Algoritmo evolutivo para el problema de planificación en proyectos de desarrollo de software

Germán Dupuy - Natalia Stark - Carolina Salto

Facultad de Ingeniería - Universidad Nacional de La Pampa
Argentina
germandupuy24@gmail.com - {nstark, saltoc}@ing.unlpam.edu.ar}

Resumen La planificación de tareas y la asignación de recursos en proyectos de desarrollo de mediana a larga escala es un problema extremadamente complejo y es uno de los principales desafíos de la gestión del proyecto, debido a su complejidad. El objetivo es minimizar la duración y el costo del proyecto. En este trabajo proponemos un algoritmo genético (AG) tradicional usando codificación binaria para representar una solución al problema de planificación de proyectos software. En particular nos centramos en la elección del operador de cruce, junto con su probabilidad; proponemos comparar el cambio en el rendimiento del AG al utilizar operadores genéticos tradicionales respecto de otros más específicos para el problema. Los experimentos mostraron que utilizar una recombinación tradicional es capaz de aumentar el rendimiento del algoritmo, manteniendo en niveles aceptables la velocidad de convergencia.

Keywords: proyectos de planificación software, algoritmos genéticos, cruce, probabilidades

1. Introducción

En la actualidad, la gestión de los proyectos de desarrollo de software requiere de la programación temporal, la planificación y la monitorización de tareas, asignando recursos de manera eficiente para conseguir objetivos específicos, cumpliendo con un conjunto de restricciones. En líneas generales, la planificación de proyectos software [1] consiste en determinar quién debe hacer qué, cuándo, con qué recursos y en qué momento. Las tareas pueden ser muy diversas: desde mantener documentos hasta escribir programas. Por recursos entendemos personal con sus habilidades y el tiempo. Los objetivos que se persiguen es minimizar la duración y el costo. Además de cumplir con dichos objetivos se debe asegurar una calidad mínima del producto [2].

Los objetivos en conflicto y las restricciones hacen de éste un problema de optimización NP-hard [11], para el cual no existen soluciones algorítmicas polinomiales conocidas, razón por la cual, las aplicaciones de apoyo disponibles en el mercado hacen un seguimiento pasivo del proyecto. Dado que para el administrador del proyecto sería de gran utilidad disponer de una herramienta que le

facilite obtener planificaciones de forma automática que concilien los objetivos en conflicto a nivel de proyecto, muchos investigadores recurrieron a metaheurísticas, en particular algoritmos genéticos (AGs), para resolver el problema [1,2,3,9]. El trabajo de Alba y Chicano [1] es probablemente el trabajo más conocido y representa el estado del arte en la resolución del problema usando AGs.

En este trabajo nos centramos en la resolución del problema de planificación de proyecto de software (PSP) usando algoritmos genéticos [8]. Sin embargo, es importante resaltar, que en algoritmos genéticos en general la elección de un operador de cruce y su probabilidad de aplicación constituyen aspectos críticos en el diseño del algoritmo [10], debido a que el uso de parametrizaciones inadecuadas frecuentemente producen una reducción importante en el rendimiento del algoritmo [12]. Esto se debe en general a que los espacios de búsqueda resultantes presentan características indeseables, como muchos óptimos locales y múltiples regiones factibles desconectadas, que hacen que el proceso de optimización resulte considerablemente más difícil.

El algoritmo genético propuesto para resolver el problema utiliza cromosomas binarios para representar soluciones al PSP. Estas soluciones son matrices que codifican la dedicación de cada empleado para realizar una determinada tarea. Por simplicidad en la implementación, estas matrices se traducen en cadenas binarias que manipula el algoritmo genético. Usar esta traducción permite aplicar distintos tipos de operadores de recombinación: operadores de cruce tradicionales (un punto, dos puntos, uniforme, entre otros) que actuarían directamente sobre la cadena binaria que representa la matriz solución o bien aquellos diseñados para la recombinación de matrices. Por lo tanto, el objetivo que se persigue en este trabajo consiste en determinar cuál operador genético se adapta mejor en la resolución del problema en cuestión, como así también determinar cuál es la probabilidad de cruce más adecuada, otro aspecto en conflicto relacionado con el operador de cruce. Hemos considerado un conjunto amplio de instancias, variando los distintos parámetros del algoritmo, para evitar sesgar las conclusiones y evitar la posibilidad de "hand-tuning" del algoritmo para una instancia particular del problema.

Este trabajo está organizado como sigue. En la Sección 2 se define el Problema de Planificación de Proyectos. En la sección 3 se trata la aplicación del AG al problema en cuestión, luego en la sección 4 se describe la parametrización utilizada, seguidamente se muestran los resultados en la sección 5 y por último, en la sección 6 se presentan algunas conclusiones y trabajos futuros.

2. Definición del Problema

El problema de planificación de proyectos (PSP, Project Scheduling Problem) plantea la necesidad de un procedimiento para asignar recursos limitados a un conjunto de tareas en un cierto plazo. Se está frente a un proceso de toma de decisiones que tiene como meta la optimización de objetivos que eventualmente pueden estar en conflicto entre sí. Los recursos gestionados son personas con un

conjunto de habilidades, un salario y un grado máximo de dedicación al proyecto. Los objetivos son, normalmente, minimizar la duración y el costo del proyecto.

La formulación del problema es como sigue [4]. Se tiene un conjunto de E empleados y un conjunto de T tareas con ciertas habilidades requeridas, además de un grafo de precedencia de tareas (TPG). Cada empleado posee un conjunto de habilidades, recibe un salario mensual y tiene un grado máximo de dedicación al proyecto (cociente entre la cantidad de horas dedicadas al proyecto y la cantidad de horas de una jornada laboral completa). Cada tarea está caracterizada por un conjunto de habilidades asociadas necesarias para poder ser llevadas a cabo y un esfuerzo expresado en personas-mes. Las tareas se deben completar según el orden establecido por el grafo de precedencia de tareas, el cual es un grafo dirigido que indica qué tareas deben completarse antes de iniciar otra.

El objetivo del problema es, entonces, asignar empleados a tareas, minimizando tanto el costo como la duración del proyecto. Se debe considerar que a cada tarea la realiza al menos una persona, las habilidades de los empleados que realizan cierta tarea deben cubrir el conjunto de habilidades requeridas por esa tarea y, por último, la dedicación de un empleado al proyecto no debe exceder su dedicación máxima. Si alguna de estas restricciones no se cumplen, se considera que la solución al problema no es factible. Por lo tanto, una solución al problema se puede representar por medio de una matriz $X = (x_{ik})$ de tamaño $E \times T$ donde $x_{ik} \geq 0$. El elemento x_{ik} es el grado de dedicación del i-esimo empleado a la k-esima tarea.

La calidad de una solución depende de tres factores: la duración y el costo del proyecto y la factibilidad de la solución. La duración de cada tarea (tkdur) es el cociente entre el esfuerzo para realizar la tarea y la sumatoria de los grados de dedicación de cada empleado a esa tarea. Obtenida la duración de cada tarea y teniendo en cuenta el TPG, se calcula el tiempo de inicio y fin de cada una, lo que da lugar a conocer la duración del proyecto, denotada por *tiempo*. El costo (*costo*) es el producto entre el salario del empleado y el tiempo de dedicación al proyecto; este último es la suma de la dedicación a cada tarea del empleado multiplicada por la duración de la misma.

3. Algoritmo Genético para el PSP

En este trabajo se utiliza un AG tradicional con representación binaria y reemplazo generacional, a diferencia de las propuestas de Alba y Chicano [1] que utilizan un AG de estado estacionario y de Minku et. al [9] que trabajan con una (1+1)-AG. La solución al problema se representa por una matriz X, donde el elemento x_{ij} indica el grado de dedicación del i-ésimo empleado a la j-ésima tarea. Se considera que el grado de dedicación máxima de cada empleado es igual a 1, ya que se supone que ningún empleado trabaja horas extras, es decir $x_{ij} \in [0, 1]$. Como la dedicación de cada empleado x_{ij} es un valor real, el intervalo de dedicación se discretiza en ocho valores uniformemente distribuidos, tomando para ello tres bits [1]. Como técnica de implementación y por simplicidad, se adoptó que la matriz solución X se represente como una cadena binaria en una

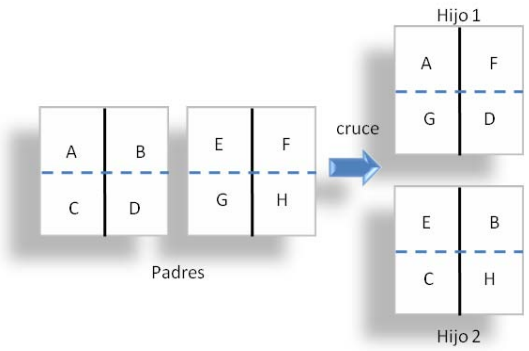

Figura 1. Funcionamiento del cruce de matrices

dimensión (x), es decir se realiza un procedimiento de aplanamiento de la matriz. En consecuencia, la dimensión de la cadena binaria x es igual a $E \times T \times 3$. La función de fitness que usa el AG para medir qué tan buena es una solución es la siguiente:

$$f(x) = \begin{cases} 1/q & \text{si la solución es factible} \\ 1/(q+p) & \text{en caso contrario} \end{cases} \quad (1)$$

donde $q = w_{costo} \times costo + w_{tiempo} \times tiempo$, $p = w_{penal} + w_{t\_inc} \times t\_inc + w_{h\_nocub} \times h\_nocub + w_{extras} \times extras$ y w_{costo}, w_{tiempo}, w_{penal}, w_{t_inc}, w_{h_nocub} y w_{extras} son parámetros predefinidos, *costo* y *tiempo* son el costo y el tiempo de finalización del proyecto asociados a la solución, *t_inc* es la cantidad de tareas que no tienen empleados asociados, *h_nocub* es la cantidad de habilidades requeridas no cubiertas para realizar todas las tareas del proyecto, y *extras* es el tiempo total de horas extras de todos los empleados durante la ejecución del proyecto. Como los dos términos q y p aparecen en el denominador y el objetivo es minimizarlos, entonces $f(x)$ se debe maximizar.

Un punto crucial en el diseño de un algoritmos genético es determinar un operador de cruce adecuado al problema. Dado que algunos son más disruptivos que otros, el tipo de cruce usado puede dirigir una búsqueda demasiado exhaustiva sobre una región o por el contrario hacer una gran exploración sobre el espacio de búsqueda. Como el AG propuesto en este trabajo maneja cromosomas representados como cadenas binarias, se propone utilizar el tradicional cruce de un punto para arreglos binarios (1X) introducido por Holland [7].

Por otra parte, como la solución al problema es una matriz, se decidió analizar el comportamiento del algoritmo utilizando un cruce diseñado especialmente para recombinar matrices, conocido como cruce 2D de un punto (2DMX) [13]. Luego de transformar la cadena binaria x en la matriz X, el cruce selecciona una fila y una columna (la misma en los dos padres) en forma aleatoria y luego intercambia los elementos en el cuadrante superior izquierdo y en el cuadrante

inferior derecho de ambos individuos. La Figura 1 muestra en forma gráfica el funcionamiento del cruce.

Analizando el grado de ruptura que podrían provocar ambos operadores, es de esperar que el operador 1X resulte con un menor grado de ruptura de la información (en este caso planificación) contenida en las soluciones padres ya que podría pasar filas completas (cada fila indica el grado de dedicación de cada empleado a las distintas tareas) de los padres al hijo. Por otra parte, como se puede apreciar en la Figura 1 la preservación de la información contenida en los padres es más compleja de lograr en el caso de que el AG utilice 2DMX.

La eficacia de un operador está muy relacionada con su probabilidad de aplicación (p_c). Por consiguiente, la elección de un valor adecuado de p_c afecta críticamente el comportamiento y rendimiento de los AGs. En particular la probabilidad de cruce controla la capacidad de los AGs para explotar un pico localizado con el objetivo de alcanzar un óptimo local. Cuanto más alta es la probabilidad de cruce, se da lugar a una rápida explotación. Pero, una p_c muy alta deberá disromper//perturbar los individuos en forma más rápida de lo que puedan ser explotados. Las configuraciones sugeridas para la probabilidad de cruce son $p_c = 0{,}6$ [5], $p_c = 0{,}95$[6] y $p_c \in [0{,}75, 0{,}95]$[12]. Estos valores fueron derivados de estudios empíricos sobre un determinado conjunto de problemas de prueba, y pueden ser inadecuado debido a que el valor de p_c es específico para el problema en estudio. Teniendo en cuenta estas consideraciones, en este trabajo se plantea analizar distintos valores de probablidades, desde bajos a altos, para determinar cuál de ellas es la más adecuada para los operadores estudiados, a fin de obtener un buen rendimiento del AG para resolver el problema de planificación de proyectos.

4. Parametrización

En esta sección se muestran los valores paramétricos utilizados para evaluar el rendimiento de las variantes del AG propuesto para resolver el problema de planificación de proyectos de software. Para llevar a cabo esta comparación se proponen las siguientes variantes de AG: (*i*) un AG con operador de cruce binario de un punto (AG_1X) y (*ii*) un AG con operador de cruce sobre matrices (AG_2DMX). Además, para evitar ajustar el algoritmo a una situación particular y permitir una comparación justa, por cada variante se propone el estudio de su comportamiento al utilizar distintas probabilidades de cruce, a saber: 0.3, 0.5, 0.75 y 0.9, es decir, de bajos valores de cruce a altos. De esta manera, resultan el estudio de ocho variantes algorítmicas (2 cruces por 4 probabilidades cada uno).

El tamaño de la población (μ) es de 64 individuos y es inicializada aleatoreamente. En cada generación se crean 64 individuos (λ). Como criterio de selección de padres se utiliza torneo binario y el reemplazo es ($\mu + \lambda$) utilizando selección proporcional por ruleta. El operador de mutación es el bit-flip con probabilidad igual a 0.005. El máximo número de evaluaciones está fijado en 20000.

Para realizar las experimentaciones y comparar los efectos de los operadores de cruce con las distintas probabilidades se tomaron las instancias propuestas

en [4]. Estas instancias representan una variedad de escenarios de la vida real. Los componentes de las instancias son: empleados, tareas, habilidades, y el grafo de precedencia de tareas (TPG). Las instancias empleadas representan proyectos que varían en el número de empleados: 5, 10 y 15, en el número de tareas: 10, 20 y 30, y en el número de habilidades de los empleados: 4 a 5, 6 a 7, 5 y 10 habilidades. Una de ellas podría ser por ejemplo un proyecto que cuenta con 10 empleados, 30 tareas, y 6 a 7 habilidades por empleado. De la combinación de habilidades, empleados y tareas se obtiene un total de 36 instancias. Los valores de los pesos de la función de fitness son [4]: $w_{costo} = 10^{-6}$, $w_{tiempo} = 10^{-1}$, $w_{penal} = 100$, $w_{t_inc} = 10$, $w_{h_nocub} = 10$ y $w_{extras} = 0{,}1$.

La experimentación realizada ha sido extensa: las ocho variantes algorítmicas en estudio se evaluaron con cada una de las instancias, en total se obtienen un total de 288 combinaciones (8 algoritmos $\times$ 36 instancias). Debido a la naturaleza estocástica de los algoritmos, cada una de ellas se ejecutan 30 veces para obtener una muestra confiable. En total se realizaron 30 $\times$ 288 ejecuciones haciendo un total de 8640 ejecuciones. Hemos realizado un análisis estadístico de los resultados a fin de obtener conclusiones significativas, en particular se aplicó el test no paramétrido Kruskal Wallis, considerando un nivel de significancia de $\alpha = 0{,}05$, para indicar un nivel de confidencia del 95 % en los resultados.

El algoritmo ha sido implementado con la librería MALLBA bajo C++. El entorno de ejecución consiste de máquinas con triple procesador AMD Phenom8450 a 2GHz, con 2 GB de RAM, bajo Linux, versión 2.6.27-4GB kernel.

5. Resultados

En esta sección se presentan y analizan los resultados obtenidos por las dos variantes de algoritmos genéticos presentados (AG_1X y AG_2DXM). En una primera etapa, se determina para cada variante cuál es la probabilidad de cruce que permite un mejor comportamiento del algoritmo. En una segunda fase, se procederá a comparar las mejores variantes a fin de determinar la configuración óptima del algoritmo.

Comenzamos el estudio con el examen del rendimiento de las variantes propuestas bajo distintas probabilidades de cruce. Al analizar los mejores resultados obtenidos para AG_1X, no se observa un claro patrón de comportamiento para determinar cuál probabilidad es la más adecuada para resolver el problema o grupo de instancias. Por tal motivo hemos recurrido a los estudios estadísticos para definir cuál probabilidad es la que brinda diferencias estadísticas significativas. El Cuadro 1 muestra los p-values del estudio estadístico realizado sobre AG_1X al comparar los valores medios obtenidos con las distintas probabilidades de cruce para cada una de las instancias del problema. Se puede observar que en 21 de las 36 instancias analizadas los valores son mayores a 0.05, el nivel de significancia, indicando que el comportamiento del AG no está influenciado por la probabilidad de cruce utilizada para resolver el problema PSP. En las restantes instancias (15 de 36) donde las diferencias entre los algoritmos es significante, se realizó un test de múltiple comparaciones que indica que AG_1X con $p_m = 0{,}3$

Cuadro 1. p-values del estudio paramétrico sobre las mejores soluciones de AG_1X discriminadas por instancia

empleados	habilidades	tareas		
		10	20	30
5	4-5	**0,265**	0,020	**0,051**
	6-7	**0,239**	0,018	**0,149**
	5	**0,590**	**0,153**	0,009
	10	0,000	0,001	**0,531**
10	4-5	0,000	**0,152**	**0,092**
	6-7	**0,130**	0,000	0,000
	5	**0,404**	0,008	0,044
	10	**0,360**	**0,327**	0,002
15	4-5	**0,331**	0,001	**0,081**
	6-7	0,015	0,016	**0,072**
	5	**0,627**	0,003	**0,875**
	10	**0,093**	**0,112**	**0,167**

marca las diferencias, resultando la variante algorítmica con el peor comportamiento. En cuanto al resto de las variantes planteadas, AG_1X con $p_m = 0{,}75$ y $p_m = 0{,}9$ no presentan rangos de valores medios con significancia estadística, pero para estas instancias en 8 de las 14 instancias 0.9 presenta valores medios más altos (lo que indica mejor calidad de resultados en promedio).

En cuanto a los resultados obtenidos con AG_2DXM, se observa una situación similar a la anterior referente a determinar en forma empírica la probabilidad que contribuyó a obtener los mejores resultados. A diferencia del análisis estadístico previo, en esta oportunidad los p-values son marcadamente menores al nivel de significancia 0.05, indicando que las distintas probabilidades arrojan resultados diferentes en todas las instancias analizadas (debido a esta homogeneidad de resultados es que no se ha incluido el cuadro correspondiente). El test de múltiples comparaciones sugiere que en comparación el rendimiento del algoritmo fue significantemente mejorado al usar una probabilidad de 0.9 en 27 de las 36 instancias. Nuevamente, la probabilidad que marca las diferencias es 0.3.

Concluimos, a partir de los resultados, que AG_1X y AG_2DXM obtienen sus mejores rendimientos cuando la probabilidad de cruce está configurada a 0,9. Esto sugiere que el algoritmo necesita altos valores de probabilidades para lograr obtener un buen rendimiento. Esta configuración de ambas variantes son las que se usarán para llevar a cabo las siguientes comparaciones.

El siguiente paso consiste en determinar cuál combinación de cruce y probabilidad es la que genera un mejor rendimiento del AG para resolver el problema de PSP. La Figura 2 muestra los valores de fitness de cada algoritmo para cada instancia. Como se puede observar AG_1X obtiene los mejores valores de fitness en la mayoría de las instancias (valores de fitness más altos). El test no paramétrico indica que en 30 de las 36 instancias hay diferencias significativas entre las dos variantes, dando soporte a las observaciones anteriores. De esta manera, el usar un cruce de un punto juega un rol importante para mejorar el rendimiento del algoritmo.

En función de la cantidad de generaciones promedio que cada variante necesita para alcanzar sus mejores soluciones (ver Figura 3), AG_2DXM presenta mayor velocidad de convergencia en 21 de las 36 instancias. Pero el estudio es-

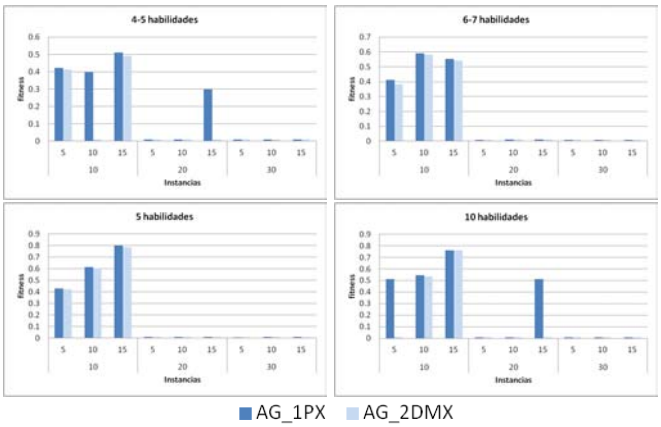

Figura 2. Mejores valores de fitness alcanzados por cada algoritmo, discriminados por instancia.

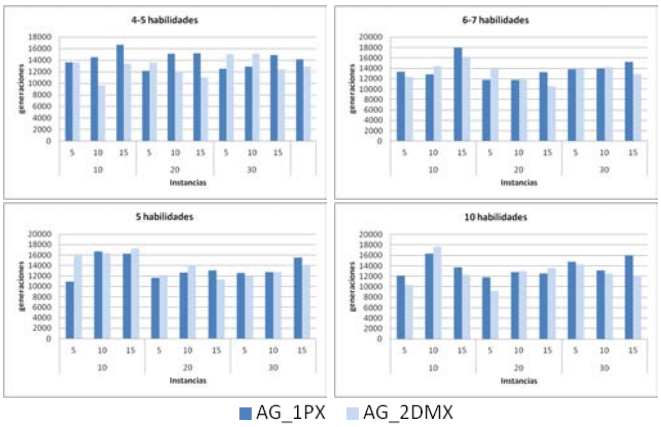

Figura 3. Evaluaciones promedio necesarias para obtener las mejores soluciones por cada algoritmo, descriminadas por instancia.

tadístico realizado indica que las diferencias son significativas sólo en 9 de las 36 instancias. Esto sugiere la superioridad de AG_1X sobre AG_2DXM para alcanzar mejores soluciones con una velocidad de convergencia similar.

Un aspecto importante a analizar es el porcentaje de éxito de cada variante, determinada por el porcentaje de soluciones obtenidas que son factibles, es decir soluciones que cumplen con todas las restricciones impuestas en la resolución del problema (ver Sección 2). El Cuadro 2 muestra el porcentaje de las mejores soluciones de cada algoritmo que son factibles (cero indica que no se encontraron soluciones factible). El porcentaje de factibilidad está totalmente vinculado con la cantidad de tareas que se deben realizar en cada proyecto: cuanto mayor es

Cuadro 2. Porcentaje de soluciones factibles de cada algoritmo AG_1X y AG_2DXM

		10 tareas		**20 tareas**		**30 tareas**	
empleados	habilidades	AG_1X	AG_2DXM	AG_1X	AG_2DXM	AG_1X	AG_2DXM
5	4-5	**100**	**100**	0	0	0	0
	6-7	**29**	21	0	0	0	0
	5	71	**93**	0	0	0	0
	10	0	0	0	0	0	0
10	4-5	**64**	0	0	0	0	0
	6-7	**100**	**100**	**7**	0	0	0
	5	86	**100**	0	0	0	0
	10	**100**	**100**	0	0	0	0
15	4-5	**100**	36	**21**	0	0	0
	6-7	**100**	**100**	0	0	0	0
	5	**100**	**100**	0	0	0	0
	10	14	**64**	**100**	36	0	0

la cantidad de tareas menor es el porcentaje de factibilidad obtenido por ambos algoritmos. En el caso de proyectos con 10 tareas ambos algoritmos encuentran soluciones factibles para todas las intancias, en 9 de las 12 instancias el porcentaje de AG_1X está por encima del 60 %, caso similar se observa para AG_2DXM (8 sobre 12 instancias). También vale resaltar que en 7 instancias alguno de los dos algoritmos obtienen en todas las ejecuciones soluciones factibles (porcentaje de éxito del 100 %). Para proyectos con 20 tareas AG_1X obtiene mayor porcentaje de soluciones factibles que AG_2DXM. Comparando nuestros resultados con los porcentajes de factibilidad obtenidos en [4] y en [9], observamos que son similares. Una observación importante es que en las instancias que representan proyectos con 30 tareas, los porcentajes de éxito mostrados en la literatura son cercanos a 0, no presentando importantes diferencias con nuestra propuesta.

6. Conclusiones

Este trabajo presenta una opción evolutiva para determinar la asignación de tareas a empleados, con el objetivo de minimizar el tiempo de duración del proyecto. La opción evolutiva es un algoritmo genético tradicional con representación binaria. El estudio consistió en analizar el comportamiento de este algoritmo bajo dos tipos de cruce: el tradicional operador de un punto para representaciones binarias y un operador de cruce a nivel matricial ya que las soluciones al problema son matrices que representan la dedicación de cada empleado a las tareas. También se analizaron cuatro valores distintos de probabilidades de cruce ya que es un parámetro influyente en el rendimiento de un algoritmo. Los resultados muestran que para este problema es conveniente utilizar probabilidades de cruce altas. La variante algorítmica utilizando el operador tradicional obtuvo el mejor rendimiento, obteniendo buena calidad de patrones de planificación. Los porcentajes de factibilidad de las soluciones obtenidas son altos para problemas de planificación no tan complejos, estos valores son similares a los obtenidos por otras variantes evolutivas de la literatura para resolver el problema en cuestión.

Como trabajo futuro proponemos el desarrollo de un operador genético más específico del problema o un algoritmo de búsqueda local que permita mejorar

los niveles de factibilidad de las soluciones obtenidas, para que tengan sentido para aquellos administradores que llevan a cabo un proyecto de desarrollo de software. En este sentido, también se pondrá atención en el mecanismo de manejo de restricciones, tal como está planteado se usa un mecanismo de penalización pero se podrían considerar mecanismos más avanzados. Además del estudio experimental para proponer otros pesos para la función de fitness.

Reconocimientos

Los autores agradecen el apoyo de la UNLPam y ANPCYT (proyecto 09F-049). Germán Dupuy agradece al CIN por la beca de EVC 2012.

Referencias

1. E. Alba and J. F. Chicano. Software project management with GAs. *Information Sciences*, 177:2380–2401, 2007.
2. C. K. Chang, M. J. Christensen, and T. Zhang. Genetic algorithms for project management. *Annals of Software Engineering*, 11:107–139, 2001.
3. C. K. Chang, H. Jiang, Y. Di, D. Zhu, and Y. Ge. Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology*, 50(11):1142–1154, 2008.
4. J.F. Chicano. *Metaheurísticas e Ingeniería del Software.* PhD thesis, University of Málaga, 2007.
5. K. DeJong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, MI, 1975.
6. J. J. Greffenstette. Optimization of control parameters for genetic algorithms. *IEEE Transaction on System Man and Cybernetic*, 16(1):122–128, 1986.
7. J.H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, 1975.
8. M. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, third revised edition, 1996.
9. L.L. Minku, D. Sudholt, and X. Yao. Evolutionary algorithms for the project scheduling problem: runtime analysis and improved design. In *Proceedings of International Conference on Genetic and Evolutionary Computation Conference*, GECCO '12, pages 1221–1228, 2012.
10. G. Ochoa, I. Harvey, and H. Buxton. On recombination and optimal mutation rates. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 488–495. Morgan Kaufmann, 1999.
11. L. Ozdamar and G. Ulusoy. A survey on the resource-constrained project scheduling problem. *IIE Transactions*, 27:574–586, 1995.
12. J.D. Schaffer, R.A. Caruana, L.J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60, 1989.
13. B.M. Wall. *A genetic algorithm for resource-constrained scheduling.* PhD thesis, MIT, 1996.