Generación automática de programas: Savant Virtual para el problema de la mochila

Renzo Massobrio[1], Bernabé Dorronsoro[2], Francisco Palomo-Lozano[2], Sergio Nesmachnow[1] y Frédéric Pinel[3]

[1] Universidad de la República, Uruguay
{renzom, sergion}@fing.edu.uy
[2] Universidad de Cádiz, España
{bernabe.dorronsoro, francisco.palomo}@uca.es,
[3] Université de Luxembourg, Luxemburgo
frederic.pinel@uni.lu

Resumen Este trabajo estudia la aplicación de Savant Virtual (SV) para la generación automática de programas para resolver el problema de la mochila. SV es un nuevo método que utiliza aprendizaje computacional para aprender cómo resolver el problema a partir de un algoritmo de referencia, siendo capaz de resolver eficientemente nuevas instancias, incluso de mayor tamaño. SV sólo se ha aplicado con anterioridad a un problema de planificación, por lo que es de gran interés estudiar su comportamiento en otros problemas. El estudio se realiza sobre instancias caracterizadas por su nivel de dificultad, lo que facilita el análisis del comportamiento de SV según la dificultad del problema. En este trabajo tomamos como referencia el óptimo, y SV calcula soluciones con un error medio del 3,2 % con respecto al óptimo para todas las instancias (llegando a encontrar el óptimo en varios casos). Así mismo, SV tiene un coste computacional sensiblemente menor al del algoritmo exacto tomado como referencia.

Palabras clave: optimización, aprendizaje supervisado, problema de la mochila

1. Introducción

La resolución de problemas de optimización es una tarea compleja. En la literatura se ha propuesto una plétora de métodos para resolver problemas complejos, siguiendo un enfoque generalista orientado a maximizar la aplicabilidad y la versatilidad de las técnicas de optimización [10]. Esta decisión implica que se debe dominar el problema a resolver, para poder instanciar los métodos genéricos, haciendo más compleja la tarea del programador. Asimismo, las dificultades se agudizan con la tendencia actual en el diseño de hardware, orientada a arquitecturas paralelas con un número cada vez mayor de recursos de cómputo [8]. Por estos motivos, existe una necesidad real de nuevas herramientas que ayuden al programador a aprovechar eficientemente este tipo de arquitecturas.

Savant Virtual (SV) es un novedoso método que genera automáticamente programas paralelos para resolver problemas de optimización. Aplicando técnicas de aprendizaje computacional, SV es capaz de aprender de las soluciones a un

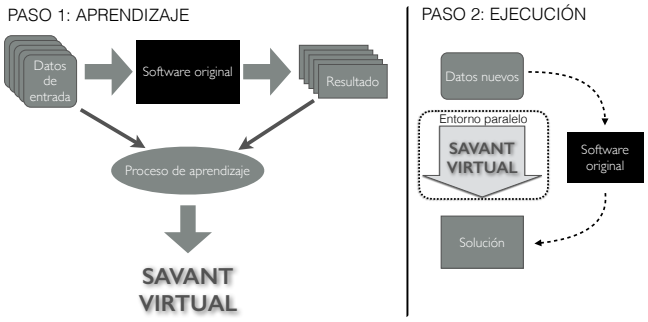


Figura 1: SV aprende la salida producida por un algoritmo para un número dado de entradas (izquierda), generando un programa completamente distinto que reproduce su comportamiento con alta eficiencia computacional (derecha).

problema dado calculadas por un algoritmo de referencia (tratado como una caja negra). Tras el aprendizaje, SV resuelve con gran precisión instancias nuevas del problema, incluso instancias con un número mayor de variables. Además, el diseño de SV permite aprovechar eficientemente los recursos de arquitecturas paralelas, lo que hace posible escalar el tamaño de la instancia.

SV fue propuesto recientemente [12], mostrando su utilidad para el problema de planificación de tareas. SV fue capaz de encontrar el óptimo en el 95,5 % de las 200 instancias con las que se evaluó el método (no utilizadas en el proceso de aprendizaje), considerando 30 ejecuciones independientes de SV, para un espacio de búsqueda de 2^{24} soluciones. Al considerar instancias con un espacio de búsqueda de 2^{2048} soluciones, SV fue incluso capaz de mejorar significativamente los resultados del algoritmo utilizado para generar las soluciones de referencia.

La principal contribución de este artículo es el estudio del comportamiento de SV en otro problema de optimización combinatoria: el problema de la mochila. El estudio es necesario para mostrar la utilidad de SV como método de optimización combinatoria, puesto que anteriormente sólo se ha evaluado en un problema. El análisis permite analizar el comportamiento de SV de acuerdo a la dificultad gradual de las instancias del problema, aplicando la metodología de [6].

El artículo se estructura como sigue. La Sección 2 presenta el Savant Virtual. La Sección 3 describe trabajos relacionados sobre programación automática. El problema de la mochila se describe en la Sección 4 y la aplicación de SV al problema se detalla en la Sección 5. La Sección 6 reporta el análisis experimental. Finalmente, se presentan las conclusiones y líneas de trabajo futuro.

2. Savant Virtual

SV aprende el funcionamiento de programas de optimización para generar automáticamente un programa paralelo completamente nuevo. SV requiere un proceso de aprendizaje en el que se genera el código paralelo que en un segundo paso se ejecuta para resolver el problema (Figura 1).

En el proceso de aprendizaje, SV recibe un conjunto de instancias del problema y los resultados del programa de referencia para cada una de ellas. Mediante técnicas de aprendizaje computacional, es capaz de abstraer el comportamiento de cualquier algoritmo que genere resultados correspondientes a las instancias del problema. Una vez completado el paso de aprendizaje, SV puede ejecutarse en paralelo para reproducir los resultados que ofrecería el programa original a instancias del problema no utilizadas en el proceso de aprendizaje.

El diseño de SV utiliza un clasificador para determinar el valor de cada variable en función del conjunto de características de la instancia del problema. Por tanto, SV se puede ejecutar siguiendo el modelo de computación paralela MapReduce [5], que permite utilizar fácilmente tantos procesos paralelos como variables en el problema. Además, resolver problemas con un número mayor de variables es tan sencillo como ejecutar más réplicas del clasificador en paralelo.

Las principal dificultad de este método, común en el campo del aprendizaje computacional, radica en la capacidad de extraer de la instancia del problema las características que permitan al clasificador realizar una clasificación correcta y precisa. En la Sección 6.2 estudiamos este aspecto.

3. Trabajos relacionados

En general, los trabajos existentes enfocados en la paralelización automática de software se basan en la aplicación de transformaciones en el código fuente [2,7] y técnicas de inteligencia computacional [11,15]. La ventaja de estos métodos es que preservan la semántica del programa original, pero como contrapartida la eficiencia y escalabilidad de los programas generados son limitadas.

Walsh and Ryan [13] presentaron Paragen, un método que combina Programación Genética (PG) y transformaciones sobre el código para generar programas paralelos. Paragen desensambla un programa en sus componentes y luego trata de reconstruirlo para obtener un código paralelo. Una de las suposiciones de esta técnica es que el código resultante se ejecuta en una máquina virtual síncrona, evitando así el acceso concurrente a la memoria compartida. Nuestro enfoque no realiza esta suposición y es apropiado para procesadores independientes (e incluso distintos) conectados por red.

Weise y Tang [14] proponen utilizar PG para evolucionar programas distribuidos. La técnica no asegura preservar la semántica de los programas, por lo que su idoneidad funcional sólo se puede evaluar mediante simulaciones. Como en los casos anteriores, esta técnica trabaja sobre el código original, mientras que nuestra propuesta toma el código de referencia como una caja negra, permitiendo paralelizar programas para los que el código fuente no está disponible.

4. Caso de estudio: el problema de la mochila

El problema de la mochila es un problema clásico de optimización combinatoria $\mathcal{NP}$-difícil [9]. Dado un conjunto E de n objetos, cada uno con un *beneficio*, p_i, y un *peso*, w_i, el problema consiste en encontrar un subconjunto de objetos

que maximicen el beneficio total, sin exceder la capacidad de la mochila W. Se asume que todos los beneficios y pesos son positivos, que todos los pesos son menores a W y que el peso total de E excede a W.

La Ecuación 1 formula el problema (optimización lineal entera), donde $x_i \in \{0, 1\}$ indica si el objeto i se incluye o no en la mochila.

$$\text{arg máx} \left\{ \sum_{i=1}^{n} p_i x_i \;\middle|\; \sum_{i=1}^{n} w_i x_i \leqslant W \right\} \tag{1}$$

A pesar de su formulación sencilla, el problema de la mochila tiene un espacio de soluciones de alta dimensión y suele utilizarse como problema de prueba (*benchmark*) para algoritmos de optimización. Así mismo, el problema cuenta con múltiples variantes y problemas relacionados con aplicación directa en varias áreas. Un caso concreto es el Next Release Problem [1], un problema real en Ingeniería de Software que es una versión del problema de la mochila para modelar la toma de decisiones sobre las funcionalidades a incluir en el desarrollo de una nueva versión de un producto de software.

En el contexto de nuestro trabajo, el problema de la mochila es útil para evaluar SV por varios motivos: (1) es un problema de optimización combinatoria $\mathcal{NP}$-difícil; (2) permite estudiar el comportamiento de SV para problemas con variables binarias y con restricciones simples (SV se ha aplicado previamente a un problema de planificación con variables enteras y sin restricciones); (3) se dispone de una metodología para generar instancias con una dificultad gradual, basada en la correlación beneficio/peso de los objetos.

5. Aplicación de Savant Virtual al problema de la mochila

El clasificador asigna el valor de una única variable (también referido como *objeto* en el problema de la mochila). Cada clasificador utiliza únicamente información relativa al objeto en cuestión para la asignación (Figura 2).

En este trabajo se utiliza una Support Vector Machine (SVM) [4] como clasificador, siguiendo el diseño original de SV [12]. La SVM implementa un método supervisado de aprendizaje computacional para clasificación, para decidir si el objeto es incluido en la mochila o no. Se utilizan las soluciones óptimas al problema [6] para el entrenamiento. En cada instancia del problema hay N observaciones, donde N es el número de objetos de la instancia. Utilizamos la implementación de SVM disponible en libSVM [3]. El procedimiento seguido para el aprendizaje utiliza el *kernel* Radial Basis Function (RBF), un proceso de validación cruzada para configurar los parámetros de la SVM y del kernel, y los datos de entrada (las características extraídas de la instancia) están escalados.

La aplicación de la SVM retorna la probabilidad de asignar el objeto dado a una de las dos posibles clases. Como se aprecia en la Figura 2, se ejecuta en paralelo una réplica de la SVM para cada variable del problema. Tras este proceso, se tiene un vector que contiene la probabilidad de que cada objeto se incluya o no en la mochila, según lo aprendido por la SVM. Estas probabilidades se asignan de forma completamente independiente, sin tener en cuenta las asignaciones

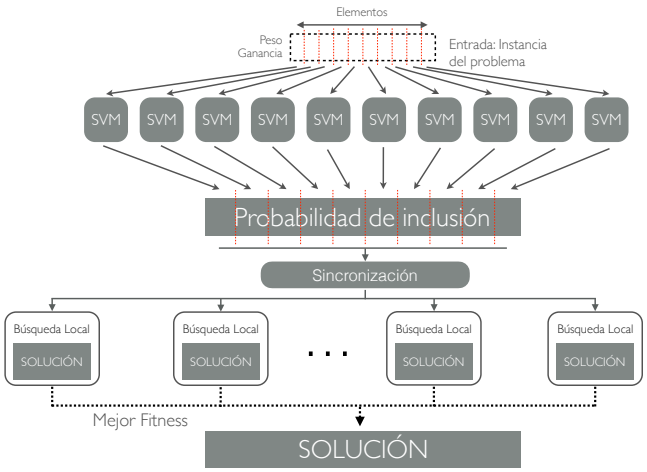


Figura 2: Aplicación de SV al problema de la mochila.

realizadas en los otros objetos. Por tanto, es posible que se produzcan errores en la predicción de la SVMs, llevando a soluciones lejanas al óptimo, o incluso no factibles (que sobrepasan la capacidad de la mochila). Por ello, SV incorpora un segundo paso en el que aplica una búsqueda local (BL) para corregir y mejorar las soluciones. En el modelo general, pueden ejecutarse tantas instancias de la BL en paralelo como recursos haya disponibles, retornando la solución que mejor valor obtiene para la función objetivo. En nuestro trabajo, se aplica una BL y se estudiaron dos mecanismos para la corrección de soluciones no factibles:

- *Corrección ávida por beneficio (CB)*: elimina iterativamente el objeto de menor beneficio hasta que la solución sea válida.
- *Corrección ávida por peso (CP)*: iterativamente busca los objetos cuyo peso sea mayor o igual al sobrepeso de la solución y entre ellos elimina el de menor peso. Si ningún objeto cumple la condición, se elimina el de mayor peso.

Como operador de BL se implementó una búsqueda aleatoria que parte del vector solución calculado por la SVM (seleccionando la clase con mayor probabilidad para cada variable) e intenta mejorar la solución aplicando cambios sobre la misma. En cada iteración de la BL se invierte un valor aleatorio de la solución, se evalúa la calidad de la nueva solución, y la búsqueda continúa desde dicha solución si mejoró a la anterior. Para una solución con beneficio B; peso P; sobrepeso $S = P - C$, donde C es la capacidad de la mochila; $k > 0$; $m \in (0, 1)$; el valor de evaluación se asigna de acuerdo al Algoritmo 1.

6. Análisis experimental

En esta sección se describen las instancias del problema utilizadas y se analizan la mejor configuración para el aprendizaje de la SVM y su precisión en la clasificación sobre instancias no utilizadas en el proceso de entrenamiento. Finalmente, se analiza la calidad de las soluciones encontradas por SV.

Algoritmo 1: Algoritmo de evaluación de soluciones durante la BL

entrada: solucion, instancia
escalar (P, B, S, C, instancia)
si $S \leq 0$ **entonces devolver** B
si no, si $S \leq m \cdot C$ **entonces devolver** $B - k \cdot S$
en otro caso devolver $-S$

6.1. Instancias de evaluación

El benchmark utilizado [6], disponible en https://ucase.uca.es/nrp, contiene instancias clasificadas según la correlación entre el peso y el beneficio de los objetos, que es una medida de la dificultad que entraña resolver la instancia para la mayoría de algoritmos. El benchmark contiene 50 datasets, y cada uno de ellos contiene instancias de tamaño 100 a 1.500 (variando de 100 en 100 objetos). Para cada tamaño hay instancias cuya correlación varía de 0,0 a 1,0, con saltos de 0,05, haciendo un total de 15.750 instancias.

6.2. Entrenamiento de la SVM

En los análisis de configuración del proceso de aprendizaje de la SVM se utilizó el dataset 1 como entrada de la etapa de entrenamiento y se evaluó la precisión al predecir los datasets 2 a 5. Se analizaron tres configuraciones de atributos del problema a tomar en cuenta en el proceso de clasificación:

- *C1*: Peso del objeto, su beneficio, y la capacidad de la mochila (3 atributos).
- *C2*: Peso del objeto, su beneficio, y el ratio entre la capacidad de la mochila y el número total de objetos (3 atributos).
- *C3*: Peso del objeto, su beneficio, la capacidad de la mochila, y el número total de objetos (4 atributos).

Se estudió la *precisión media* (porcentaje de predicciones correctas realizadas por la SVM) para cada uno de los datasets utilizando las tres configuraciones de atributos mencionadas. En el problema abordado, corresponde a predecir correctamente si un determinado objeto debe o no ser incluido en la mochila. Los resultados mostraron pequeñas diferencias entre las configuraciones, obteniendo altos valores de precisión en todos los casos, entre 89,4 % y 89,7 %. Por este motivo, el análisis experimental se realizó con la configuración *C1*, por ser la más simple y directa de acuerdo a la formulación del problema de la mochila.

Los resultados del entrenamiento utilizando distinta cantidad de observaciones del dataset 1 se reportan en la Tabla 1. Se observa una diferencia significativa (del orden de 31 %) en los valores de precisión alcanzados al entrenar con un 15 % de las observaciones del dataset 1, frente a los valores obtenidos al entrenar con un 10 % de las observaciones. Al aumentar la cantidad de observaciones por encima del 15 % del total las mejoras en precisión son marginales, mientras que los tiempos de entrenamiento aumentan significativamente. Por esta razón se decidió entrenar la SVM con el 15 % de las observaciones del dataset 1.

Tabla 1: Precisión de SV según el número de observaciones en el entrenamiento

	Número de observaciones (% del total)				
Dataset	252000 (100 %)	126000 (50 %)	63000 (25 %)	37800 (15 %)	25200 (10 %)
Dataset-2	89,6 %	89,5 %	89,4 %	89,4 %	58,0 %
Dataset-3	89,5 %	89,4 %	89,3 %	89,3 %	57,9 %
Dataset-4	89,6 %	89,5 %	89,3 %	89,3 %	58,0 %
Dataset-5	89,7 %	89,6 %	89,4 %	89,4 %	58,0 %

Finalmente, se realizó la configuración de los parámetros de la SVM y del kernel RBF utilizado para la clasificación (parámetros C y γ respectivamente). Para esta etapa se utilizó la técnica de validación cruzada (*cross validation*) sobre un conjunto de 5.000 entradas seleccionadas aleatoriamente del dataset 1. Los resultados de la validación cruzada revelan que los mejores resultados se alcanzan al utilizar $C = 8192$ y $\gamma = 0,5$. La precisión al utilizar la validación cruzada aumenta del $89,35\,\%$ al 90,48 %, en media para todos los datasets.

6.3. Precisión de la SVM

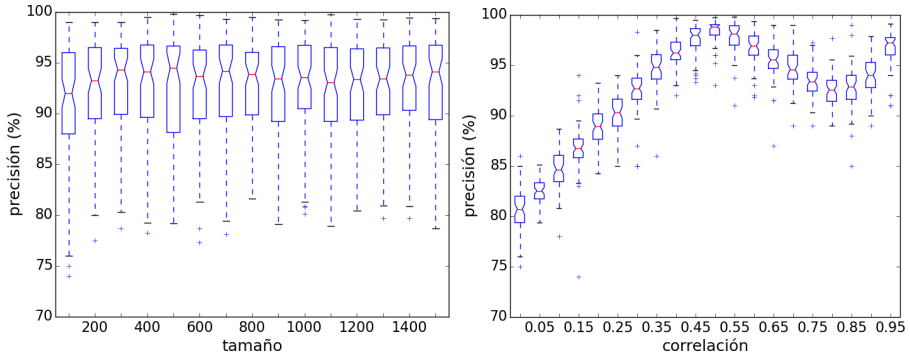
La Figura 3 muestra los diagramas de caja correspondientes a la precisión alcanzada en cada conjunto de instancias, agrupadas según tamaño (3a) y correlación peso/ganancia (3b). La mediana de la precisión en la asignación de la SVM es ampliamente superior al 90 % para todos los tamaños de instancia estudiados (Figura 3a). No se aprecian diferencias significativas entre la precisión de la SVM para los distintos tamaños de instancia. Para todos los tamaños de instancia existen casos cercanos al 100 % de precisión.

Las instancias con correlación cercana al 0,5 son las más sencillas de predecir para la SVM, ofreciendo valores cuya mediana se encuentra sobre el 97 % (Figura 3b). Además, la mediana de la precisión de la SVM es superior al 80 % en el peor caso. Por último, se pueden apreciar diferencias significativas en la mayoría de las comparaciones entre las instancias con los distintos valores de correlación.

La Figura 4 muestra un ejemplo de la precisión detallada de la SVM para clasificar correctamente cada objeto (los resultados medios corresponden a todas las instancias de tamaño 100 y correlación 0,5 de los datasets 2 a 5). Los objetos se consideran ordenados en forma ascendente según su peso. Se puede apreciar cómo la probabilidad media de realizar una correcta asignación es del entorno del 90 %, inclusive alcanzando el 100 % (o valores muy cercanos) en varios objetos, tanto de peso pequeño como grande. Además, la probabilidad de acierto no es inferior al 70 % en ningún caso.

6.4. Calidad de soluciones encontradas por SV

En esta sección analizamos el comportamiento de SV en función de la calidad de resultados obtenidos, y no de su similitud con la solución original, como



(a) Según el tamaño de la instancia (b) Según la correlación de la instancia

Figura 3: Precisión alcanzada por la SVM respecto a la solución óptima.

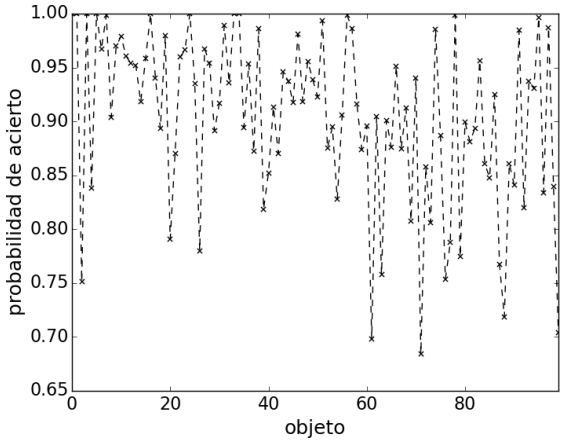


Figura 4: Probabilidad de acierto de la SVM para cada objeto en instancias de tamaño 100 y correlación 0,5

hemos hecho en las secciones anteriores. En este sentido, la Tabla 2 muestra los resultados alcanzados por SV con las distintas versiones de BL propuestas, agrupados de acuerdo al tamaño y a la correlación de la instancia. Se muestra la tasa promedio del beneficio alcanzado con respecto al óptimo al aplicar solamente los operadores de corrección de soluciones (CB y CP) y al combinar la búsqueda local con los operadores de corrección (BL+CB, BL+CP). Se realizaron 30 ejecuciones independientes de 1.000 iteraciones cada una de la búsqueda local sobre cada instancia de prueba. Se utilizó $m = 0,2$ y $k = 2$ como parámetros para la función de evaluación de soluciones durante la BL. De esta manera, se permite explorar soluciones no factibles que sobrepasen la capacidad de la mochila hasta en un 20 %, aplicando una penalización que favorezca a las soluciones factibles.

Se aprecia una elevada calidad de los resultados obtenidos por SV. Según el tamaño de la instancia, SV consigue soluciones con sólo el 3 % de error con

respecto al óptimo tras aplicar la BL propuesta con cualquiera de los dos métodos de corrección, en media. Creemos que se trata de un resultado alentador, puesto que la BL es un proceso simple en el que solamente se evalúan 1.000 cambios aleatorios en la solución. Además, tras aplicar únicamente cualquiera de los dos sencillos métodos de reparación de soluciones, el resultado de SV se encuentra alrededor del 10 % del óptimo. Con respecto al estudio basado en la correlación de las instancias, vemos que SV es capaz de encontrar siempre el óptimo para todas las instancias con correlación 0,5 y 0,95. La calidad de las soluciones, en media para todas las correlaciones, es de tan sólo un 3,15 % y 3,20 % de error con respecto al óptimo para SV usando BL+CB y BL+CP, respectivamente.

Tabla 2: Calidad de las soluciones de SV respecto al óptimo (valor medio para todas las instancias de cada clase) en función del tamaño (n) y la correlación (ρ)

n	CB	CP	BL+CB	BL+CP	ρ	CB	CP	BL+CB	BL+CP
100	0.89	0.89	0.96	0.96	0.00	0.70	0.70	0.91	0.91
200	0.90	0.91	0.97	0.97	0.05	0.72	0.72	0.92	0.92
300	0.90	0.91	0.97	0.97	0.10	0.76	0.76	0.93	0.93
400	0.90	0.91	0.97	0.97	0.15	0.78	0.78	0.94	0.94
500	0.90	0.90	0.97	0.97	0.20	0.81	0.81	0.95	0.95
600	0.90	0.91	0.97	0.97	0.25	0.84	0.84	0.96	0.96
700	0.90	0.91	0.97	0.97	0.30	0.87	0.87	0.97	0.97
800	0.90	0.91	0.97	0.97	0.35	0.91	0.91	0.98	0.98
900	0.90	0.91	0.97	0.97	0.40	0.94	0.94	0.98	0.98
1000	0.90	0.91	0.97	0.97	0.45	0.96	0.96	0.99	0.99
1100	0.90	0.91	0.97	0.97	0.50	0.98	0.98	1.00	0.99
1200	0.90	0.91	0.97	0.97	0.55	0.99	0.99	0.99	0.99
1300	0.91	0.91	0.97	0.97	0.60	0.98	0.99	0.99	0.99
1400	0.91	0.91	0.97	0.97	0.65	0.98	0.99	0.98	0.98
1500	0.90	0.91	0.97	0.97	0.70	0.97	0.99	0.98	0.98
					0.75	0.97	0.99	0.97	0.97
					0.80	0.96	0.99	0.97	0.97
					0.85	0.96	0.99	0.98	0.98
					0.90	0.97	0.99	0.98	0.98
					0.95	0.95	0.95	1.00	1.00

7. Conclusiones y trabajo futuro

Este trabajo presenta la aplicación del método Savant Virtual para la resolución del problema de la mochila. El enfoque utilizado permite generar de forma automática programas capaces de aprender el comportamiento de otro algoritmo de optimización que resuelve el problema utilizando técnicas de aprendizaje automático. Para el caso de estudio abordado en este trabajo, se utilizó un algoritmo exacto como referencia. El análisis experimental permitió concluir que las soluciones alcanzadas por SV tienen un error medio del 3,2 % con respecto al

óptimo para todos los tipos de instancias estudiadas. Adicionalmente, el enfoque de paralelismo propuesto permite utilizar n hebras independientes entre sí (siendo n el tamaño de la instancia del problema), permitiendo resolver instancias grandes del problema en forma masivamente paralela.

Las principales líneas de trabajo futuro incluyen mejorar los resultados de SV incorporando búsquedas locales especializadas y la ejecución paralela de varias búsquedas locales. Asimismo, resulta de interés aplicar SV a otros problemas de optimización y el diseño de modelos híbridos entre SV y metaheurísticas.

Agradecimientos

El trabajo de R. Massobrio y S. Nesmachnow ha sido parcialmente financiado por PEDECIBA y ANII, Uruguay. B. Dorronsoro agradece su apoyo al MINECO (TIN2014-60844-R y RYC-2013-13355). F. Palomo-Lozano ha sido parcialmente financiado por TIN2014-60844-R y TIN2015-65845-C3-3-R, y la red de excelencia TIN2015-71841-REDT (SEBASENet).

Referencias

1. Bagnall, A., Rayward-Smith, V., Whittley, I.: The next release problem. Information and Software Technology 43(14), 883–890 (2001)
2. Callahan, C.D., Cooper, K.D., et al.: ParaScope: A parallel programming environment. J. of High Perf. Comp. Apps. 2(4), 84–99 (1988)
3. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011)
4. Cortes, C., Vapnik, V.: Support-vector networks. Ma. Learn. 20(3), 273–297 (1995)
5. Dean, J., S, G.: Mapreduce: Simplied data processing on large clusters. In: Sixth Symposium on Operating System Design and Implementation). pp. 137–150 (2004)
6. Harman, M., Krinke, J., Medina-Bulo, I., Palomo, F., Ren, J., Yoo, S.: Exact scalable sensitivity analysis for the next release problem. ACM Transactions on Software Engineering and Methodology 23(2), 19:1–19:31 (2014)
7. Irigoin, F., Jouvelot, P., Triolet, R.: Semantical interprocedural parallelization: An overview of the PIPS project. In: Conf. on Supercomp. pp. 244–251. ACM (1991)
8. Keckler, S., Olukotun, K., Hofstee, H. (eds.): Multicore Processors and Systems. Springer US (2009)
9. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
10. Koziel, S., Yang, X. (eds.): Computational Optimization, Methods and Algorithms. Springer Berlin Heidelberg (2011)
11. Nisbet, A.: GAPS: A compiler framework for genetic algorithm (GA) optimised parallelisation. In: High-Perf. Comp. & Networking. pp. 987–989. Springer (1998)
12. Pinel, F., Dorronsoro, B.: Savant: Automatic generation of a parallel scheduling heuristic for map-reduce. Int. J. of Hybrid Intelligent Systems 11(4), 287–302 (2014)
13. Walsh, P., Ryan, C.: Paragen: a novel technique for the autoparallelisation of sequential programs using GP. In: Proceedings of the First Annual Conference on Genetic Programming. pp. 406–409. MIT Press (1996)
14. Weise, T., Tang, K.: Evolving distributed algorithms with genetic programming. IEEE Transactions on Evolutionary Computation 16(2), 242–265 (2012)
15. Williams, K.P.: Evolutionary algorithms for automatic parallelization. Ph.D. thesis, University of Reading (1998)