. pp. 26–37.

Palencia, J. C., Harbour, M. G., 1999. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In: Proceedings of the 20th IEEE Real-Time Systems Symposium. pp. 328–339.

Pearl, J., 1984. Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley.

Pop, P., Eles, P., Peng, Z., 2002. Flexibility driven scheduling and mapping for distributed real-time systems. In: Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications. pp. 337–346.

Pop, P., Eles, P., Peng, Z., 2003a. Schedulability analysis and optimization for the synthesis of multi-cluster distributed embedded systems. In: Proceedings of the Conference on Design, Automation and Test in Europe. Vol. 1. pp. 184–189.

Pop, T., Eles, P., Peng, Z., 2003b. Design optimization of mixed time/event-triggered distributed embedded systems. In: Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. pp. 83–89.

Porto, S., Kitajima, J., Ribeiro, C., 2000. Performance evaluation of a parallel tabu search task scheduling algorithm. Parallel Computing 26 (1), 73–90.

Porto, S., Ribeiro, C., 1995. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. International Journal of High Speed Computing 7 (1), 45–71.

Samii, S., Yin, Y., Peng, Z., Eles, P., Zhang, Y., 2009. Immune genetic algorithms for optimization of task priorities and flexray frame identifiers. In: Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications. pp. 486–493.

Schrijver, A., 1998. Theory of linear and integer programming. John Wiley & Sons Inc.

Shang, L., Dick, R., Jha, N., 2007. SLOPES: Hardware-software cosynthesis of low-power real-time distributed embedded systems with dynamically reconfigurable FPGAs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26 (3), 508–526.

Tindell, K., Burns, A., Wellings, A., 1992. Allocating hard real-time tasks: an NP-hard problem made easy. Real-Time Systems 4 (2), 145–165.

Tindell, K., Clark, J., 1994. Holistic schedulability analysis for distributed hard real-time systems. Microprocessing and Microprogramming 40 (2-3), 117–134.

Tsang, E., 1993. Foundations of constraint satisfaction. Vol. 289. Academic Press London.

Zheng, W., Zhu, Q., Di Natale, M., Sangiovanni-Vincentelli, A., 2007. Definition of task allocation and priority assignment in hard real-time distributed systems. In: Proceedings of the 28th IEEE Real-Time Systems Symposium. pp. 161–170.

Zhu, Q., Yang, Y., Scholte, E., Di Natale, M., Sangiovanni-Vincentelli, A., 2009. Optimizing extensibility in hard real-time distributed systems. In: Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium. pp. 275–284.