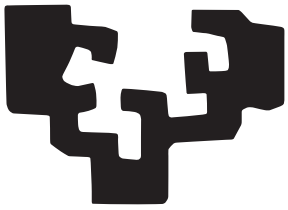
eman ta zabal zazu



Universidad del País Vasco

Euskal Herriko Unibertsitatea

Konputazio Zientziak eta Adimen Artifizialaren Saila

Departamento de Ciencias de la Computación e Inteligencia Artificial

Development of hybrid metaheuristics based on instance reduction for combinatorial optimization problems

by

Pedro Pinacho Davidson

Supervised by Christian Blum and José A. Lozano

Dissertation submitted to the Department of Computer Science and Artificial Intelligence of the University of the Basque Country (UPV/EHU) as partial fulfilment of the requirements for the PhD degree in Computer Science

Donostia - San Sebastián, Abril 2017

al Tata por siempre estar...
a Ingrid, y nuestros hijos Diego y Joaquín.

Agradecimientos

Antes que nada, quisiera agradecer a mis directores y guías *Christian Blum* y *José Antonio Lozano*, pues sin su consejo, empuje, confianza y apoyo ante todo, este trabajo nunca habría finalizado. Muchas gracias de todo corazón.

También debo agradecer a la *Universidad Santo Tomás*, por brindarme financiamiento durante los años de estudio, sin esta ayuda la tarea tampoco hubiese sido posible.

Por otro lado también le ofrezco mi gratitud al profesor *Ricardo Contreras* de la Universidad de Concepción, por facilitar las condiciones para la realización de mi pasantía de investigación y además por siempre recibirme con una taza de café y grata conversación.

Gracias a mis padres por el apoyo en estos infinitos años de formación, a mi esposa e hijos por el tiempo que les robé y que espero saber compensar, a mis amigos por soportar mi ansiedad y monotemáticas conversaciones, en especial en los últimos meses. Finalmente a Leticia y Josu por darme su apoyo a la distancia mostrándome su camino recorrido.

Eskerrik asko.

Contents

Part I

Metaheurísticas híbridas basadas en reducción de instancias

1

Algoritmos metaheurísticos

Las ciencias de la computación han enfrentado desde su concepción una gran cantidad de problemas complejos de optimización. Éstos al ser generalizados y formalizados permiten con su progresiva solución superar una gran variedad de desafíos en todos los campos de la actividad humana. La dificultad radica en que muchos de estos problemas tienen un carácter combinatorial y no permiten su resolución directa con *métodos exactos* más que en casos donde el tamaño del espacio de soluciones a explorar es pequeño, lo que muchas veces no coincide con el escenario real de aplicación, y por tanto resulta inviable.

En los casos, donde el tamaño del problema no permita la exploración exhaustiva del espacio de soluciones por medio de métodos exactos, se puede recurrir a *métodos aproximados*. Éstos, si bien no garantizan encontrar una solución óptima, muchas veces pueden encontrar soluciones de alta calidad, y por lo tanto, su utilización puede asociar un alto impacto industrial, ecológico o social. Es justamente desde esta perspectiva, que el presente trabajo explora y presenta nuevas formas para enfrentar *problemas de optimización combinatorial*.

Los problemas de optimización combinatorial (CO) $\mathscr{P} = (P, f)$, fueron definidos en un trabajo fundacional de Papadimitriou y Steiglitz [93] como un conjunto finito de objetos P y una función objetivo $f : P \mapsto \mathbb{R}^+$ que asigna un valor de costo no-negativo a cada objeto $s \in P$. Así, el proceso de optimización consiste en encontrar un objeto $s*$ de costo mínimo. En este contexto, debe notarse que minimizar una función objetivo f es lo mismo que maximizar la función $-f$. Ésta es la razón por la cual todo problema de optimización combinatorial puede ser descrito como un proceso de minimización.

El enfoque más sencillo dentro de los métodos aproximados, es la aplicación de *técnicas heurísticas* que construyan progresivamente una solución al problema en cuestión, como lo realizan las *heurísticas constructivas*, o bien a través de la búsqueda sistemática dentro de la vecindad de una solución previamente determinada, como se realiza en el caso de las heurísticas de *búsqueda local*.

Si bien las heurísticas se caracterizan por encontrar soluciones rápidamente, tienden a obtener soluciones mediocres debido a la imposibilidad de eludir óptimos locales, esto se debe principalmente a que las heurísticas no proveen de mecanismos que les permitan escapar de estas condiciones como sí lo tienen las técnicas *metaheurísticas*, las cuales son introducidas a continuación.

1.1 Metaheurísticas

Los algoritmos metaheurísticos (MH) combinan heurísticas constructivas y/o de búsqueda local con otras ideas dentro, de un *framework de control* de mayor nivel que supervisa el proceso de optimización llevado a cabo. Esto lo realizan pudiendo utilizar la información proveniente de las distintas subtécnicas que la componen y fases del algoritmo.

Las metaheurísticas poseen prestaciones que les permiten hacer frente a los óptimos locales, logrando con ello mejorar sustantivamente los resultados de las técnicas heurísticas. El origen de las metaheurísticas se encuentra en el dominio de las comunidades de Inteligencia Artificial y la Investigación de Operaciones [103][54][15] en los años '70. Una parte de ellas están inspiradas en procesos naturales como la evolución de las especies o el comportamiento de las hormigas durante la búsqueda y recolección de alimento, y la otra responde a una simple extensión de heurísticas como *Greedy* o búsqueda local. Otras características inherentes a las metaheurísticas son el uso de componentes estocásticos y la posesión de múltiples parámetros que controlan su comportamiento, que deben ser ajustados de forma *ad-hoc* en cada problema abordado.

Una forma común de clasificar las metaheurísticas es distinguiendo las basadas en una única solución *(single solution based metaheuristics)* de aquellas basadas en un conjunto o población de ellas *(population based metaheuristics)* [19]. En forma simplificada se puede establecer que una MH es exitosa, si lograr balancear correctamente los dos aspectos del proceso de búsqueda en el espacio de soluciones. El primero de ellos tiene que ver con la *intensificación* o explotación de las mejores soluciones en zonas confinadas del espacio de soluciones, mientras el segundo guarda relación con la *diversificación* o exploración de diversas zonas, en búsqueda de nichos potenciales de nuevas soluciones potencialmente de calidad. Un factor diferenciador entre las MH es la forma como se realiza este balance [5]. En términos de las taxonomía provista se puede afirmar que las MH simples basadas en una única solución se orientan más hacia la explotación de soluciones, mientras que las MH basadas en población de soluciones se caracterizan por su capacidad de exploración del espacio de búsqueda.

Algunos ejemplos prominentes de metaheurísticas basadas en un única solución son búsqueda de vecindad variable *(variable neighborhood search)*, búsqueda local iterativa *(iterated local search)*, recocido simulado *(simulated*

annealing) y búsqueda Tabú *(tabu search)*, el procedimiento de búsqueda aleatorizada voraz adaptativa *(GRASP)*, algoritmos voraces iterados *(Iterated Greedy Algorithm)*. Mientras que los exponentes más relevantes de las MH basadas en población de soluciones lo constituyen la familia de algoritmos evolutivos *(evolutionary algorithms)* y la optimización por colonias de hormigas *(ant colony optimization)*.

1.1.1 Metaheurísticas basadas en solución única

Esta clase de metaheurísticas también son llamadas *métodos de la trayectoria*, debido que parten desde una solución inicial y se desplazan desde esta describiendo una trayectoria sobre el espacio de búsqueda. Algunos de estos métodos por consiguiente pueden ser vistos como una extensión inteligente de las técnicas de búsqueda local. A continuación se realiza una breve descripción de las ideas básicas bajo las este tipo de MH.

Procedimiento de búsqueda aleatorizada voraz adaptativa (GRASP)

La metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP), corresponde a una de las metaheurísticas más simples, la cual conjuga heurísticas constructivas y de búsqueda local. En términos generales GRASP contempla dos etapas, en la primera se utiliza una heurística greedy aleatoria, con la cual se construye una solución para la instancia del problema abordado. En cada paso de construcción de una solución llevada a cabo por el algoritmo greedy aleatorio se selecciona un nuevo componente de solución desde una lista restringida de candidatos. La longitud de esta lista está determinada por un parámetro que llamaremos α, en caso de que $\alpha = 1$ la heurística se transforma en una heurística greedy (determinista), mientras que si α toma la longitud del tamaño de componentes de solución disponibles, conllevará a la generación de una solución completamente aleatoria.

La segunda etapa consiste en la aplicación de la técnica de búsqueda local sobre la solución provista previamente por la técnica greedy aleatoria, con esto se buscan soluciones de mejor calidad en la vecindad de la solución original. El algoritmo itera entre estos dos pasos hasta cumplir con algún criterio de término. Es interesante notar el bajo consumo de memoria de esta técnica, como también el hecho de que arranca con una nueva solución en cada iteración *(multi-start)*. Festa y Resende revisan extensamente GRASP en [44] y su aplicación sobre problemas de optimización combinatorial en [45], por más antecedentes se recomienda revisar [104].

Algoritmos voraces iterados (IG)

Los algoritmos voraces iterados *(Iterated Greedy Algorithm (IG))* [108], tiene la fortaleza de ser muy eficiente en términos computacionales debido a la

simpleza su algoritmo. La idea subyacente es bastante sencilla. *IG* comienza generando una solución inicial, posteriormente en cada iteración del algoritmo la solución es parcialmente destruida en forma probabilística. Esta destrucción consiste en la remoción de componentes de solución. La probabilidad de eliminación de componentes está determinada en función de la potencial utilidad de estos. Luego de la etapa de destrucción se realiza un proceso de regeneración usando posiblemente la misma técnica greedy de construcción ya empleada en la fase previa. Es admisible además utilizar optativamente luego de la reconstrucción una búsqueda local. Al finalizar cada iteración se determina la aceptación de la solución generada como solución actual, en base a un criterio que por un extremo puede aceptarla sólo si ésta significa una mejora respecto a la actual, y por otro aceptarla siempre; en la práctica se establecen criterios intermedios, los cuales pueden incluso tener un carácter dinámico. Este algoritmo itera sobre las etapas de destrucción, reconstrucción y verificación de aceptación mientras no se cumpla un criterio de término establecido.

Búsqueda local iterativa (ILS)

La búsqueda local iterativa *(Iterated Local Search (IG))*, fue introducida por Stützle en [112], corresponde a una idea simple con cierta similitud a GRASP; pero con la diferencia de que en ILS no se parte de una solución independiente con cada iteración del algoritmo. ILS produce una solución inicial mejorada con búsqueda local, que es posteriormente modificada iterativamente a través de un proceso de perturbación, el cual consiste en modificar componentes de la solución relacionada con un óptimo local. Esto depende lógicamente del problema. Las perturbaciones pueden ser de tamaño fijo o adaptativo. Este proceso que considera la historia de búsqueda realizada debe ser equilibrado, puesto que de ser muy débil no permite escapar de los puntos de atracción de la solución actual, y por otro lado al ser muy fuerte tendría el mismo efecto de un re-arranque aleatorio. De forma similar a los algoritmos voraces iterados, cada iteración finaliza con la aplicación de una función que verifica la aceptación de la nueva solución creada como solución actual. Esta función también puede considerar la historia del proceso de búsqueda. Una revisión completa de ILS y sus aplicaciones está disponible en [77].

Recocido simulado (SA)

El Recocido simulado *(Simulated Annealing (SA))* es sin lugar a dudas una de las metaheurísticas más antiguas [84], y una de las primeras en considerar un mecanismo específico para escapar de mínimos locales durante el proceso de optimización. Esta técnica imita el proceso de recocido de metales y vidrio desde estados de alta energía que al ser enfriados con un plan adecuado se estabilizan en configuraciones de baja energía. Este descenso controlado de la temperatura permite reducir las tensiones internas de los materiales tratados a través de un proceso de reordenamiento molecular que disminuye en intensidad

a medida que baja la temperatura. En términos algorítmicos esto se recrea en (SA) de la siguiente forma: primero se genera una solución inicial para el problema planteado y una temperatura inicial alta, luego a medida que esta temperatura decrece se selecciona de forma iterativa una solución en la vecindad de la actual y se acepta como nueva solución actual considerando lo siguiente: si la solución es mejor a la presente siempre se acepta, mientras que si esta es peor es aceptada probabilísticamente en base a la temperatura actual y la diferencia de calidad de la solución respecto a la vigente, lo cual se denomina "movida cuesta arriba". La temperatura se baja de forma planificada a medida que se itera –disminuyendo con ello la probabilidad de una "movida cuesta arriba"– hasta alcanzar algún criterio de término. Esta técnica ha tenido un amplio estudio y desarrollo de variaciones, lo cual se puede verificar en la siguiente bibliografía [30, 47, 72, 74, 114, 25].

Búsqueda Tabú (TS)

La búsqueda tabú *(Tabu Search (TS))*, es una metaheurística muy antigua [55], la cual hace uso explícito del historial de búsqueda para escapar de mínimos locales, además de como mecanismo para controlar la exploración del espacio de búsqueda. Esto lo logra utilizando el recurso memoria a través de una estructura llamada *lista Tabú*, que almacena atributos de las soluciones ya visitadas, e impide a la metaheurística volver a seleccionarlas mientras estén dentro de tal lista. Desde este punto de vista, la lista genera una vecindad restringida donde la metaheurística puede realizar la exploración en cada iteración, y desde donde se selecciona la mejor solución disponible y se visita. La lista Tabú al estar llena, descarta el registro de las soluciones más antiguas visitadas. Debe considerarse que, si la lista tuviese longitud infinita, se bloquearía permanentemente la posibilidad de explorar zonas del espacio de búsqueda. La determinación de la longitud de la lista Tabú es muy importante, puesto que si el tamaño es pequeño permite que el algoritmo realice una exploración confinada en una zona de el espacio de búsqueda. Por otro lado, si la lista es grande la búsqueda se diversifica debido a la gran cantidad de zonas prohibidas de visitar. Es posible encontrar implementaciones donde el tamaño de la lista Tabú es determinado de forma dinámica. Una descripción extensa de esta técnica se encuentra en [54] y [21].

Búsqueda de vecindad variable (VNS)

Este algoritmo fue propuesto por Hansen y Mladenvic [85, 86] y puede ser visto como una variante probabilística de *(Variable Neighborhood Descent (VND))*, teniendo también al igual que éste, estrategias para conmutar entre distintas funciones de vecindad, pero en el caso de la búsqueda de vecindad variable *(Variable Neighborhood Search (VNS))*, se realiza de forma controlada para diversificar la búsqueda y escapar de mínimos locales. VNS requiere un conjunto finito de funciones de vecindad para el problema como entrada. Al entrar

en operación esta metaheurística genera una solución inicial y luego en cada iteración realiza tres operaciones *sacudir, búsqueda local y mover*. Sacudir está referido a elegir una solución aleatoria de la $k-$esima vecindad de la solución actual, para luego generar una búsqueda local sobre esta solución, si el resultado obtenido es mejor que la solución actual, la nueva solución reemplaza a la preexistente, en este caso se reinicia k referenciando la primera función de vecindad, sino se procede a la siguiente función disponible, incrementando k. Una revisión de VNS y sus extensiones está disponible en [60].

1.1.2 Metaheurísticas basadas en población de soluciones

Este tipo de MH, a diferencia de los métodos de trayectorias, utilizan un conjunto de soluciones que pueden ser recombinadas o alteradas principalmente motivado por mecanismos sintetizados de la teoría de la evolución de Darwin *(Algoritmos Evolutivos)* , o bien buscan explotar las interacciones entre estas soluciones buscando el desarrollo de un *proceso inteligente de búsqueda* inspirado en la etología, sin un mayor control central sino centrado en la autoorganización *(Swarm Intelligence)*. Los principales exponentes de este tipo de MH se presentan a continuación.

Optimización basada en colonia de hormigas (ACO)

La Metaheurística *Ant Colony Optimization* (ACO) fue introducida por Dorigo [37, 34, 36], esta tiene una inspiración biológica, en particular esta observa la conducta y mecanismos usados por las colonias de hormigas para la búsqueda de la ruta más corta entre fuentes de alimento y su colonia. El algoritmo trabaja sobre el conjunto de *componentes de solución* para el problema abordado, a los cuales les asocia valores de feromonas. Estos valores se establecen a través de un modelo probabilístico parametrizado que resulta ser el componente más importante de ACO. El funcionamiento de este algoritmo se basa en dos pasos:

1. Se construye un conjunto de soluciones candidatas usando modelos de feromonas, esto es, a través de una distribución probabilística parametrizada sobre el espacio de búsqueda.
2. Las soluciones candidatas son usadas para actualizar los valores de feromonas, permitiendo orientar las búsqueda hacia futuras soluciones de alta calidad.

La actualización de los valores de feromonas asociada a los componentes de solución permite concentrar la búsqueda en regiones del espacio donde posiblemente se encuentren soluciones de alta calidad, esto está basado en la premisa de que buenas soluciones contienen buenos componentes. Las variantes más populares de ACO son MAX-MIN Ant System (MMAS) [113] y Ant Colony System (ACS) [35].

Algoritmos evolutivos (EA)

Estas Metaheurísticas están inspiradas en la evolución natural de las especies y en particular el principio de selección natural. Los algoritmos evolutivos *(Evolutionary Algorithms (EA))*, a diferencia de las otras metaheurísticas revisadas y en similitud a ACO no se basan en una única solución actual de trabajo, sino que en un conjunto de ellas, a este conjunto se le denomina población de individuos. Esta población corresponde a un conjunto de soluciones candidatas.

Operativamente, la población se inicializa de forma aleatoria al comienzo del algoritmo, para luego hacerla evolucionar generación tras generación a través de funciones de recombinación o *crossover* que se encargan de generar nuevas soluciones, esto se logra mezclando soluciones previas seleccionadas probabilísticamente dependiendo de su calidad. Para realizar esto, se utiliza una medida de calidad basada en la función objetivo del proceso de optimización, esta medida de calidad asociada a cada individuo de la población se le denomina *fitness*. Los *EAs* también implementan un operador específico para escapar de mínimos locales, el cual introduce variaciones arbitrarias *(mutación)* en las soluciones con independencia de las características heredadas de las soluciones padres. Este operador es usado con baja probabilidad, debido a que una alta probabilidad puede destruir las mejoras logradas por el proceso de evolución. Dentro de los algoritmos evolutivos, podemos ver agrupados a técnicas como *algoritmos genéticos* [62], *estrategias evolutivas* [102], *programación evolutiva* [48] y *programación genética* [73].

Una ventaja importante de estas técnicas, es la eficacia que presentan para explorar ampliamente el espacio de búsqueda, mientras que su principal debilidad radica en que le resulta dificultoso realizar mejoras finas de las soluciones. Una técnica complementaria utilizada para mejorar el desempeño ante estas dificultades, lo constituye el realizar búsqueda local en la generación de nuevos individuos, a lo que también se le denomina *algoritmo memético* [87].

1.1.3 Otras Metaheurísticas

Existen otras metaheurísticas con una fuerte inspiración biológica. Este es el caso de *Particle Swarm Optimization (PSO)* [69, 29], el cual se motiva en comportamiento de bandadas de aves y cardúmenes de peces. Esta técnica ha sido aplicada con éxito por Kennedy y Eberhart [68] para la optimización continua de pesos de redes neuronales. Otra técnica en esta categoría es la colonia de abejas artificiales *(Artificial Bee Colony (ABC))*. Esta técnica basada en población se basa en tres tipos de agentes-abejas (empleadas, espectadoras, exploradoras) que colaboran en la búsqueda y recolección de polen, la cual ha sido usada con éxito para abordar algunos problemas de optimización combinatoria [92, 105].

Otra metaheurística antigua no muy activa en la literatura actual es *Guided Local Search (GLS)* [119, 120]. Esta resulta interesante por poseer

una estrategia distinta para escapar de óptimos locales, la particularidad de la misma radica en que se basa en una modificación dinámica de la función objetivo basado en el historial de búsqueda, lo que genera un cambio en el escenario de búsqueda. Otras técnicas metaheurísticas poco conocidas son *Squeaky Wheel Optimization* [67], *Extremal Optimization* [16], y *Great Deluge Algorithm* [39]. La revisión presentada no es exhaustiva, para una discusión más completa sobre metaheurísticas, el lector puede revisar [13] [19] [14] [115].

1.2 Metaheurísticas híbridas

Las metaheurísticas híbridas son un campo de investigación relativamente reciente [14]. En este se explora la mezcla de ideas de distintas metaheurísticas y métodos exactos, buscando con esto sacar beneficio del ejercicio de complementar una técnica con distintas estrategias de optimización. El desarrollo de metaheurísticas híbridas no es trivial y requiere experticia en distintas técnicas de optimización. Por otro lado, en su aplicación estas técnicas tienden a no ser generalizables con facilidad, mostrando excelentes resultados en ciertas áreas de problemas y resultados mediocres en otras.

En los primeros años del desarrollo de métodos metaheurísticos se aplicaron en comunidades científicas sin mayor interacción y/o vinculación entre ellas de forma pura. Esto estuvo justificado por los buenos resultados obtenidos en relación al estado del arte imperante. Posteriormente, cuando las mejoras comenzaron a ser limitadas y se encontraron los límites al rendimiento de estos algoritmos, nació la necesidad de explorar variantes que integraran las características de las propuestas preexistentes dando origen a esta área de desarrollo. Actualmente existe mucho interés en metaheurísticas híbridas, lo cual se refleja en la existencia de conferencias y workshops como CPAIOR [95, 118, 75], Hybrid Metaheuristics [6, 7], Matheuristics [79, 59, 38, 80], la publicación de textos especializados como [27, 14], y la existencia de herramientas relacionadas, entre las cuales destaca ParadisEO [22].

Existen distintos tipos de hibridación, a continuación se establecen las ideas principales que ellos instrumentalizan.

Representación de Soluciones Incompletas y Decodificadores

En este caso se representan las soluciones candidatas al problema abordado dentro de la metaheurística de una forma indirecta, un ejemplo del uso de representaciones indirectas lo constituyen los algoritmos genéticos, en estos se utiliza ya sea por la necesidad de facilitar la acción de operadores de recombinación como crossovers o mutaciones, o por el hecho de mantener propiedades deseables. En estos casos se establece un decodificador, esto es una función que realice la transformación de esta solución al dominio de búsqueda real de donde proviene su complejidad. Este tipo de metaheurísticas híbridas potencia el desarrollo de *decodificadores inteligentes* que funcionen con soluciones

parciales o incompletas, las cuales puedan ser completadas por una técnica complementaria. Esta técnica puede basarse en una aproximación de programación lineal entera [4], programación dinámica [71] , o programación basada en restricciones [106] entre otras. Un ejemplo de este tipo de híbrido es la combinación de VNS con ILP utilizado para el problema *generalized spanning tree problem* el cual puede ser revisado en [64].

Hibridación basada en Búsqueda en grandes vecindades

La metaheurística basada en *Large Neighborhood Search (LNS)*, se sustenta en que dada una solución para una instancia del problema dado, se itera sobre ella realizando sucesivos pasos de *destrucción-reconstrucción*. La destrucción remueve aleatoriamente o selectivamente a través de una heurística partes de la solución original, generándose una solución parcial. Sobre esta solución parcial se aplica una técnica posiblemente exacta en búsqueda de la mejor solución posible que contiene la solución parcial, esto es, dentro de la *gran vecindad* definida por el proceso previo de destrucción. Un ejemplo de este tipo de hibridación fue propuesta por Shaw [110] donde se utiliza programación basada en restricciones para explorar grandes vecindades aplicado a problemas de enrutamiento vehicular.

Construcción de soluciones paralelas, no-independientes dentro de metaheurísticas

Muchas heurísticas y metaheurísticas denominadas constructivas se basan en la construcción paso a paso de soluciones, este es el caso de la conocida heurística *greedy*. Por otro lado ACO y GRASP usan por su parte, mecanismos de construcción de soluciones secuenciales, independientes y probabilísticos que les permiten también determinar en cada paso el siguiente componente de solución a considerar. La probabilidad para la determinación de estos pasos constructivos considera una función heurística (greedy comúnmente) y posiblemente el historial de búsqueda, lo cual define una distribución de probabilidad sobre el espacio de búsqueda, y constituye lo que llamaremos *conocimiento del problema primario*. Este tipo de hibridación se basa en establecer y usar el *conocimiento del problema secundario* para mejorar el desempeño de la búsqueda. Este conocimiento proviene de otros métodos de búsqueda en árbol como *branch-and-bound* y derivados como *beam search*, los cuales otorgan capacidades para limitar la búsqueda a través de la evaluación de soluciones parciales o extender las soluciones con variantes permitiendo con esto paralelizar el proceso de búsqueda. Un ejemplo de este tipo de metaheurísticas que usa conocimiento complementario del problema y construcción paralela no independiente de soluciones es *BEAM-ACO* [8] aplicado a *open shop scheduling problem*.

Hibridación basada en registros de solución completos

Este tipo de hibridación trata de potenciar las capacidades de algoritmos evolutivos, lo que se logra por medio de la implementación de un árbol de exploración complementario basado en *branch-and-bound*. Este árbol permite tener prestaciones adicionales como una *mutación informada*, brindando la posibilidad de evitar revisitar soluciones ya descubiertas y apoyar la evaluación de soluciones costosas. Esta técnica es en especial útil cuando un EA enfrenta espacios de búsqueda pequeños sin estar provisto de operadores de variabilidad que entreguen innovación suficiente, posiblemente causado porque el algoritmo fue calibrado para mejorar el desempeño en la búsqueda fina de soluciones de alta calidad. Un ejemplo de este tipo de hibridación puede ser revisado en [65, 66]. donde se aplica la técnica para enfrentar el problema *generalized minimum spanning tree*.

Hibridación basada en reducción de instancias

Los *solvers* para modelos de programación entera representan una excelente herramienta para distintos problemas de optimización combinatorial, este es el caso de herramientas como CPLEX[1] y GUROBI[2] . Lamentablemente éstas no resultan útiles en instancias de problemas medianos y grandes donde sus soluciones decaen en calidad o simplemente no se obtienen. Para estos casos la propuesta de hibridación recae en disminuir el tamaño de la instancia de forma inteligente, para lograr un tamaño de problema que haga útil la aplicación de la herramienta *solver*, manteniendo la posibilidad de obtener soluciones de alta calidad e incluso soluciones óptimas.

1.3 Hibridación basada en reducción de instancias

La presente disertación se basa justamente en el desarrollo de metaheurísticas basadas en reducción de instancias de problema. En este contexto se muestra como progresivamente se desarrollaron distintas heurísticas, modelos ILP, metaheurísticas y estrategias que fueron validados sobre distintos problemas de optimización combinatorial para de esta forma dar sustento a una propuesta de *framework de optimización general*, el cual constituye el más importante resultado de este proceso de investigación.

1.3.1 Trabajos relacionados

El uso de técnicas de reducción de instancias se aplica de diferentes formas sobre metaheurísticas en la literatura. Una de estas es dentro de un operador

[1] IBM ILOG CPLEX Optimization Studio es un producto de IBM ®
[2] Gurobi Optimizer es un producto de Gurobi Optimization ®

de la MH, Aggarwal et. al. aplica esta idea en algoritmos evolutivos [1], donde se unen dos o más soluciones *(solution merging)* generando un operador de cruza *(crossover)* optimizado. Esto posteriormente fue aplicado al problema *quadratic assignment* [2].

Rothberg [107] establece una integración entre algoritmos evolutivos y MIP *(mixed integer programming)*, en esta propuesta se muestra como a intervalos regulares de iteración del EA se aplica una heurística de ramificación y poda *(branch-and-bound)* y luego se realiza una unión *(merge)* de las soluciones, con lo cual se fijan las variables de la solución comunes a los padres. Al fijar los valores se produce una instancia de problema reducido, la cual es abordada por un *solver* MIP. En este contexto la mutación opera de una forma relativamente similar, seleccionando sobre un padre un conjunto aleatorio de variables a ser fijadas, posteriormente se aplica también un *solver* MIP. También se aborda la selección de padres para operadores de cruza en EA, lo cual se prueba sobre los problemas *single machine scheduling*, *optimal linear arrangement*, *traveling salesman problem* [121], y en *k-cardinality tree problem* [10]. Un enfoque distinto es provisto en [81, 82], en donde se utiliza ACO para generar múltiples rutas para el problema *vehicle routing* considerando restricciones de factibilidad, entonces se aplica un solver exacto sobre un problema *set-partitioning* relajado con el objetivo de seleccionar un subconjunto de rutas, éstas son utilizadas para la generación de nuevas rutas en la siguiente iteración.

La aplicación de MIP *solvers* sobre instancias reducidas de problemas si bien no figura sistemáticamente en la literatura; se puede apreciar en [28] donde se propone un algoritmo para resolver un problema de planificación *(scheduling)* a ser usado en el proceso de fabricación de fibra óptica. Éste utiliza en la primera fase una técnica de búsqueda local (LS) para crear soluciones factibles, usando posteriormente un *solver* MIP y obteniendo como resultado soluciones al menos tan buenas como las encontradas previamente por LS. Otra propuesta en dos fases en presentada en [3] y [31] donde se aborda el problema del vendedor viajero (TSP). Primero se generan soluciones de alta calidad por medio de una metaheurística, las cuales son combinadas en una sub-instancia de problema reducido a la que se le aplica en la segunda fase un *solver* exacto.

Klau propone en [70] una propuesta para enfrentar el problema *prize-collecting Steiner tree*. En este se reduce la instancia de forma que aún contenga la solución óptima, luego se aplica un algoritmo memético a la subinstancia reducida. Finalmente se usa un solver MIP sobre la unión de todas las soluciones de la primera y última población del algoritmo memético.

Dentro de las metaheurísticas basadas en reducción de instancia, es importante destacar el framework *Generate-and-Solve* (GS) debido a la cercanía al trabajo propuesto en esta disertación. GS propone una metaheurística de dos fases, en la primera un componente llamado SRI *(Solver of Reduced Instances)*, el cual corresponde a un método exacto se aplica sobre una subinstancia del problema original, de tal forma que cualquier solución obtenida

también es una solución de la instancia original de problema. En la segunda fase un componente metaheurístico llamado GRI *(Generator of Reduced Instances)* se utiliza para generar subinstancias del problema que contengan soluciones de alta calidad. GS es aplicado a problemas de corte, empaque y carga [90, 91, 89, 97, 109] y problemáticas asociadas a redes inalámbricas [33, 32, 98]. GS considera retroalimentación de información para el control de su funcionamiento desde el componente SRI hacia el GRI, lo cual también contempla el control del tamaño de la subinstancia generada, a través de un *operador de control de densidad* [97]. Cabe destacar que el componente GRI ha sido implementado a través de algoritmos evolutivos [97, 98] y recocido simulado [32, 109].

1.4 Organización de la disertación

Esta disertación está estructurada en base a una compilación de publicaciones que enfrentan la temática del uso de reducción de instancias de problemas de distintas formas y en distintos contextos. Esto se presenta en el texto en dos partes. La primera parte establece los fundamentos de las técnicas utilizadas (Capítulo 1), los contextos de problemas abordados (Capítulo 2) y una descripción de las contribuciones realizadas (Capítulo 3). La Parte I cuenta también con las conclusiones de la investigación realizada y los trabajos proyectados en la línea de investigación (Capítulo 4), además de una descripción de las publicaciones realizadas durante todo el proceso de investigación (Capítulo 5). En la Parte II se entrega *in extenso* las publicaciones seleccionadas para esta disertación.

2

Problemas combinatoriales abordados

La solución de los problemas de optimización combinatorial, es uno de los desafíos más importantes de la matemática. Considerando la complejidad de problemas [52] de forma simplificada, tenemos dos clases, la primera, la *(Clase P)* es aquella de todos aquellos problemas que pueden ser resueltos en tiempo polinomial. Mientras que para la segunda clase *(Clase NP-hard)* se desconoce un algoritmo que resuelva estos problemas de forma eficiente (en tiempo polinomial). Muchos de los CO pertenecen a esta segunda clase, por lo cual no pueden hasta el momento ser resueltos de forma exacta de forma eficiente, más que en instancias de de cierto tamaño, que depende del problema. En particular las propuestas algorítmicas presentadas en esta disertación abordan un conjunto de problemas CO *(NP-hard)*, los cuales son descritos a continuación.

2.1 Minimum Common String Partition (MCSP)

Los problemas de optimización relacionados con cadenas de texto *(strings)*, como resultan ser las representaciones de cadenas de ADN son muy comunes en el campo de la bioinformática. Ejemplos de esto son *el problema de selección de cadenas (string selection problem)* [83, 88, 94], *el problema la subsecuencia más larga común (longest common subsequence problem)* y sus variantes [63, 111], *problemas de alineación (alignment problems)* [56, 100] y de *búsqueda de similitud (similarity search)* [99]. Estos problemas resultan ser computacionalmente muy difíciles, muchas veces NP-Hard [52]. El problema de particiones de cadenas de texto mínimas comunes MCSP *(minimum common string partition problem)* está dentro de esta categoría. MCSP puede ser descrito de la siguiente forma: Dadas dos cadenas de texto relacionadas, las cuales deben ser particionadas en una misma colección de subcadenas, se debe minimizar el tamaño de la partición. Formalmente, dadas dos cadenas de texto s_1 y s_2, ambas de longitud n sobre un alfabeto Σ, considere que s_1 y s_2 deben estar relacionadas, esto significa que cada símbolo debe aparecer la misma cantidad de veces en ambas cadenas, lo cual implica que ambas cadenas tienen la misma

longitud. Una solución válida para el problema MCSP se obtiene por realizar la partición de s_1 en un conjunto P_1 de subcadenas no superpuestas, y s_2 en un conjunto P_2 de subcadenas con similar característica, tal que $P_1 = P_2$. Además, se busca una solución válida tal que $|P_1| = |P_2|$ es mínimo.

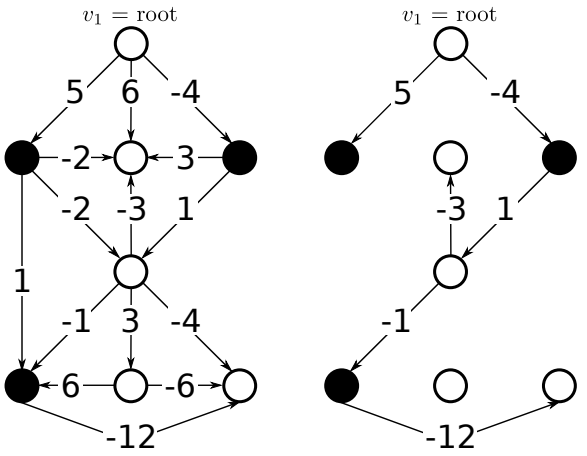
Como ejemplo se puede considerar el siguiente caso. Sean dos cadenas de ADN, $s_1 =$ **AGACTG** y $s_2 =$ **ACTAGG**, claramente s_1 y s_2 están relacionados debido a que **A** y **G** aparecen dos veces en cada cadena, mientras **C** y **T** aparecen una. Una solución trivial para este problema, puede ser obtenida al generar una partición para ambas cadenas con subcadenas de longitud 1, esto es, $P_1 = P_2 = \{\mathbf{A}, \mathbf{A}, \mathbf{C}, \mathbf{T}, \mathbf{G}, \mathbf{G}\}$, en este caso el valor de la función objetivo sería 6. Por otro lado, el valor de la función objetivo para la solución óptima es 3, y se alcanza cuando $P_1 = P_2 = \{\mathbf{ACT}, \mathbf{AG}, \mathbf{G}\}$.

El problema MCSP tiene aplicación en el campo de la bioinformática. Chen et al [24] establecen que este problema está muy relacionado con el de *ordenamiento por reversos con duplicados (sorting by reversals with duplicates)*, el cual es clave para el problema de reordenamiento de genoma.

2.2 Minimum Covering Arborescence (MCA)

El problema de cobertura mínima considerado en esta sección pertenece a la familia de problemas MWRA *(minimum weight rooted arborescence)*. En este tipo de problemas, dado un grafo dirigido acíclico con pesos enteros en los arcos –donde en algunos casos los pesos pueden estar restringidos a valores positivos, mientras en otros estos pueden ser negativos–. Se busca una solución sobre el grafo, esta corresponde a un subgrafo del grafo problema que establece una arborescencia con raíz en un nodo predefinido. Dentro de este contexto, una arborescencia con raíz es un árbol dirigido no necesariamente de cobertura, en el que todos los arcos apuntan en sentido contrario de la raíz. El objetivo consiste en encontrar dentro de las soluciones válidas, una con peso mínimo. El costo de cada solución al problema está dado por la suma del peso de sus arcos. Este tipo de problema tiene aplicación en visión computacional y planificación de la producción multi-etapas.

El problema MCA se define formalmente de la siguiente forma: Considere un grafo dirigido acíclico (DAG) denotado por $G = (V, A)$, donde $V = \{v_1, \ldots, v_n\}$ es el conjunto de n nodos y $A \subseteq \{(i, j) \mid i \neq j \in V\}$ es el conjunto de m arcos dirigidos. Se asume que v_1 es el nodo raíz. Cada arco $a \in A$ tiene asignado un peso entero $w(a) \in \mathbb{Z}$. Además el conjunto predefinido $X \subseteq V$ de los nodos del grafo problema debe ser incluido en la solución válida. De esta forma cualquier arborescencia $T = (V(T), A(T))$—donde $V(T) \subseteq V$ es el conjunto de nodos de T y $A(T) \subseteq A$ es el conjunto de arcos de T—con raíz en v_1 y con $X \subseteq V(T)$ es una solución válida al problema. Sea $\mathcal{A}$ el conjunto de todas las arborescencias. El valor de la función objetivo (esto es, el peso) $f(T)$ de una arborescencia $T \in \mathcal{A}$ se define de la siguiente forma:



(a) Ejemplo de grafo de entrada.

(b) solución óptima de valor -14.

Fig. 2.1: (a) muestra un grafo de entrada con ocho nodos y quince arcos. El nodo de más arriba es el nodo raíz (root) v_1. Además, los nodos coloreados negros forman el conjunto X, esto es, estos deben ser incluidos en cualquier solución válida. (b) muestra la solución óptima de valor -14.

$$f(T) := \sum_{a \in A(T)} w(a) \ . \tag{2.1}$$

El objetivo del problema MCA es encontrar una arborescencia $T^* \in \mathcal{A}$ tal que el peso de T^* es menor o igual al peso de cualquier arborescencia en $\mathcal{A}$. En otras palabras, el objetivo es minimizar la función objetivo $f(\cdot)$. Un ejemplo del problema MCA puede ser visto en la Figura 2.1.

2.3 Weighted Independent Domination (WID)

WID es un problema de optimización combinatorial que fue introducido en [23]. Este corresponde a una extensión del bien conocido *independent domination (ID) problem*. En este problema, dado un grafo $G = (V, E)$, V es el conjunto de nodos y E se refiere al conjunto de aristas. Una arista $e \in E$ que conecta los nodos $u \neq v \in V$, es de igual forma llamada (u, v) o (v, u). La *vecindad* $N(v)$ de un nodo $v \in V$ se define como $N(v) := \{u \in V \mid (v, u) \in E\}$, además la *vecindad cerrada* $N[v]$ de un nodo $v \in V$ se define como $N[v] := N(v) \cup \{v\}$. Por otro lado, el conjunto de aristas incidentes sobre el nodo $v \in V$ de denota como $\delta(v)$. Debe notarse, que en este contexto, una arista $e \in E$ es llamada incidente al nodo v, si v forma una de las dos puntas de e. Dado un grafo no-dirigido $G = (V, E)$, un subconjunto $D \subseteq V$ de nodos es

llamado *conjunto dominante* si cada nodo $v \in V \setminus D$ es adyacente al menos a un nodo de D, esto es, si por cada nodo $v \in V \setminus D$ existe al menos un nodo $u \in D$ tal que $v \in N(u)$. Adicionalmente, un conjunto $I \subseteq V$ es llamado un *conjunto independiente* si para cualquier par $v \neq v' \in I$, se cumple que v y v' no están conectados por una arista en G. En consecuencia, un subconjunto $D \subseteq V$ es llamado *conjunto independiente dominante* si D es ambos un conjunto independiente y un conjunto dominante. Finalmente, dado un conjunto independiente y dominante $D \in V$, para todo $v \in V \setminus D$ se define la *vecindad restringida por D*, $N(v \mid D)$ como $N(v \mid D) := N(v) \cap D$, esto es, la vecindad de v es restringida a todos sus vecinos que están en D.

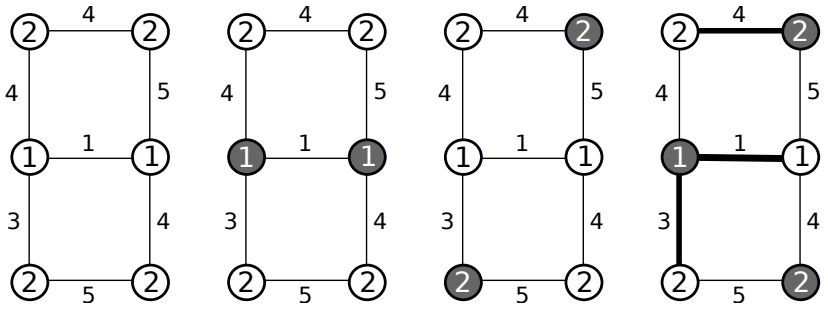
En el problema WID se tiene que dado un grafo no-dirigido $G = (V, E)$ con pesos en nodos y aristas. Más específicamente, donde a cada $v \in V$ y $e \in E$ se le asocia un peso entero $w(v) \geq 0$, $w(e) \geq 0$, respectivamente. La resolución consiste en encontrar un conjunto dominante independiente D en G que minimice la siguiente función de costo:

$$f(D) := \sum_{u \in D} w(u) + \sum_{v \in V \setminus D} \min\{w(v, u) \mid u \in N(v \mid D)\} \tag{2.2}$$

En otras palabras, el valor de la función objetivo de D es obtenido por la suma de los pesos de los nodos en D, más la suma de los pesos de las aristas de peso mínimo que conectan los nodos que no están en D con nodos dentro de D. Como ejemplo considere los grafos en la Figura 2.2. En este ejemplo el peso de los nodos está indicado dentro de éstos, y el peso de las aristas se puede visualizar junto a las mismas. Un posible grafo de entrada se muestra en la Figura 2.2a. Un conjunto dominante de peso mínimo óptimo (el conjunto de nodos grises) se muestra en la Figura 2.2b. Sin embargo, debe notarse que este conjunto no es independiente, debido a que dos nodos que forman parte del mismo son adyacentes entre si. Un conjunto independiente dominante de peso mínimo óptimo es presentado en la Figura 2.2c. Debe notarse que para ambos, el *minimum weight dominating set problem* y *minimum weight independent dominating set problem*, los pesos de las aristas no son considerados. Finalmente, la solución óptima para el problema WID se presenta en Figura 2.2d. Las aristas de peso mínimo seleccionadas para conectar nodos en D con nodos fuera de D están señaladas con líneas remarcadas. El valor de la función objetivo para este solución es 13, lo cual se forma de la composición del peso de los nodos (2 + 1 + 2) y el de las aristas (4 + 1+ 3).

2.4 k-Cardinality Tree (kCT)

El problema k-cardinality tree (kCT) fue definido en [58]. Siendo mostrado en [46][101] que corresponde a un problema NP-hard que tiene distintas aplicaciones en leasing de campos petroleros [57], distribución de instalaciones [50][49], minería de cielo abierto [96], descomposición de matrices [17][18] y telecomunicaciones [53].



(a) Grafo con pesos en nodos y aristas.

(b) Conjunto dominante de peso mínimo

(c) Conjunto independiente dominante de peso mínimo.

(d) Solución óptima para el problema WID.

Fig. 2.2: Ejemplo que relaciona el problema WID con el problema *minimum weight dominating set* y con el problema *minimum weight independent dominating set*

Técnicamente, el problema kCT puede ser descrito de la siguiente forma. Sea $G(V, E)$ un grafo no-dirigido en el cual cada arista $e \in E$ tiene un peso $w_e \geq 0$ y cada nodo $v \in V$ tiene un peso $w_v \geq 0$. Además, se denota por $\mathcal{T}_k$ el conjunto de todos los árboles de cardinalidad k en G, esto es, con exactamente k aristas. El problema consiste en encontrar un árbol de cardinalidad k, $T_k \in \mathcal{T}_k$ que minimize:

$$f(T_k) = \left(\sum_{e \in E_{T_k}} w_e \right) + \left(\sum_{v \in V_{T_k}} w_v \right) \quad , \tag{2.3}$$

donde dado un árbol T, E_T denota en conjunto de aristas de T, y V_T corresponde al conjunto de nodos de T. Una instancia al problema kCT se denota con una tupla (G, k), donde G es un grafo no dirigido con peso en las aristas, y k es la cardinalidad buscada.

La literatura presenta principalmente dos casos especiales del problema general kCT, las cuales se definen como sigue: (1) *edge-weighted* kCT (e-kCT), donde el peso de los nodos es cero y (2) *node-weighted* kCT (n-kCT), donde todos los pesos de las aristas son cero. En esta investigación fue de interés el subproblema (e-KCT), el cual fue abordado por técnicas exactas [51][26][116] y heurísticas [41][40][26]. Una de las mejores heurísticas elaboradas construye un árbol de expansión de un grafo dado, que subsecuentemente aplica un algoritmo de programación dinámica polinomial [78] que encuentra el mejor árbol de cardinalidad k en el árbol de expansión. Investigaciones posteriores se enfocan más en el desarrollo de metaheurísticas, (para ejemplos se puede revisar [9] [20] [117]) y metaheurísticas híbridas [10].

3

Contribuciones

En esta sección se establecen y describen de forma preliminar las contribuciones que sustentan esta disertación, las cuales son discutidas en extenso en las publicaciones incorporadas en el texto.

3.1 Efectos de la reducción de instancias sobre un modelo ILP para el problema MCSP

La investigación realizada comienza con la exploración del uso de modelos ILP sobre el problema MCSP, ya descrito en la sección 2.1. En este contexto se presenta el primer modelo ILP para solucionar este problema.

3.1.1 Modelo ILP propuesto

El modelo se basa en la definición de dos matrices de dimensiones $m \times n$, $M1$ y $M2$ con filas $1 \leq i \leq m$ que representan los m bloques comunes o subcadenas comunes entre ambas cadenas relacionadas consideradas en el problema. Además las columnas $1 \leq j \leq n$ representan la posición de estos bloques comunes en s_1 en el caso de $M1$ y en s_2 en el caso de $M2$, y donde n es la longitud de ambas cadenas. En general las entradas de las matrices $M1$ y $M2$ son establecidas en cero, con excepción de aquellas posiciones ocupadas por el bloque común en estas matrices. Consideremos la posición (i, j) de una matriz con la notación $M_{i,j}$ y finalmente consideremos para cada bloque común existente entre s_1 y s_2 el cual llamaremos b_i una variable binaria x_i. Considerando estas definiciones podemos expresar el problema MCSP en forma del siguiente modelo ILP, el cual denotaremos como $\textsc{Ilp}_{\text{orig}}$.

$$\min \sum_{i=1}^{m} x_i \tag{3.1}$$

sujeto a:

$$\sum_{i=1}^{m} M1_{i,j} \cdot x_i = 1 \quad \text{para } j = 1, \ldots, n \tag{3.2}$$

$$\sum_{i=1}^{m} M2_{i,j} \cdot x_i = 1 \quad \text{para } j = 1, \ldots, n \tag{3.3}$$

$$x_i \in \{0, 1\} \quad \text{para } i = 1, \ldots, m$$

La función objetivo minimiza el número de bloques comunes seleccionados, mientras la restricción (3.2) se asegura que la cadena correspondiente al bloque común seleccionado no se solapa en la cadena s_1, en tanto (3.3) se asegura lo mismo pero sobre la cadena s_2. Además se debe notar que las restricciones (3.2) y (3.3) implícitamente aseguran que la suma de las longitudes de las cadenas correspondientes a los bloques comunes seleccionados sea igual a n.

3.1.2 Evaluación de Modelo ILP sobre instancias de problema pequeñas

El modelo ILP_{orig} es evaluado inicialmente en base a un conjunto de cadenas de prueba propuesto por Ferdous y Sohel Rahman [42], el cual está orientado a la aplicación sobre el área bioinformática. Este conjunto se estructura en base a cuatro grupos de cadenas de problemas organizadas según su longitud, todas ellas basadas en un alfabeto de tamaño 4, compuesto por las letras $\{\mathbf{A}, \mathbf{C}, \mathbf{G}, \mathbf{T}\}$. Este conjunto de prueba se caracteriza por poseer 15 secuencias de ADN reales y además por el hecho que la longitud máxima de las cadenas no supera las 600 letras. El Grupo1 está compuesto por cadenas de longitud máxima de 100 caracteres, el Grupo2 por su parte tiene cadenas con longitudes entre 201 y 400 caracteres, el Grupo3 utiliza cadenas entre 401 y 600 caracteres, y finalmente el grupo Real posee cadenas con longitudes variables entre 200 y 600. En todos los casos de prueba se usó CPLEX para la resolución del modelo ILP_{orig}. Los resultados fueron comparados con lo de una implementación de la heurística *greedy*, una implementación ACO [42] y un algoritmo de búsqueda probabilística sobre árboles (TreSea) [11], que hasta el momento resultaba ser el estado del arte sobre el problema.

Los resultados logrados por ILP_{orig}, fueron destacables, el modelo resultó tener mejores resultados en todas las instancias de pruebas, superando a todos los algoritmos en la comparación y en particular a la mejor propuesta existente en la literatura (TreSea) en términos de mejora promedio (porcentual), por un 4.8% en el caso del Grupo1, 9.2% para el Grupo2, 9.7% en el caso del Grupo3 y un 9.9% para los casos de secuencias de ADN reales; no obstante

Table 3.1: Resultados de aplicar CPLEX a ILP_{orig} en el contexto de instancias de problemas grandes.

length	value	time (s)	gap	$\|B\|$
800	210	2531	10.7%	214622
1000	304	1673	26.4%	334411
1200	342	3435	22.6%	480908
1400	401	6459	24.9%	653401
1600	442	10987	24.1%	854500
1800	486	18276	24.0%	1084533
2000	n.a.	n.a.	n.a.	1335893

estos buenos resultados, es relevante constatar que ILP_{orig} sólo pudo encontrar soluciones óptimas para el Grupo1 y excepcionalmente en uno de los casos del grupo Real, lo que llevó a cuestionar la escalabilidad del rendimiento entregado por el modelo ILP a medida que se incrementa la longitud de las instancias de problema, razón por la cual se procedió a estudiar sus resultados con problemas más grandes, lo cual es revisado a continuación.

3.1.3 Evaluación del Modelo ILP sobre instancias de problemas grandes

Para lograr poner a prueba el modelo ILP propuesto sobre problemas más grandes se desarrolló un nuevo conjunto de evaluación basado en cadenas aleatorias ficticias de ADN, las longitudes de las cadenas de ADN de este nuevo conjunto son {800, 1000, 1200, 1400, 1600, 1800, 2000}. El modelo fue ejecutado sobre CPLEX con una restricción de tiempo preliminar máximo de cómputo de 3600 segundos de CPU, y con una detención forzada si el cómputo de la primera solución se proyectase más allá de las 12 horas. El resultado de la aplicación de ILP_{orig} se puede ver en la Tabla 3.1. En esta tabla, en las primeras 3 columnas se aprecia la longitud de las cadenas, la solución obtenida y el tiempo utilizado respectivamente, además en la cuarta columna, se muestra el (gap), o la brecha existente entre la solución encontrada y la cota inferior al momento del término de la ejecución del algoritmo. Por otro lado, la última columna $|B|$, entrega el tamaño del conjunto de subcadenas comunes para la instancia de problema.

Los resultados dan a conocer que el modelo ILP_{orig} se vuelve inviable de ser aplicado sobre CPLEX de forma eficiente en los tiempos considerados, puesto que desde tamaños de instancia de problema de 1400 ya se excede la cota de 3600 segundos establecida, mostrando una tendencia exponencial que impide que se obtengan resultados dentro de la cota de 12 horas para el caso de longitud 2000.

3.1.4 Reducción heurística del tamaño de una instancia de problema

Con el objetivo de superar el problema de desempeño de la aplicación de modelos ILP sobre instancias de problemas grandes, se opta por probar el comportamiento del modelo ILP sobre una instancia reducida de problema. Para esto se analizó la distribución de los bloques comunes en términos de su longitud. La idea intuitiva en este contexto es que mejores soluciones requieren menos bloques y más grandes. El estudio arrojó de forma consistente e independiente del tamaño de problema que alrededor del 75% de los bloques comunes disponibles en $|B|$ son de tamaño 1, y por otro lado sólo una pequeña porción de estos participan de las soluciones óptimas. Considerando esto, se desarrolló una heurística determinista basada en MIP que permita reducir el espacio de componentes de solución (bloques comunes) privilegiando el uso de cadenas más largas. Esto se logra a través de la aplicación de dos fases de ILP, ambas basadas en el modelo ILP_{orig} utilizándolo con variantes que permitan en una primera fase construir una solución parcial basada en bloques comunes extensos, y en una segunda fase completar la solución parcial agregando los bloques comunes faltantes.

En la primera fase de la heurística el conjunto B debe poseer todos los bloques disponibles. Para este propósito sea $B_{\geq l}$ (donde $l \geq 1$) denota el subconjunto de B que contiene todos los bloques comunes b_i de B con $|t_i| \geq l$–donde t_i es un string común que se encuentra tanto en s_1 como en s_2–, esto es, todos los bloques con cadenas más largas o iguales a l. Debe notarse en este contexto que $B_{\geq 1} = B$. Además $|B_{\geq 1}| \geq |B_{\geq 2}| \geq |B_{\geq 3}| \geq \ldots \geq |B_{\geq \infty}|$. Sea $l_{\max}$ el más pequeño valor para l tal que $|B_{\geq l_{\max}}| > 0$. Obsérvese que $B_{\geq l_{\max}}$ sólo contiene bloques comunes con las cadenas más largas. Estableciendo un valor específico para l de $[2, l_{\max}]$, el siguiente modelo ILP, al cual nos referiremos como ILP_{ph1}, puede ser resuelto.

$$\min \sum_{b_i \in B_{\geq l}} x_i + y \tag{3.4}$$

sujeto a:

$$\sum_{b_i \in B_{\geq l}} M1_{i,j} \cdot x_i \leq 1 \quad \text{para } j = 1, \ldots, n \tag{3.5}$$

$$\sum_{b_i \in B_{\geq l}} M2_{i,j} \cdot x_i \leq 1 \quad \text{para } j = 1, \ldots, n \tag{3.6}$$

$$\sum_{b_i \in B_{\geq l}} |t_i| \cdot x_i = n - y \tag{3.7}$$

$$x_i \in \{0, 1\} \quad \text{para } b_i \in B_{\geq l}$$
$$y \in \{0, 1, \ldots, n\}$$

ILP_{ph1} está basado en una variable binaria x_i por cada bloque común $b_i \in B_{\geq l}$, y una variable adicional $y \in \{0, 1, \ldots, n\}$ la cual cuenta el número de

posiciones que no son cubiertas por ningún bloque. Además las matrices $M1$ y $M2$ son las mismas introducidas en la sección 3.1.1, esto es, ellas están definidas sobre todo el conjunto B. La función objetivo minimiza el número de bloques seleccionados y el número de posiciones no cubiertas. Las restricciones (3.5-3.6) son las mismas de ILP_{orig} (ver seccion 3.1.1), más allá del hecho de que las igualdades han sido reemplazadas por símbolos $\leq$ (permitiendo dejar posiciones no cubiertas). Además, en la restricción (3.7) la variable y es agregada al lado derecho, permitiendo con esto el conteo del número de posiciones no cubiertas. En términos simples, la idea de ILP_{ph1} es producir una solución parcial para la instancia MCSP original que cubra lo más posible ambas cadenas de entrada por medio de la menor cantidad posible de bloques comunes.

La resolución de ILP_{ph1} se le denomina como *phase 1* de la heurística propuesta. Sea $\mathcal{S}_{ph1}$ la solución provista por la fase 1. Debido a las restricciones de ILP_{ph1}, esta solución parcial es una solución válida para el problema MCSP original. La idea de la segunda fase es entonces producir la mejor solución completa posible que contenga $\mathcal{S}_{ph1}$. Esto se logra resolviendo el siguiente ILP, al cual se le llamará ILP_{ph2}.

$$\min \sum_{b_i \in B_{ph2} \cup \mathcal{S}_{ph1}} x_i \tag{3.8}$$

sujeto a:

$$\sum_{b_i \in B_{ph2} \cup \mathcal{S}_{ph1}} M1_{i,j} \cdot x_i = 1 \quad \text{para } j = 1, \ldots, n \tag{3.9}$$

$$\sum_{b_i \in B_{ph2} \cup \mathcal{S}_{ph1}} M2_{i,j} \cdot x_i = 1 \quad \text{para } j = 1, \ldots, n \tag{3.10}$$

$$\begin{aligned} x_i &= 1 \quad \text{para } b_i \in \mathcal{S}_{ph1} \\ x_i &\in \{0,1\} \quad \text{para } b_i \in B_{ph2} \end{aligned} \tag{3.11}$$

Donde $B_{ph2} := B(\mathcal{S}_{ph1}) \subset B$ es el conjunto de los bloques comunes que puedes ser agregados a $\mathcal{S}_{ph1}$ sin violar restricciones. Debe notarse que el modelo ILP_{ph2} es el mismo modelo ILP_{orig}, sólo que ILP_{ph2} únicamente considera bloques comunes de B_{ph2} y que éste fuerza a cualquier solución a contener todos los bloques comunes de $\mathcal{S}_{ph1}$; ver restricciones (3.11).

Los resultados de las pruebas realizadas, muestran que la heurística desarrollada logra vencer al mejor algoritmo en la literatura por un promedio de un 6.5% cuando se utiliza el set de prueba propuesto por Ferdous y Sohel (Sección 3.1.2); aunque sólo logra igualar al modelo ILP original (ILP_{orig}) en 15 de los 45 casos propuestos. Por otro lado, al evaluar la heurísticas con instancias de problemas grandes a través del set de prueba producido para esta investigación (ver Sección 3.1.3), ésta logra amplia ventaja sobre (ILP_{orig}), utilizando un fracción del tiempo necesitado por esta última lo que se justifica por el significativamente más pequeño conjunto de componentes de solución

utilizado (bloques comunes), el cual resulta ser de sólo entre un 2.5% a un 6.1% del total de bloques disponibles.

3.2 Desarrollo de una metaheurística híbrida basada en reducción de instancias

Luego de los buenos resultados de la heurística basada en reducción de instancias, se procedió a buscar una extensión de la idea a través del desarrollo de un *framework* que permitiese buscar *genericidad* en la aplicación de esta solución sobre distintos tipos de problemas combinatoriales, manteniendo los beneficios de poder usar un *solver* de propósito general como CPLEX para la solución de instancias de problemas grandes, lo que resulta inviable de forma directa.

De esta forma se proyectó una metaheurística basada en dos componentes fundamentales, primero una técnica heurística constructiva y en segundo lugar un solver exacto para modelos ILP. El algoritmo propuesto se basa en la siguiente idea general:

1. Se produce una subinstancia reducida de un problema, donde la solución a esta subinstancia resulta ser una solución también para la instancia original, no necesariamente la óptima global.
2. Se aplica un solver exacto a la instancia reducida del problema, buscando una solución de alta calidad.
3. Se utiliza el resultado del solver exacto como feedback para una siguiente iteración del algoritmo.

En base a estas ideas se proyectó la metaheurística híbrida llamada *Construct, Merge, Solve & Adapt (CMSA)*, la cual se describe a continuación.

3.2.1 Construct, Merge, Solve & Adapt

El algoritmo Construct, Merge, Solve & Adapt (CMSA) trabaja de la siguiente forma. Durante cada iteración primero se genera un número de soluciones válidas para la instancia de problema original de forma probabilística (Construct). Luego en un segundo paso los componentes de solución utilizados por las soluciones creadas son agregados en conjunto C' (Merge), para luego ser utilizados por un solver exacto y encontrar una solución óptima (Solve). Finalmente para cerrar cada iteración, la información sobre los componentes utilizados y no utilizados de C' brindada por el solver es usada para descartar componentes menos útiles a través de algún mecanismo de envejecimiento (Adapt).

En el Algoritmo 1, se puede ver el detalle del funcionamiento de CMSA y sus parámetros. Cómo se puede apreciar CMSA tiene como parámetros básicos $\text{age}_{\max}$ el cual determina la cantidad máxima de iteraciones que un componente de solución puede estar en C' sin ser utilizado por el solver, además de

Algorithm 1 CONSTRUCT, MERGE, SOLVE & ADAPT (CMSA)

input: instancia de problema $\mathcal{I}$, valores para parámetros n_a y $\text{age}_{\max}$
$S_{\text{bsf}} :=$ NULL, $C' := \emptyset$
$\text{age}[c] := 0$ for all $c \in C$
while tiempo de CPU no alcanzado **do**
  for $i = 1, \ldots, n_a$ **do**
    $S :=$ ProbabilisticSolutionGeneration(C)
    for todo $c \in S$ **and** $c \notin C'$ **do**
      $\text{age}[c] := 0$
      $C' := C' \cup \{c\}$
    end for
  end for
  $S'_{\text{opt}} :=$ ApplyExactSolver(C')
  if S'_{opt} es mejor que S_{bsf} **then** $S_{\text{bsf}} := S'_{\text{opt}}$
  Adapt(C', S'_{opt}, $\text{age}_{\max}$)
end while
output: S_{bsf}

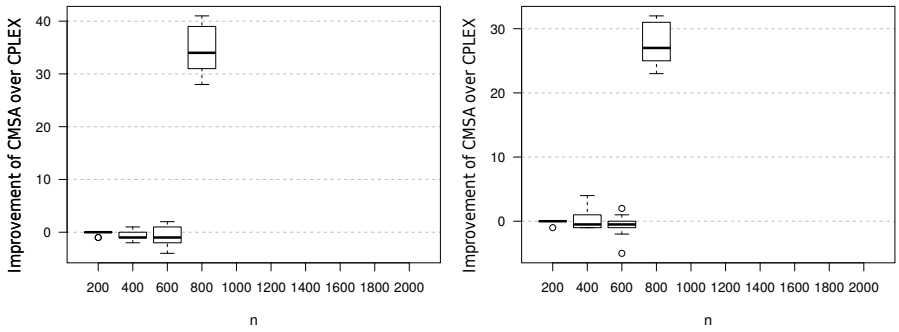
n_a el cual determina el número de soluciones construidas probabilísticamente en el primer paso en cada iteración.

3.2.2 Aplicación de CMSA sobre MCSP & MCA

CMSA fue evaluado sobre los problemas MCSP (ver Sección 2.1) y MCA (ver Sección 2.2). En el caso de MCSP las pruebas se vuelven a realizar sobre el conjunto definido por Ferdous y Sohel (Sección 3.1.2), además de un nuevo set con instancias con $n \in \{200, 400, \ldots, 1800, 2000\}$, que incorpora alfabetos de 12 y 20 caracteres además del tradicional de 4 (ADN), usando dos distribuciones de probabilidad distintas, configurando con esto un conjunto de 600 instancias de prueba. En el caso del problema MCA, también se elaboró un conjunto de prueba el cual está compuesto por DAGs con tamaño en nodos de $n \in \{500, 1000, 5000\}$ y densidades determinadas por probabilidades de existencia de sus arcos, en este caso los nodos pertenecientes al subconjunto X (requisito del problema), son elegidos aleatoriamente del conjunto de nodos alcanzables en proporciones de $\{1\%, 10\%, 20\%\}$ del total de nodos, el conjunto provee un total de 270 problemas.

En el caso del problema MCSP, los resultados de CMSA fueron contrastados contra las siguientes propuestas: la heurística ILP previamente desarrollada [12], una heurística greedy [61], una metaheurística ACO [42, 43], el algoritmo TREESEA [11] y la aplicación del modelo ILP desarrollado sobre la instancia completa de problema(ILP_{compl}). Debe notarse que para los casos de CMSA y TREESEA, se optó por un proceso de calibración de parámetros[1] *ad-*

[1] Proceso llevado a cabo con el uso de la herramienta IRACE (Iterated Race for Automatic Algorithm Configuration) desarrollado en IRIDIA (Université libre de Bruxelles) [76]



(a) Resultados para $\Sigma = 4$, LINEAR (Izquierda), SKEWED (Derecha)

Fig. 3.1: Diferencias entre los resultados de CMSA y los obtenidos por la aplicación de CPLEX a ILP_{orig} relacionado con 600 instancias del segundo set de prueba. Cada caja muestra estas diferencias para 10 instancias. Valores negativos indican que CPLEX obtiene ventaja sobre CMSA.

hoc al problema, llevado a cabo en base a un conjunto de datos diferentes a los utilizados en los experimentos. Por otro lado, al considerar el problema MCA, los contendientes de CMSA fueron la heurística constructiva PGREEDY, que de hecho se corresponde con la fase ProbabilisticSolutionGeneration de CMSA y ILP_{compl}, el cual se refiere a la aplicación del solver CPLEX sobre la instancia completa original del problema. En este caso también se realizó un proceso de calibración de parámetros mediante IRACE [76] que se aplicó a CMSA y PGREEDY

3.2.3 Resultados de la aplicación de CMSA

De la aplicación de CMSA y las otras técnicas evaluadas sobre los problemas estudiados, se pueden extraer las siguientes observaciones relevantes. En el caso de CMSA sobre MCSP, se puede notar que el tamaño del alfabeto incide en los resultados, siendo mayor la complejidad para los algoritmos cuando el alfabeto es más restringido (4 letras). En resumen CMSA se muestra competitivo con ILP_{orig}, en instancias pequeñas, siendo CMSA superior a medida que crece el tamaño de las instancias, venciendo a todos los algoritmos con los cuales se comparó incluído ILP_{orig}, el cual en contraste sobre cierto tamaño de instancia de problema ya no logró obtener resultados dentro de los tiempos admisibles. En la Figura 3.1, se puede apreciar gráficamente las diferencias entre ILP_{orig} y CMSA, en términos de la mejora lograda por CMSA sobre ILP_{orig}. En estas gráficas se puede apreciar con facilidad como se distancian los resultados a medida que crecen los tamaños de instancias para el caso del alfabeto Σ más restringido ($\Sigma = 4$).

Para explicar las diferencias entre los resultados entre CMSA y la aplicación de CPLEX a ILP_{orig} puede inspeccionarse la Figura 3.2, en esta se

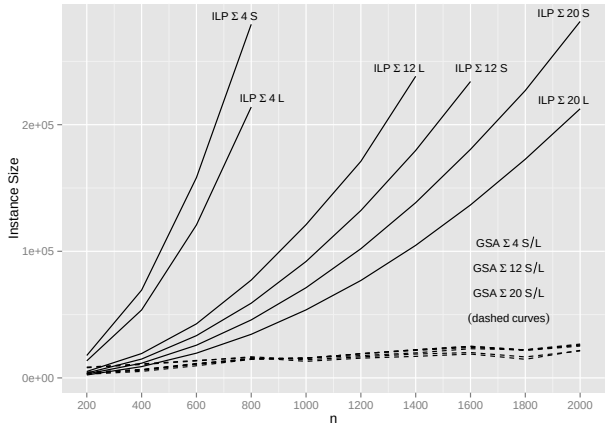


Fig. 3.2: Evolución del tamaño de instancia original usado por CPLEX y de subinstancia usado por CMSA

muestra información sobre el tamaño original de cada instancia de problema en términos del número de componentes de solución disponibles, comparado con el tamaño promedio de la sub-instancias generadas por CMSA.

En la misma figura se demarcan las líneas sólidas que indican el tamaño del conjunto completo de componentes con el formato "ILP Σ X Y", que indica en X el tamaño del alfabeto y con Y el tipo de distribución $Y \in \{L, S\}$, donde L se refiere a una distribución lineal y S a una sesgada, mientras por otro lado las líneas segmentadas muestran el tamaño de la subinstancia usada por CMSA. Considerando lo anterior, se puede apreciar que las seis curvas de líneas sólidas muestran un crecimiento exponencial asociado al incremento de n; mientras que por otro lado, las seis curvas segmendadas exhiben un crecimiento más bien lineal asociado al incremento de n. Además se puede observar que el tamaño de las subinstancias utilizadas por CMSA es significativamente menor que el tamaño de la instancia original. Esta es la razón de porqué el componente CPLEX de la fase (SOLVE) de CMSA puede resolver la subinstancia cerca de los valores óptimos en corto tiempo.

En el caso del problema MCA, los resultados muestran un comportamiento similar al mostrado sobre el problema MCSP, en este caso CMSA se mostró competitivo con ILP_{comp}, en casos de instancias pequeñas y superior en casos grandes donde incluso ILP_{comp} no logró mostrar resultados en las cotas de tiempo establecidas. Al igual que en el caso de MCSP se explica por el bajo porcentaje de uso de componentes de solución en la subinstancia empleada por CMSA, que incluso desciende porcentualmente respecto al tamaño de la instancia original a medida que aumenta el tamaño de los grafos y su densidad, esto se puede revisar en la Figura 3.3.

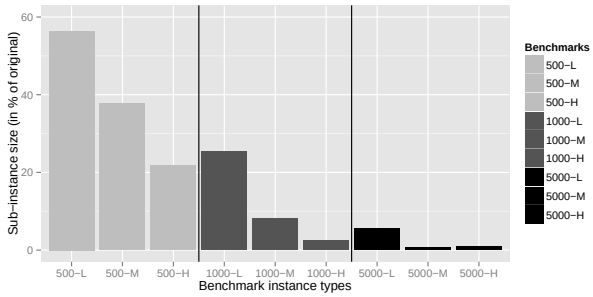


Fig. 3.3: Tamaño promedio de las sub-instancias abordadas por CMSA asociadas a 20 instancias de conjunto de prueba. Las pruebas están categorizadas en nueve subconjuntos diferentes. Los tamaños de las subinstancias están mostrados como porcentajes de los tamaños de las instancias originales.

3.3 Sustitución de CPLEX en CMSA por la metaheurística PBIG

En esta sección se revisará la última parte de la investigación realizada. Esta consiste en el uso del framework CMSA para mejorar el desempeño de otras metaheurísticas. Esto en particular se propone a través del uso de una implementación de un algoritmo greedy iterativo basado en población *(population-based iterated greedy,* (PBIG) para el problema WID (ver Sección 2.3) usada dentro de CMSA como componente de la fase (SOLVE), adicionalmente se muestran contribuciones complementarias realizadas, las cuales consisten en modelos ILP utilizados para enfrentar el problema WID.

La implementación de este nuevo enfoque es necesario, debido que existen casos donde CPLEX no es lo suficientemente eficiente para ser usado como componente SOLVE dentro de CMSA.

3.3.1 Modelos ILP aplicados sobre el problema WID

Se desarrollaron tres modelos ILP para enfrentar el problema WID, los cuales se describen a continuación.

3.3.1.1 ILP-1: Modelo basado en variables indicadoras

El primer modelo ILP —que llamaremos ILP-1—usa tres conjuntos de variables binarias. Por cada nodo $v \in V$ este usa una variable binaria x_v. Además, por cada arista $e \in E$ el modelo usa una variable binaria y_e y una variable binaria z_e. Así x_v indica si v es elegido para la solución. Además, z_e indica si $e \in E$ es seleccionada para conectar un nodo no elegido con uno elegido. La variable y_e es una variable indicadora, que establece si e es seleccionable o no para formar parte de la solución.

$$\text{(Ilp-1)} \quad \min \quad \sum_{v \in V} x_v w(v) + \sum_{e \in E} z_e w(e) \tag{3.12}$$

$$\text{sujeto a:} \quad x_v + x_u \leq 1 \qquad \text{para } e = (u,v) \in E \tag{3.13}$$

$$x_v + \sum_{u \in N(v)} x_u \geq 1 \qquad \text{para } v \in V \tag{3.14}$$

$$x_v + x_u = y_e \qquad \text{para } e = (u,v) \in E \tag{3.15}$$

$$z_e \leq y_e \qquad \text{para } e \in E \tag{3.16}$$

$$x_v + \sum_{e \in \delta(v)} z_e \geq 1 \qquad \text{para } v \in V \tag{3.17}$$

$$x_v \in \{0,1\} \qquad \text{para } v \in V$$

$$y_e \in \{0,1\} \qquad \text{para } e \in E$$

$$z_e \in \{0,1\} \qquad \text{para } e \in E$$

Las restricciones (3.13) son las que establecen la independencia, esto es, estas restricciones se aseguran que dos nodos adyacentes no pueden formar parte de la solución. La restricción (3.14) verifica el conjunto dominante. Esta se asegura de que cada nodo $v \in V$, ya sea el nodo en si mismo o al menos uno en su vecindad esté considerado en la solución. Esos dos conjuntos de restricciones son comunes a los tres modelos presentados. Las restricciones en (3.15), aseguran que las variables indicadoras estén correctamente activadas. Debe notarse que cada arista que contribuye a la función objetivo debe siempre conectar a un nodo no seleccionado para la solución con uno considerado por ésta. Así, si dado una arista $e = (u,v)$, ya sea que v o u está en la solución, la variable y_e es forzada a tomar el valor 1, lo cual indica que esta arista es seleccionable. Las restricciones en (3.16) relacionan las variables indicadoras con las variables que muestran que aristas son seleccionadas. En particular, si una variable indicadora y_e tiene valor 0, z_e es forzada a tomar valor 0, lo cual significa que e no puede ser elegido. Finalmente, las restricciones en (3.17) se aseguran que cada nodo $v \in V$ que no forma parte de la solución —esto es, cuando $x_v = 0$—está conectado por una arista a un nodo que sí forma parte de la solución. Debido al objetivo de minimización en la optimización las aristas con los pesos más bajos serán seleccionadas para este propósito.

3.3.1.2 ILP-2: Eliminando las variables indicadoras

El segundo modelo ILP —el cual llamaremos Ilp-2 —sigue la misma idea de Ilp-1, pero con la salvedad de que no requiere de las variables indicadoras. Esto es, el modelo Ilp-1 sólo necesita las variables binarias x_v para todo $v \in V$ y las variables z_e para todo $e \in E$. El significado de estas variables ya fue descrito anteriormente.

$$\text{(ILP-2)} \quad \min \quad \sum_{v \in V} x_v w(v) + \sum_{e \in E} z_e w(e) \tag{3.18}$$

$$\text{sujeto a:} \quad x_v + x_u \leq 1 \qquad \text{para } e = (u, v) \in E \tag{3.19}$$

$$x_v + \sum_{u \in N(v)} x_u \geq 1 \qquad \text{para } v \in V \tag{3.20}$$

$$x_v + x_u \geq z_e \qquad \text{para } e = (u, v) \in E \tag{3.21}$$

$$(1 - x_v) + (1 - x_u) \geq z_e \qquad \text{para } e = (u, v) \in E \tag{3.22}$$

$$x_v + \sum_{e \in \delta(v)} z_e \geq 1 \qquad \text{para } v \in V \tag{3.23}$$

$$x_v \in \{0, 1\} \qquad \text{para } v \in V$$

$$z_e \in \{0, 1\} \qquad \text{para } e \in E$$

Debe notarse que las restricciones del conjunto independiente (3.19), las del conjunto dominante (3.20), y las restricciones que se aseguran de que cada nodo $v \in V$ que no forma parte de la solución está conectado por una arista a un nodo que forma parte de la solución, son las mismas que las de modelo ILP-1. Sin embargo, los dos grupos de restricciones de ILP-1 (restricciones (3.15) y (3.16)), son reemplazados por los grupos (3.21) y (3.22). Debe notarse que cuando ambos x_v y x_u —relacionados mediante la arista $e = (u, v) \in E$— están con valor cero, la restricción (3.21) fuerza a la variable z_e a tomar valor cero, lo que significa que una arista que conecta los dos nodos no seleccionados puede ser elegida por la solución. Además cuando ambos x_v y x_u —nuevamente relacionados mediante la arista $e = (u, v) \in E$— tienen valor 1, la restricción (3.27) fuerza a la variable z_e a tomar el valor cero, lo cual significa que la arista que conecta los dos nodos seleccionados no puede ser elegida para la solución.

3.3.1.3 ILP-3: Uso de variables explícitas para considerar peso de arcos

El tercer modelo ILP—el cual llamaremos de ahora en adelante ILP-3— es estructuralmente diferente a ILP-1 y ILP-2. La idea central de éste es modelar la contribución del peso de las aristas de cada nodo en términos de una variable entera q_v para todo $v \in V$. Obviamente, el peso de las aristas de un nodo seleccionado $v \in V$—esto es, cuando $x_v = 1$—debe ser cero, mientras la contribución de la arista de un nodo no seleccionado $v \in V$ debe ser igual al peso de la arista de peso mínimo que conecta este nodo a un nodo seleccionado.

$$\text{(ILP-3)}\quad \min \sum_{v\in V} x_v w(v) + q_v \tag{3.24}$$

$$\text{sujeto a:}\quad x_v + x_u \leq 1 \qquad \forall\, e=(u,v)\in E \tag{3.25}$$

$$x_v + \sum_{u\in N(v)} x_u \geq 1 \qquad \forall\, v\in V \tag{3.26}$$

$$q_v \leq (1-x_v)M \qquad \forall\, v\in V \tag{3.27}$$

$$q_v \geq 0 \qquad \forall\, v\in V \tag{3.28}$$

$$q_v \geq x_u w(e) - \left(x_v M + \sum_{\substack{e'=(v,v')\in\delta(v)\\ \text{s.t. } w(e')<w(e)}} x_{v'} M \right) \qquad \begin{array}{l}\forall\, v\in V,\\ e=(v,u)\in\delta(v)\end{array} \tag{3.29}$$

$$x_v \in \{0,1\} \qquad \forall\, v\in V$$

$$q_v \in \{-|V|\cdot M,\ldots,M\} \qquad \forall\, v\in V$$

Debe observarse que las restricciones de conjunto independiente (3.25) y las de conjunto dominante (3.26) están también en ILP-1 y ILP-2. Además, las restricciones (3.27) establecen el límite superior de la contribución de una artista en cero en casos que el nodo correspondiente forma parte de la solución, esto es, del conjunto dominante e independiente. En caso de que el nodo no forme parte de la solución, la restricción (3.27) establece el límite superior en un valor de una gran constante M, la cual fue establecida en función del peso máximo de todas las aristas del grafo. Además, el conjunto de restricciones en (3.28) establece el límite inferior de todas las contribuciones en cero. Finalmente, las restricciones (3.29) establecen el límite inferior de la contribución de aristas, para que éstas sean iguales al peso de la arista de mínimo peso que conecta el respectivo nodo con uno de su vecindad seleccionada.

3.3.2 Aplicación de Metaheurística PBIG sobre el problema WID

Los tres modelos ILP presentados, fueron desarrollados parar ser evaluados como componentes de la fase SOLVE de CMSA. Lamentablemente el desempeño de estos tres modelos fue pobre, incluso con instancias más pequeñas que las originales en casos con 500 y 1000 nodos. Esto llevó a desarrollar un nuevo planteamiento, en el cual el *solver* es reemplazado por otra metaheurística que es aplicada sobre la sub-instancia reducida generada por las primeras fases del CMSA, pero con un tiempo más limitado. La metaheurística seleccionada para este propósito es PBIG, la cual se describe a continuación, aplicada a WID.

Una descripción de alto nivel de la implementación de PBIG—al que se referirá como PBIG—está dada en el algoritmo 2. PBIG requiere el grafo problema G, y los parámetros: (1) tamaño de población $p_{\text{size}} \in \mathbb{Z}^+$, (2) el límite inferior (D^l) y el límite superior (D^u), para el nivel de destrucción aplicado

Algorithm 2 PBIG para el problema WID

1: **input:** grafo de entrada G, parámetros $p_{\text{size}} > 0$, $D^l, D^u, d_{\text{rate}}, l_{\text{size}} \in [0,1]$
2: $\mathcal{P} :=$ GenerateInitialPopulation$(p_{\text{size}}, d_{\text{rate}}, l_{\text{size}})$
3: **while** no se satisfaga la condición de término **do**
4: $\quad\mathcal{P}_{\text{new}} := \emptyset$
5: $\quad$**for** cada solución candidata $S \in \mathcal{P}$ **do**
6: $\quad\quad\hat{S} :=$ DestroyPartially(S)
7: $\quad\quad S' :=$ Reconstruct$(\hat{S}, d_{\text{rate}}, l_{\text{size}})$
8: $\quad\quad$AdaptDestructionRate(S, S')
9: $\quad\quad\mathcal{P}_{\text{new}} := \mathcal{P}_{\text{new}} \cup \{S'\}$
10: $\quad$**end for**
11: $\quad\mathcal{P} :=$ Accept$(\mathcal{P}, \mathcal{P}_{\text{new}})$
12: **end while**
13: **output:** argmin $\{f(S) \mid S \in \mathcal{P}\}$

a cada solución de la población por cada iteración, (3) el grado de determinismo $d_{\text{rate}} \in [0,1]$, y (4) el tamaño de la lista de candidatos $l_{\text{size}} > 0$. Los dos últimos parámetros controlan el nivel de avaricia *(greediness)* del proceso probabilístico de reconstrucción de soluciones. Además, se debe notar que los valores para los límites mencionados deben cumplir $0 \leq D^l \leq D^u \leq 1$. Para la siguiente descripción, cada solución S es un subconjunto de los nodos de V, tiene un valor de función objetivo $f(S)$, y una tasa de destrucción D_S individual posiblemente dinámica.

El algoritmo funciona como sigue. Primero las p_{size} soluciones de la población inicial son generadas por la función GenerateInitialPopulation (p_{size}, d_{rate}, l_{size}) (ver línea 2 de Alg. 2). Luego, cada iteración consiste de los siguientes pasos: Primero, una población vacía $\mathcal{P}_{\text{new}}$, llamada población de descendientes es creada. Luego, cada solución $S \in \mathcal{P}$ es parcialmente destruida usando el procedimiento DestroyPartially(S) (ver línea 6 de Alg. 2). Esto resulta en una solución parcial $\hat{S}$. Sobre la base de $\hat{S}$, se construye una solución completa S' usando el procedimiento Reconstruct$(\hat{S}, d_{\text{rate}}, l_{\text{size}})$ (ver línea 7 de Alg. 2). Aquí, la tasa de destrucción D_S de la solución S es adaptada dependiente de la calidad de la solución S' en la función AdaptDestructionRate(S, S'). Cada solución nueva obtenida es almacenada en $\mathcal{P}_{\text{new}}$. Nótese que las dos fases de destrucción y reconstrucción son aplicadas a todas las soluciones de $\mathcal{P}$ de forma independiente entre ellas. Cuando la iteración está completa, el procedimiento Accept$(\mathcal{P}, \mathcal{P}_{\text{new}})$ selecciona las mejores p_{size} soluciones de $\mathcal{P} \cup \mathcal{P}_{\text{new}}$ para la población de la próxima generación. En el caso de que dos soluciones de $\mathcal{P} \cup \mathcal{P}_{\text{new}}$ sean iguales, el criterio de desempate está basado en las tasas de destrucción individuales. Más específicamente, la solución S con la tasa individual más alta de destrucción D_S es preferida sobre la otra. Finalmente el algoritmo termina cuando el límite de tiempo de procesamiento ha sido alcanzado, y la mejor solución es retornada.

3.3.3 Uso de PBIG como componente de CMSA

El framework CMSA fue desarrollado para tratar de dar escalabilidad a la eficiencia de solvers ILP exactos en el contexto de instancias de problemas grandes, donde de forma directa resultan completamente inviables. Debido a los deficientes resultados de los modelos ILP mostrados sobre el problema WID, esta versión de CMSA no usa un solver exacto sino que en su lugar trata las sub-instancias con el algoritmo Pbig, el cual se aplica sobre cada sub-instancia con cierta limitación de tiempo. El algoritmo resultante se denomina (Cmsa-Pbig), el cual es provisto como Algoritmo 3.

En el contexto del problema WID, el conjunto de componentes de solución utilizados por esta versión de Cmsa corresponde al conjunto de nodos del grafo de entrada. Además, las soluciones pueden ser probabilísticamente construidas por dos versiones de greedy estocásticas definidas para este propósito. Estos algoritmos requieren dos parámetros: (1) la tasa de determinismo que llamaremos (d^{cmsa}_{rate} en el contexto de Cmsa-Pbig) y (2) el tamaño de la lista de candidatos (l^{cmsa}_{size} en el contexto de Cmsa-Pbig).

El algoritmo inicia su operación de la siguiente forma, primero, la mejor solución hasta el momento S_{bsf} es inicializada a NULL, indicando que aún no existe una solución, Además la subinstancia actual $V' \subseteq V$ (donde V es el conjunto de nodos del grafo de entrada G) es inicializado como un conjunto vacío. Entonces en cada iteración, un número de n_{a} soluciones es probabilísticamente generado a través de la función ProbabilisticSolutionGeneration($\text{opt}_{\text{greedy}}$, d^{cmsa}_{rate}, l^{cmsa}_{size}), en este caso el parámetro $\text{opt}_{\text{greedy}}$ permite seleccionar una de las dos implementaciones elaboradas de Greedy para estos propósitos. Los nodos encontrados en las soluciones construidas son agregados a V'. Además cada nodo $v \in V'$ tiene una *edad*, etiquetada como $age[v]$, la cual se inicializa en cero. Luego, Pbig es aplicado en la función ApplyPBIG(V') para encontrar soluciones de alta calidad, pero restringido a los nodos de V', esto es, el proceso de re-construcción de Pbig está restringido a elegir nodos de V'. Si la solución resultante, llamada S'_{pbig}, es mejor que la mejor solución hasta el momento S_{bsf}, la solución S'_{pbig} es adoptada como la mejor solución hasta el momento. Luego, la subinstancia V' es adaptada en base a la solución S_{bsf} en conjunto con las edades de los nodos en V'. Esto es realizado por la función Adapt(V', S'_{pbig}, age_{max}) de la siguiente forma. Primero, la edad de cada nodo en $V' \setminus S'_{\text{pbig}}$ es incrementada mientras la edad de cada nodo en S'_{pbig} es reinicializada a cero. Subsecuentemente, aquellos nodos de V' con una edad mayor a age_{max} —el cual es un parámetro del algoritmo—es removido de V'. Esto causa que nodos que no son seleccionados por las mejores soluciones de Pbig no hagan más lenta la ejecución de Pbig en las siguientes iteraciones.

Resultados

Se realizó una extensa comparación de algoritmos sobre el problema WID, lo que incluyó a los tres modelos ILP presentados (Ilp-1 y Ilp-2,Ilp-3), las

Algorithm 3 CMSA-PBIG para el problema WID

input: grafo de entrada G, valores de parámetros para PBIG y valores para los parámetros d_{rate}^{cmsa}, l_{size}^{cmsa}, $\text{age}_{\max}$, n_{a}, $t_{\max}$, and $\text{opt}_{\text{greedy}}$
$S_{\text{bsf}} :=$ NULL
$V' := \emptyset$
$age[v] := 0$ para todo $v \in V$
while el tiempo límite de CPU no sea alcanzado **do**
  for $i = 1, \ldots, n_{\text{a}}$ **do**
    $S :=$ ProbabilisticSolutionGeneration($\text{opt}_{\text{greedy}}$, d_{rate}^{cmsa}, l_{size}^{cmsa})
    for todo $v \in S$ **and** $v \notin V'$ **do**
      $age[v] := 0$
      $V' \leftarrow V' \cup \{v\}$
    end for
  end for
  $S'_{\text{pbig}} \leftarrow$ ApplyPBIG(V')
  if $f(S'_{\text{pbig}}) < f(S_{\text{bsf}})$ **then** $S_{\text{bsf}} := S'_{\text{pbig}}$
  Adapt(V', S'_{pbig}, $\text{age}_{\max}$)
end while
output: S_{bsf}

dos versiones de heurísticas greedy desarrolladas, el algoritmo PBIG y CMSA-PBIG, esto es, siete algoritmos en total. Las pruebas se realizaron sobre un total de 540 instancias de problemas provenientes de dos conjuntos de prueba de similar tamaño. El primer conjunto se denomina *conjunto de grafos aleatorios*, y sus instancias se caracterizan porque todos los nodos tienen similar probabilidad de conectarse entre sí. El conjunto posee una variedad de grafos determinados por la probabilidad de la ocurrencia de una arista, su cantidad de nodos y distintos tipos de esquemas de generación de pesos, los cuales pueden dar más importancia a los nodos o a las aristas, o bien ser neutrales. Las mismas consideraciones de esquemas de pesos, cantidad de nodos y probabilidad de nodos se aplica para el segundo conjunto de prueba llamado *conjunto de grafos geométricos aleatorios*. La diferencia radica en que este segundo grupo introduce el concepto de radio de conexión el cual limita la posibilidad de conectar nodos entre si, a los que están en dentro de esta distancia determinada por el radio, esto se logra con una disposición aleatoria inicial de coordenadas para todos los nodos de la instancia.

Es importante notar que PBIG y CMSA-PBIG fueron sometidos a procesos de calibración usando la herramienta IRACE.

Luego de las pruebas realizadas, al considerar todas las instancias en conjunto, se puede constatar que CMSA-PBIG es el algoritmo con mejor desempeño, seguido por PBIG. Luego los siguen los modelos ILP, donde ILP-2 es generalmente el de mejor rendimiento seguido por ILP-3 y ILP-1. Finalmente los de peor rendimiento son las heurísticas greedy. Todas las diferencias tienen significancia estadística. Una representación de esto se puede ver en la Figura 3.4, a través de un gráfico de diferencias críticas (CD), donde cada algoritmo

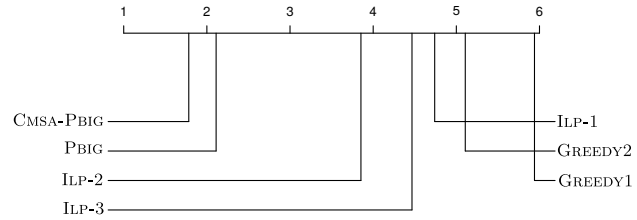


Fig. 3.4: Gráfico de diferencias críticas para las 540 instancias de problemas en conjunto. El eje muestra el ránking promedio para los 7 algoritmos considerados.

es posicionado en un segmento de acuerdo al ranking promedio respecto al total de instancias de problema. La diferencia crítica es computada con nivel de significancia del 0.05, y para este caso entre todas las propuestas existen diferencias estadísticas significativas. Adicionalmente se pudo notar que las estructuras de los grafos no alteran el desempeño, siendo éste similar para los distintos tipos de grafos. Cmsa-Pbig y Pbig son las mejores propuestas sobre el problema WID entre las técnicas verificadas, con significancia estadística en todos los casos. Ahora, en la comparación entre Cmsa-Pbig y Pbig, se puede observar que Cmsa-Pbig tiene —para todos los subconjuntos de instancias— un mejor desempeño promedio que Pbig. Esta diferencia es estadísticamente significativa cuando consideramos todas las instancias de problema en conjunto, y en el caso de grafos orientados a aristas. Este resultado es de gran interés, debido a que muestra que, con la aplicación del framework Cmsa sobre otra metaheurística, es posible incrementar el rendimiento de la metaheurística.

4

Conclusiones

En este trabajo se ha mostrado cómo instancias grandes de problemas de optimización combinatorial pueden ser abordadas por técnicas exactas (ILP) sobre herramientas *solver* estándares, lo cual se logra, cuando previamente se realiza una reducción inteligente del tamaño de la instancia de problema; sin lo cual la aplicación de ILP sería inviable. Esto se descubrió en un principio, gracias al desarrollo de una heurística exacta basada en MIP, que entregó el sustento para el desarrollo de un framework general llamado Cmsa, que resulta ser la principal contribución de este trabajo.

Cmsa mostró su utilidad en la simplificación y depuración de subinstancias de problemas grandes permitiendo el uso de modelos ILP sobre ellas, además esta técnica mostró su valor al mejorar el rendimiento de otras metaheurísticas, lo que es un resultado de gran interés, y que abre un campo de investigación futuro. De forma detallada, las principales conclusiones de esta investigación son las siguientes:

- Durante el proceso de investigación se desarrolló un conjunto de técnicas algorítmicas que configuran las contribuciones de esta tesis. Estas son un modelo ILP para el problema MCSP que mejora lo descrito en la literatura para problemas pequeños, y una heurística basada en ILP, que también mejora los resultados publicados, pero con la capacidad de enfrentar instancias de problemas grandes. Además se presenta una nuevo framework de optimización de propósito general Cmsa, que permite ser la base de desarrollo de metaheurísticas híbridas y que prueba su utilidad al ser usado complementariamente con modelos ILP y otra metaheurística (Pbig). En este contexto se logra posicionar a Cmsa en el *estado-del-arte* para los problemas MCSP, MCA y WIDP, en especial al tratar con instancias de problemas grandes. Durante el proceso de desarrollo de Cmsa, también se logró el desarrollo de tres modelos ILP para el problema WIDP y dos variantes Greedy para el mismo problema, junto con la implementación de la metaheurística Pbig.

- Cmsa permite extender el uso de modelos ILP sobre instancias de problemas que de forma directa les sería inviable usando herramientas *solver* estándar–en este caso particular CPLEX–. En términos generales, la resolución de instancias reducidas por el framework propuesto, muestra resultados competitivos con el modelo ILP aplicado sobre las instancias originales no reducidas de problemas pequeños (cuando el modelo ILP puede aplicarse de forma directa a la instancia de problema) y supera ampliamente al uso de modelos ILP de forma directa en problemas medianos y grandes.
- Otro resultado interesante de la investigación, es el hecho de que Cmsa permitió mejorar el buen desempeño de la metaheurística Pbig, que por si sola estaba posicionada en el *estado-del-arte* en la solución del problema WIDP. Esto entrega un precedente importante y abre líneas de investigación asociadas al desarrollo de otras hibridaciones.

4.1 Trabajos futuros

Derivado del trabajo realizado, se vislumbran dos líneas complementarias de trabajo, una ligada a la mejora del framework presentado, y la segunda vinculada a la explotación de éste. En el contexto de la mejora, es claro que debe revisarse con más detalle el mecanismo de envejecimiento de componentes de solución en la fase Adapt de Cmsa, la cual según experimentos preliminares muestra que podría ser mejorada considerando aspectos sobre la calidad de las soluciones encontradas y la interacción de estos componentes, en conjunto al uso un enfoque estocástico en este proceso. En el contexto de la explotación de Cmsa, se debe explorar la utilidad del framework sobre otro tipo de problemas de optimización combinatorial, como es el caso de problemas de permutación y planificación, así también la prueba en conjunto con otras metaheurísticas disponibles en el *estado-del-arte* de distintos CO, buscando mejorar los resultados de la literatura.

5

Publicaciones

Como resultado de la investigación, se desarrollaron diversos artículos en revistas y conferencias. En esta sección se introducen los artículos seleccionados para esta disertación y presentados con posterioridad en la Parte II del trabajo. Algunos trabajos publicados no fueron elegidos por constituir resultados parciales posteriormente extendidos en otras publicaciones o por no estar relacionados con la temática general presentada en este trabajo.

5.1 Publicaciones incluidas en la disertación

Capítulo 6: Mathematical Programming Strategies for Solving the Minimum Common String Partition Problem

Este trabajo muestra los resultados de enfrentar el MCSP con un modelo de programación lineal. En este contexto se muestra el primer modelo ILP desarrollado para este problema y su desempeño, poniendo énfasis en que no es aplicable a problemas grandes. El trabajo explora el desarrollo de una heurística de dos fases basada en ILP y sustentada en la reducción de instancias de problema que resulta tener resultados superiores a los existentes en la literatura.

Capítulo 7: Construct, Merge, Solve Adapt: A New General Algorithm For Combinational Optimization

Dada la ventaja mostrada por la reducción de tamaño de instancia encontrada, este trabajo presenta una nueva metaheurística híbrida de propósito general basada en este concepto. La propuesta es probada sobre los problemas MCSP y MCA mostrando ser competitiva con solvers exactos para instancias de problemas pequeños y medianos y muy superior a estos con instancias de problemas grandes.

Capítulo 8: The Weighted Independent Domination Problem: Integer Linear Programming Models and Metaheuristic Approaches

En este trabajo originalmente se busca extender la aplicación de CMSA sobre el problema WID, y en ese contexto el artículo informa del desarrollo de dos heurísticas del tipo *greedy* y tres modelos ILP (componentes de CMSA). Dado que el *solver* utilizado (CPLEX) no resultan lo suficientemente eficiente para su integración con CMSA, se presenta una metaheurística PBIG aplicada a WID y una aplicación de CMSA con PBIG como alternativa a los modelos ILP, mostrando resultados sobresalientes de estos dos últimos algoritmos.

5.2 Publicaciones no incluidas en la disertación

The Weighted Independent Domination Problem: ILP Model and Algorithmic Approaches

Este trabajo se presentó en (EvoCop2017) y fue nominado a mejor artículo. Este presenta el desarrollo de un modelo ILP y dos heurísticas voraces *(greedy)* para abordar el problema WID. Además para este mismo problema, se presenta una metaheurística PBIG. Los resultados muestran el buen desempeño del modelo ILP en instancias de problema pequeño y la superioridad de la metaheurística en instancias de problemas mayores.

Iterative Probabilistic Tree Search for the Minimum Common String Partition Problem

En este artículo se explora el uso de una técnica de búsqueda en árbol iterativa probabilística sobre el problema MCSP, mostrando como ésta supera a las implementaciones estándar de greedy, y además de una implementación de una metaheurística ACO.

Generate, Solve & Adapt: Una propuesta de metaheurística híbrida aplicada al problema de kCT

Este artículo presenta una propuesta del framework GSA, el cual es una versión temprana de CMSA. La propuesta se evalúa sobre el problema kCT mostrando ventajas sobre heurísticas voraces tradicionales y permitiendo el uso de modelos ILP sobre problemas que no podrían ser abordados con este de forma directa.

An Artificial Bioindicator System for Network Intrusion Detection

Este trabajo responde a una línea de investigación distinta a la presentada en este texto, pero realizado durante el período de estudios doctorales. En el artículo se presenta una propuesta de bioindicadores artificiales usados como clasificadores para detectar intrusiones en redes de datos. El clasificador está inspirado en el sistema inmune biológico e implementa una población de agentes que evoluciona y aprende a vivir en su entorno. En este proceso se transforman en bioindicadores que pueden reaccionar a anomalías. La propuesta permite detectar ataques desconocidos y no requiere entrenamiento previo. En las pruebas realizadas pudo mejorar los resultados de 3 algoritmos en el estado del arte usando un conjunto de prueba estandarizado.

5.3 Lista de publicaciones

En esta sección se proveen las referencias el trabajo científico publicado durante el desarrollo de esta disertación. Los artículos incluidos en esta disertación están destacados.

Revistas internacionales

- **Pedro Pinacho Davidson, Christian Blum and José A. Lozano. The Weighted Domination Problem: Integer Linear Programming Models and Metaheuristic Approaches. *European Journal of Operational Research (EJOR).* (enviado)**

- **Christian Blum, Pedro Pinacho, Manuel López.Ibáñez, José A. Lozano. Construct, Merge, Solve & Adapt: A New General Algorithm For Combinational Optimization. *Computers & Operations Research* 68 (2016), pp. 75-88.**

- **Christian Blum, José A. Lozano, Pedro Pinacho Davidson. Mathematical Programming Strategies for Solving the Minimum Common String Partition Problem. *European Journal of Operational Research (EJOR)* 242 (3) (2015), pp. 769-777.**

- Christian Blum, José A. Lozano, Pedro Pinacho Davidson. An Artificial Bioindicator System for Network Intrusion Detection . *Artificial Life* 21 (2) (2015), pp. 93-118.

Conferencias internacionales

- Pinacho, Pedro, Blum C., Lozano J.A. (2017) The Weighted Independent Domination Problem: ILP Model and Algorithmic Approaches. In: Hu B.,

López-Ibáñez M. (eds) Evolutionary Computation in Combinatorial Optimization. EvoCOP 2017. *Lecture Notes in Computer Science*, vol 10197. Springer, Cham.

- Blum C., Lozano J.A., Pinacho Davidson P. (2014) Iterative Probabilistic Tree Search for the Minimum Common String Partition Problem. In: Blesa M.J., Blum C., Voß S. (eds) Hybrid Metaheuristics. HM 2014. *Lecture Notes in Computer Science*, vol 8457. Springer, Cham.

Conferencias nacionales

- Pedro Pinacho, Christian Blum, José A. Lozano. Generate, Solve & Adapt: Una propuesta de metaheurística híbrida aplicada al problema de KCT. *XI Congreso Chileno de Investigación Operativa (OPTIMA 2015)*, Antofagasta, Chile.

References

[1] Aggarwal, C., Orlin, J., and Tai, R. (1997). Optimized crossover for the independent set problem. *Operations Research*, 45:226–234.

[2] Ahuja, R., Orlin, J., and Tiwari, A. (2000). A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27:917–934.

[3] Applegate, D., Bixby, R., Chvátal, V., and Cook, W. (1999). Finding tours in the TSP. Technical report, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany.

[4] Bertsimas, D. and Tsitsiklis, J. N. (1997). *Introduction to Linear Optimization*. Athena Scientific.

[5] Birrattari, M., Paquete, L., Stützle, T., and Varrentrapp, K. (2001). *Classification of metaheuristics and design of experiments for the analysis of components*, volume AIDA-0. Techn. Univ., FB 20, FG Intellektik, Darmstadt.

[6] Blesa Aguilera, M. J., Blum, C., Cotta, C., Fernández, A. J., Gallardo, J. E., Roli, A., and Sampels, M., editors (2008). *Proceedings of HM 2008 – Fifth International Workshop on Hybrid Metaheuristics*, volume 5296 of *Lecture Notes in Computer Science*. Springer.

[7] Blesa Aguilera, M. J., Blum, C., Di Gaspero, L., Roli, A., Sampels, M., and Schaerf, A., editors (2009). *Proceedings of HM 2009 – Sixth International Workshop on Hybrid Metaheuristics*, volume 5818 of *Lecture Notes in Computer Science*. Springer.

[8] Blum, C. (2005). Beam-ACO–hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers and Operations Research*, 32:1565–1591.

[9] Blum, C. and Blesa, M. J. (2005). New metaheuristic approaches for the edge-weighted k-cardinality tree problem. *Computers & Operations Research*, 32(6):1355–1377.

[10] Blum, C. and Blesa, M. J. (2009). Solving the KCT problem: Large-scale neighborhood search and solution merging. In Alba, E., Blum, C., Isasi, P.,

León, C., and Gómez, J. A., editors, *Optimization Techniques for Solving Complex Problems*, pages 407–421. Wiley & Sons.

[11] Blum, C., Lozano, J. A., and Pinacho Davidson, P. (2014). *Iterative Probabilistic Tree Search for the Minimum Common String Partition Problem*, pages 145–154. Springer International Publishing, Cham.

[12] Blum, C., Lozano, J. A., and Pinacho Davidson, P. (2015). Mathematical programming strategies for solving the minimum common string partition problem. *European Journal of Operational Research*, 242(3):769–777.

[13] Blum, C., Puchinger, J., Raidl, G. R., and Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11:4135–4151.

[14] Blum, C. and Raidl, G. R. (2016). *Hybrid Metaheuristics - Powerful Tools for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer.

[15] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308.

[16] Boettcher, S. and Percus, A. G. (2001). Extremal optimization for graph partitioning. *Physical Review E*, 64(2):026114.

[17] Borndörfer, R., Ferreira, C. E., and Martin, A. (1997). Matrix decomposition by branch-and-cut. Technical report.

[18] Borndörfer, R., Ferreira, C. E., and Martin, A. (1998). Decomposing matrices into blocks. *Society for Industrial and Applied Mathematics, SIAM Journal on Optimization*, 9(1):236–269.

[19] Boussaïd, I., Lepagnot, J., and Siarry, P. (2013). A survey on optimization metaheuristics. *Inf. Sci.*, 237:82–117.

[20] Bui, T. N. and Sundarraj, G. (2004). *Ant System for the k-Cardinality Tree Problem*, pages 36–47. Springer Berlin Heidelberg, Berlin, Heidelberg.

[21] Burke, E. and Kendall, G., editors (2005). *Search Methodologies: Introductory Tutorials in Optimisation and Decision Support Techniques*. Springer.

[22] Cahon, S., Melab, N., and Talbi, E.-G. (2004). ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380.

[23] Chang, S., Liu, J. J., and Wang, Y. (2014). The weighted independent domination problem in series-parallel graphs. In *Intelligent Systems and Applications - Proceedings of the International Computer Symposium (ICS) held at Taichung, Taiwan, December 12-14, 2014*, pages 77–84.

[24] Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., and Jiang, T. (2005). Computing the assignment of orthologous genes via genome rearrangement. In *Proceedings of the Asia Pacific Bioinformatics Conference 2005*, pages 363–378.

[25] Cher, Ming, T., editor (2008). *Simulated Annealing*. InTech, Shanghai, China.

[26] Cheung, S. Y. and Kumar, A. (1994). Efficient quorumcast routing algorithms. In *INFOCOM '94. Networking for Global Communications., 13th Proceedings IEEE*, pages 840–847 vol.2.

[27] Chiarandini, M., Dumitrescu, I., and Stützle, T. (2008). Very large-scale neighborhood search: Overview and case studies on coloring problems. In Blum, C., Blesa Aguilera, M. J., Roli, A., and Sampels, M., editors, *Hybrid Metaheuristics – An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*, pages 117–150. Springer.

[28] Clements, D., Crawford, J., Joslin, D., Nemhauser, G., Puttlitz, M., and Savelsbergh, M. (1997). Heuristic optimization: A hybrid AI/OR approach. In Davenport, A. and Beck, C., editors, *Proceedings of the Workshop on Industrial Constraint-Directed Scheduling.* held in conjunction with the Third International Conference on Principles and Practice of Constraint Programming (CP97).

[29] Clerc, M., editor (2006). *Particle Swarm Optimization.* Wiley-ISTE Publishers, Newport Beach, CA.

[30] Collins, N. E. (1988). Simulated annealing - an annotated bibliography. *Am. J. Math. Manage. Sci.*, 8(3-4):209–307.

[31] Cook, W. and Seymour, P. (2003). Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248.

[32] Coudert, D., Nepomuceno, N., and Rivano, H. (2010). Power-efficient radio configuration in fixed broadband wireless networks. *Computer Communications*, 33(8):898–906.

[33] Coudert, D., Nepomuceno, N. V., and Tahiri, I. (2011). Energy saving in fixed wireless broadband networks. In Pahl, J., Reiners, T., and Voß, S., editors, *Proceedings of INOC 2011 – 5th International Conference on Network Optimization*, volume 6701 of *Lecture Notes in Computer Science*, pages 484–489. Springer.

[34] Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms.* PhD thesis, Politecnico di Milano, Italy.

[35] Dorigo, M. and Gambardella, L. M. (1997). Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66.

[36] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1):29–41.

[37] Dorigo, M., Maniezzo, V., and Colorni, A. (1999). Positive Feedback as a Search Strategy. *Technical Report No. 91-016, Politecnico di Milano, Italy, 1991.*

[38] Dörner, K. et al., editors (2010). *Proceedings of Matheuristics 2010: Third International Workshop on Model Based Metaheuristics*, Vienna, Austria.

[39] Dueck, G. (1993). New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(1):86–92.

[40] Ehrgott, M. and Freitag, J. (1996). K_tree/k_subgraph: A program package for minimal weighted k-cardinality trees and subgraphs. *European Journal of Operational Research*, 93(1):224 – 225.

[41] Ehrgott, M., Freitag, J., Hamacher, H. W., and Maffioli, F. (1997). Heuristics for the k-cardinality tree and subgraph problem. *Asia-Pacific Journal of Operational Research*, 14(1):87–114.

[42] Ferdous, S. M. and Sohel Rahman, M. (2013). Solving the minimum common string partition problem with the help of ants. In Tan, Y., Shi, Y., and Mo, H., editors, *Proceedings of ICSI 2013 – 4th International Conference on Advances in Swarm Intelligence*, volume 7928 of *Lecture Notes in Computer Science*, pages 306–313. Springer.

[43] Ferdous, S. M. and Sohel Rahman, M. (2014). A MAX-MIN ant colony system for minimum common string partition problem. *CoRR*, abs/1401.4539. `http://arxiv.org/abs/1401.4539`.

[44] Festa, P. and Resende, M. G. C. (2009a). An annotated bibliography of grasp - part 1: Algorithms. *International Transactions in Operational Research*, 16(1):1–24.

[45] Festa, P. and Resende, M. G. C. (2009b). An annotated bibliography of grasp part 2: Applications. *International Transactions in Operational Research*, 16(2):131–172.

[46] Fischetti, M., Hamacher, H. W., Jørnsten, K., and Maffioli, F. (1994). Weighted k-cardinality trees: Complexity and polyhedral structure. *Networks*, 24(1):11–21.

[47] Fleischer, M. (1995). Simulated annealing: Past, present, and future. In *Proceedings of the 27th Conference on Winter Simulation*, WSC '95, pages 155–161, Washington, DC, USA. IEEE Computer Society.

[48] Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. Wiley.

[49] Foulds, L., Wilson, J., and of Waikato. Department of Management Systems, U. (1995). *Integer Programming Approaches to Facilities Layout Models with Forbidden Areas*. Research report series. Department of Management Systems, University of Waikato.

[50] Foulds, L. R. and Hamacher, H. W. (1992). A new integer programming approach to (restricted) facilities layout problems allowing flexible facility shapes. Technical report, Department of Management Science, University of Waikato, New Zealand.

[51] Freitag, J. (1993). Minimal k-cardinality trees Master's Thesis (In german), Department of Mathematics, University of Kaiserslautern, Germany.

[52] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York.

[53] Garg, N. and Hochbaum, D. S. (1994). An o(log k) approximation algorithm for the k minimum spanning tree problem in the plane. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 432–438, New York, NY, USA. ACM.

[54] Glover, F. and Kochenberger, G., editors (2003). *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers.

[55] Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA.

[56] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press.

[57] Hamacher, H. W. and Jörnsten, K. (2010). *Optimal Relinquishment According to the Norwegian Petroleum Law: A Combinatorial Optimization Approach*, pages 443–457. Springer Berlin Heidelberg, Berlin, Heidelberg.

[58] Hamacher, H. W., Jörnsten, K., and Maffioli, F. (1991). Weighted k-cardinality trees. Technical report, Dipartimento di Elettronica, Politecnico di Milano, Italy.

[59] Hansen, P., Maniezzo, V., Fischetti, M., and Stützle, T., editors (2008). *Proceedings of Matheuristics 2008: Second International Workshop on Model Based Metaheuristics*, Bertinoro, Italy.

[60] Hansen, P., Mladenović, N., and Pérez, J. A. M. (2010). Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407.

[61] He, D. (2007). A novel greedy algorithm for the minimum common string partition problem. In Mandoiu, I. and Zelikovsky, A., editors, *Proceedings of ISBRA 2007 – Third International Symposium on Bioinformatics Research and Applications*, volume 4463 of *Lecture Notes in Computer Science*, pages 441–452. Springer.

[62] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI.

[63] Hsu, W. J. and Du, M. W. (1984). Computing a longest common subsequence for a set of strings. *BIT*, 24(1):45–59.

[64] Hu, B., Leitner, M., and Raidl, G. R. (2008). Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, 14(5):473–499.

[65] Hu, B. and Raidl, G. R. (2012a). An evolutionary algorithm with solution archive for the generalized minimum spanning tree problem. In Moreno-Díaz, R., Pichler, F., and Quesada-Arencibia, A., editors, *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, volume 6927 of *Lecture Notes in Computer Science*, pages 287–294. Springer.

[66] Hu, B. and Raidl, G. R. (2012b). An evolutionary algorithm with solution archives and bounding extension for the generalized minimum spanning tree problem. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 393–400, Philadelphia, PA, USA. ACM Press.

[67] Joslin, D. E. and Clements, D. P. (1999). "Squeaky Wheel" optimization. *Journal of Artificial Intelligence Research*, 10:353–373.

[68] Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948.

[69] Kennedy, J., Eberhart, R. C., and Shi, Y. (2004). *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, CA.

[70] Klau, G. W., Ljubić, I., Moser, A., Mutzel, P., Neuner, P., Pferschy, U., Raidl, G., and Weiskircher, R. (2004). Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In *Proceedings of GECCO 2004 – Genetic and Evolutionary Computation Conference*, volume 3102 of *Lecture Notes in Computer Science*, pages 1304–1315. Springer.

[71] Kleinberg, J. and Tardos, É. (2005). *Algorithm Design*. Addison-Wesley.

[72] Koulamas, C., Antony, S., and Jaen, R. (1994). A survey of simulated annealing applications to operations research problems. *Omega*, 22(1):41 – 56.

[73] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

[74] Laarhoven, P. J. M. and Aarts, E. H. L., editors (1987). *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, Norwell, MA, USA.

[75] Lodi, A., Milano, M., and Toth, P., editors (2010). *Proceedings of CPAIOR 2010 – 7th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*. Springer.

[76] López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., and Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*.

[77] Lourenço, H. R., Martin, O. C., and Stützle, T. (2010). *Iterated Local Search: Framework and Applications*, pages 363–397. Springer US, Boston, MA.

[78] Maffioli, F. (1991). *Finding a Best Subtree of a Tree*. Politecnico di Milano. Dipartimento di Elettronica.

[79] Maniezzo, V., Hansen, P., and Voss, S., editors (2006). *Proceedings of Matheuristics 2006: First International Workshop on Mathematical Contributions to Metaheuristics*, Bertinoro, Italy.

[80] Maniezzo, V., Stützle, T., and Voss, S., editors (2009). *Matheuristics – Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer.

[81] Massen, F., Deville, Y., and Van Hentenryck, P. (2012). Pheromone-based heuristic column generation for vehicle routing problems with black box feasibility. In *Proceedings of the 9th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, CPAIOR'12, pages 260–274, Berlin, Heidelberg. Springer-Verlag.

[82] Massen, F., López-Ibáñez, M., Stützle, T., and Deville, Y. (2013). *Experimental Analysis of Pheromone-Based Heuristic Column Generation Using irace*, pages 92–106. Springer Berlin Heidelberg, Berlin, Heidelberg.

[83] Meneses, C. N., Oliveira, C. A., and Pardalos, P. M. (2005). Optimization techniques for string selection and comparison problems in genomics. *IEEE Engineering in Medicine and Biology Magazine*, 24(3):81–87.

[84] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092.

[85] Mladenović, N. (1995). A variable neighborhood algorithm – a new metaheuristics for combinatorial optimization. In *Abstracts of Papers Presented at Optimization Days. Montreal*.

[86] Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

[87] Moscato, P. (1999). New ideas in optimization. chapter Memetic Algorithms: A Short Introduction, pages 219–234. McGraw-Hill Ltd., UK, Maidenhead, UK, England.

[88] Mousavi, S. R., Babaie, M., and Montazerian, M. (2012). An improved heuristic for the far from most strings problem. *Journal of Heuristics*, 18(2):239–262.

[89] Nepomuceno, N., Pinheiro, P., and Coelho, A. L. V. (2008). A hybrid optimization framework for cutting and packing problems. In Cotta, C. and van Hemert, J., editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 87–99. Springer.

[90] Nepomuceno, N. V., Pinheiro, P. R., and Coelho, A. L. V. (2007a). Combining metaheuristics and integer linear programming: A hybrid methodology applied to the container loading problem. In *Proceedings of the XX Congreso da Sociedade Brasileira de Computação, Concurso de Teses e Dissertações*, pages 2028–2032.

[91] Nepomuceno, N. V., Pinheiro, P. R., and Coelho, A. L. V. (2007b). Tackling the container loading problem: A hybrid approach based on integer linear programming and genetic algorithms. In Cotta, C. and van Hemert, J., editors, *Proceedings of EvoCOP 2007 – 7th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 4446 of *Lecture Notes in Computer Science*, pages 154–165. Springer.

[92] Pan, Q.-K., Tasgetiren, M. F., Suganthan, P. N., and Chua, T. J. (2011). A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Information Sciences*, 181(12):2455–2468.

[93] Papadimitriou, C. H. and Steiglitz, K. (1982). *Combinatorial Optimization - Algorithms and Complexity*. Dover Publications, New York.

[94] Pappalardo, E., Pardalos, P. M., and Stracquadanio, G. (2013). *Optimization Approaches for Solving String Selection Problems*. SpringerBriefs in Optimization. Springer, New York.

[95] Perron, L. and Trick, M. A., editors (2008). *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008*, volume 5015 of *Lecture Notes in Computer Science*. Springer.

[96] Philpott, H. W. and Wormald, N. (1997). On the optimal extraction of ore from an open-cast mine. Technical report, University of Auckland, New Zealand.

[97] Pinheiro, P. R., Coelho, A. L. V., de Aguiar, A. B., and Bonates, T. O. (2011). On the concept of density control and its application to a hybrid optimization framework: Investigation into cutting problems. *Computers & Industrial Engineering*, 61(3):463–472.

[98] Pinheiro, P. R., Coelho, A. L. V., de Aguiar, A. B., and de Menezes Sobreira Neto, A. (2012). Towards aid by generate and solve methodology: application in the problem of coverage and connectivity in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 2012. Article ID 790459.

[99] Rajasekaran, S., Hu, Y., Luo, J., Nick, H., Pardalos, P., Sahni, S., and Shaw, G. (2001a). Efficient algorithms for similarity search. *Journal of Combinatorial Optimization*, 5(1):125–132.

[100] Rajasekaran, S., Nick, H., Pardalos, P., Sahni, S., and Shaw, G. (2001b). Efficient algorithms for local alignment search. *Journal of Combinatorial Optimization*, 5(1):117–124.

[101] Ravi, R., Sundaram, R., Marathe, M. V., Rosenkrantz, D. J., and Ravi, S. S. (1994). Spanning trees short or small. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94, pages 546–555, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

[102] Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog.

[103] Reeves, C. R., editor (1993). *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Inc., New York, NY, USA.

[104] Resende, M. G. C. and Ribeiro, C. C. (2010). *GRASP: Greedy Randomized Adaptive Search Procedures*, volume Technical Report SGRASP2010. ATT Labs Research.

[105] Rodriguez, F. J., García-Martínez, C., Blum, C., and Lozano, M. (2012). An artificial bee colony algorithm for the unrelated parallel machines scheduling problem. In Coello Coello, C. A., Cutello, V., Deb, K., Forrest, S., Nicosia, G., and Pavone, M., editors, *Proceedings of PPSN XII – 12th International Conference on Parallel Problem Solving from Nature*, volume 7492 of *Lecture Notes in Computer Science*, pages 143–152. Springer.

[106] Rossi, F., van Beek, P., and Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier.

[107] Rothberg, E. (2007). An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541.

[108] Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

[109] Saraiva, R. D., Nepomuceno, N. V., and Pinheiro, P. R. (2013). The generate-and-solve framework revisited: Generating by simulated annealing. In Middendorf, M. and Blum, C., editors, *Evolutionary Computation in Combinatorial Optimization*, volume 7832 of *Lecture Notes in Computer Science*, pages 262–273. Springer.

[110] Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In Maher, M. and Puget, J.-F., editors, *Principle and Practice of Constraint Programming – CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer.

[111] Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197.

[112] Stützle, T. (1998). *Local Search Algorithms for Combinatorial Problems — Analysis, Improvements, and New Applications*. PhD thesis, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany.

[113] Stützle, T. and Hoos, H. H. (2000). MAX-MIN ant system. *Future Generation Computer Systems*, 16(8):889–914.

[114] Suman, B. and Kumar, P. (2006). A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(18):1143–1160.

[115] Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley & Sons.

[116] Uehara, R. (1999). The number of connected components in graphs and its applications. Technical report, Natural Science Faculty, Komazawa University, Japan.

[117] Urosevic, D., Brimberg, J., and Mladenovic, N. (2004). Variable neighborhood decomposition search for the edge weighted k-cardinality tree problem. *Computers & Operations Research*, 31(8):1205–1213.

[118] van Hoeve, W.-J. and Hooker, J. N., editors (2009). *Proceedings of CPAIOR 2009 – 6th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*. Springer.

[119] Voudouris, C. (1997). *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, Department of Computer Science, University of Essex. pp. 166.

[120] Voudouris, C. and Tsang, E. (1999). Guided Local Search. *European Journal of Operational Research*, 113(2):469–499.

[121] Yagiura, M. and Ibaraki, T. (1996). The use of dynamic programming in genetic algorithms for permutation problems. *European Journal of Operational Research*, 92(2):387–401.

Part II

Publicaciones seleccionadas

6

Mathematical Programming Strategies for Solving the Minimum Common String Partition Problem

European Journal of Operational Research 242 (2015) 769–777

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Mathematical programming strategies for solving the minimum common string partition problem

Christian Blum [a,b,*], José A. Lozano [a], Pinacho Davidson [a,c]

[a] Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, San Sebastian, Spain
[b] IKERBASQUE, Basque Foundation for Science, Bilbao, Spain
[c] Escuela de Informática, Universidad Santo Tomás, Concepción, Chile

ARTICLE INFO

Article history:
Received 8 April 2014
Accepted 23 October 2014
Available online 4 November 2014

Keywords:
Minimum common string partition
Integer linear programming
Heuristic

ABSTRACT

The minimum common string partition problem is an NP-hard combinatorial optimization problem with applications in computational biology. In this work we propose the first integer linear programming model for solving this problem. Moreover, on the basis of the integer linear programming model we develop a deterministic 2-phase heuristic which is applicable to larger problem instances. The results show that provenly optimal solutions can be obtained for problem instances of small and medium size from the literature by solving the proposed integer linear programming model with CPLEX. Furthermore, new best-known solutions are obtained for all considered problem instances from the literature. Concerning the heuristic, we were able to show that it outperforms heuristic competitors from the related literature.

1. Introduction

Optimization problems related to strings—such as protein or DNA sequences—are very common in bioinformatics. Examples include string selection problems (Meneses, Oliveira, & Pardalos, 2005; Mousavi, Babaie, & Montazerian, 2012; Pappalardo, Pardalos, & Stracquadanio, 2013), the longest common subsequence problem and its variants (Hsu & Du, 1984; Smith & Waterman, 1981), alignment problems (Gusfield, 1997; Rajasekaran, Nick, Pardalos, Sahni, & Shaw, 2001), and similarity search (Rajasekaran, Hu, Luo, Nick, Pardalos, Sahni, & Shaw, 2001). These problems are often computationally very hard, if not even *NP*-hard (Garey & Johnson, 1979). In this work we deal with the *minimum common string partition* (MCSP) problem, which can be described as follows. We are given two related input strings that have to be partitioned each into the same collection of substrings. The size of the collection is subject to minimization. A formal description of the problem will be provided in Section 1.1. The MCSP problem has applications, for example, in the bioinformatics field. Chen, Zheng, Fu, Nan, Zhong, Lonardi, and Jiang (2005) point out that the MCSP problem is closely related to the problem of sorting by reversals with duplicates, a key problem in genome rearrangement.

In this paper we introduce the first integer linear program (ILP) for solving the MCSP problem. An experimental evaluation on problem instances from the related literature shows that this ILP can be efficiently solved, for example, by using any version of IBM ILOG CPLEX. However, a study on new instances of larger size demonstrates the limitations of the model. Therefore, we additionally introduce a deterministic 2-phase heuristic which is strongly based on the original ILP. The experimental evaluation shows that the heuristic is applicable to larger problem instances than the original ILP. Moreover, it is shown that the heuristic outperforms competitor algorithms from the related literature on known problem instances.

1.1. Problem description

The MCSP problem can technically be described as follows. Given are two input strings s_1 and s_2, both of length n over a finite alphabet Σ. These two strings are required to be *related*, which means that each letter appears the same number of times in each of them. Note that this definition implies that s_1 and s_2 have the same length. A valid solution to the MCSP problem is obtained by partitioning s_1 into a set P_1 of non-overlapping substrings, and s_2 into a set P_2 of non-overlapping substrings, such that $P_1 = P_2$. Moreover, we are interested in finding a valid solution such that $|P_1| = |P_2|$ is minimal.

Consider the following example. Given are DNA sequences $s_1 =$ **AGACTG** and $s_2 =$ **ACTAGG**. Obviously, s_1 and s_2 are related because **A** and **G** appear twice in both input strings, while **C** and **T** appear once. A trivial valid solution can be obtained by partitioning both strings into substrings of length 1, that is, $P_1 = P_2 = \{$**A**, **A**, **C**, **T**, **G**, **G**$\}$. The

* Corresponding author at: Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, San Sebastian, Spain. Tel.: +34 943017119.
E-mail addresses: christian.blum@ehu.es (C. Blum), ja.lozano@ehu.es (J. A. Lozano), ppinacho@santotomas.cl (P. Davidson).

http://dx.doi.org/10.1016/j.ejor.2014.10.049
0377-2217/

objective function value of this solution is 6. However, the optimal solution, with objective function value 3, is $P_1 = P_2 = \{\mathbf{ACT}, \mathbf{AG}, \mathbf{G}\}$.

1.2. Related work

The MCSP problem has been introduced by Chen et al. (2005) due to its relation to genome rearrangement. More specifically, it has applications in biological questions such as: May a given DNA string possibly be obtained by rearrangements of another DNA string? The general problem has been shown to be *NP*-hard even in very restrictive cases (Goldstein, Kolman, & Zheng, 2005). Other papers concerning problem hardness consider, for example, the *k*-MCSP problem, which is the version of the MCSP problem in which each letter occurs at most *k* times in each input string. The 2-MCSP problem was shown to be APX-hard in Goldstein et al. (2005). When the input strings are over an alphabet of size *c*, the corresponding problem is denoted as MCSP^c. Jiang et al. proved that the decision version of the MCSP^c problem is *NP*-complete when $c \geq 2$ (Jiang, Zhu, Zhu, & Zhu, 2012).

The MCSP has been considered quite extensively by researchers dealing with the approximability of the problem. Cormode and Muthukrishnan (2007), for example, proposed an $O(log n log^* n)$-approximation for the *edit distance with moves* problem, which is a more general case of the MCSP problem. Shapira and Storer (2002) extended on this result. Other approximation approaches for the MCSP problem have been proposed in Kolman and Waleń (2007). In this context, Chrobak, Kolman, and Sgall (2004) studied a simple greedy approach for the MCSP problem, showing that the approximation ratio concerning the 2-MCSP problem is 3, and for the 4-MCSP problem the approximation ratio is $\Omega(\log(n))$. In the case of the general MCSP problem, the approximation ratio is between $\Omega(n^{0.43})$ and $O(n^{0.67})$, assuming that the input strings use an alphabet of size $O(\log(n))$. Kaplan and Shafrir (2006) raised the lower bound to $\Omega(n^{0.46})$. Kolman proposed a modified version of the simple greedy algorithm with an approximation ratio of $O(k^2)$ for the *k*-MCSP (Kolman, 2005). Recently, Goldstein and Lewenstein proposed a greedy algorithm for the MCSP problem that runs in $O(n)$ time (see Goldstein & Lewenstein, 2011). He (2007) introduced a greedy algorithm with the aim of obtaining better average results.

Damaschke (2008) was the first one to study the fixed-parameter tractability (FPT) of the problem. Later, Jiang et al. (2012) showed that both the *k*-MCSP and MCSP^c problems admit FPT algorithms when *k* and *c* are constant parameters. Finally, Fu, Jiang, Yang, and Zhu (2011) and Ding and Fu (2013) proposed an $O(2^{nn^{O(1)}})$ time algorithm for the general case and an $O(n(logn)^2)$ time algorithm applicable under some constraints.

To our knowledge, the only metaheuristic approaches that have been proposed in the related literature for the MCSP problem are (1) the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System by Ferdous and Sohel Rahman (2014, 2013) and (2) the probabilistic tree search algorithm by Blum, Lozano, and Pinacho Davidson (2014). Both works applied their algorithm to a range of artificial and real DNA instances from Ferdous and Sohel Rahman (2013).

1.3. Organization of the paper

The remaining part of the paper is organized as follows. In Section 2, the ILP model for solving the MCSP is outlined. Moreover, an experimental evaluation is provided. The deterministic heuristic, together with an experimental evaluation, is described in Section 3. Finally, in Section 4 we provide conclusions and an outlook to future work.

2. An integer linear program to solve the MCSP

In the following we present the first ILP model for solving the MCSP. For this, the definitions provided in the following are required. Note that an illustrative example is provided in Section 2.3.

2.1. Preliminaries

Henceforth, a *common block* b_i of input strings s_1 and s_2 is denoted as a triple $(t_i, k1_i, k2_i)$ where t_i is a string which can be found starting at position $1 \leq k1_i \leq n$ in string s_1 and starting at position $1 \leq k2_i \leq n$ in string s_2. Moreover, let $B = \{b_1, \ldots, b_m\}$ be the (ordered) set of all possible common blocks of s_1 and s_2.[1] Given the definition of *B*, any valid solution $\mathcal{S}$ to the MCSP problem is a subset of *B*—that is, $\mathcal{S} \subset B$—such that:

1. $\sum_{b_i \in \mathcal{S}} |t_i| = n$, that is, the sum of the length of the strings corresponding to the common blocks in $\mathcal{S}$ is equal to the length of the input strings.
2. For any two common blocks $b_i, b_j \in \mathcal{S}$ it holds that their corresponding strings overlap neither in s_1 nor in s_2.

Moreover, any (valid) partial solution $\mathcal{S}_{\text{partial}}$ is a subset of *B* fulfilling the following conditions: (1) $\sum_{b_i \in \mathcal{S}_{\text{partial}}} |t_i| < n$ and (2) for any two common blocks $b_i, b_j \in \mathcal{S}_{\text{partial}}$ it holds that their corresponding strings overlap neither in s_1 nor in s_2. Note that any valid partial solution can be extended to be a valid solution. Furthermore, given a partial solution $\mathcal{S}_{\text{partial}}$, set $B(\mathcal{S}_{\text{partial}}) \subset B$ denotes the set of common blocks that may be used in order to extend $\mathcal{S}_{\text{partial}}$ such that the result is again a valid (partial) solution.

2.2. The integer linear program

First, two binary $m \times n$ matrices $M1$ and $M2$ are defined as follows. In both matrices, row $1 \leq i \leq m$ corresponds to common block $b_i \in B$. Moreover, a column $1 \leq j \leq n$ corresponds to position *j* in input string s_1, respectively s_2. In general, the entries of matrix $M1$ are set to zero. However, in each row *i*, the positions that string t_i (of common block b_i) occupies in input string s_1 are set to one. Correspondingly, the entries of matrix $M2$ are set to zero, apart from the fact that in each row *i* the positions occupied by string t_i in input string s_2 are set to one. Henceforth, the position (i, j) of a matrix *M* is denoted by $M_{i,j}$. Finally, we introduce for each common block $b_i \in B$ a binary variable x_i. With these definitions we can express the MCSP in form of the following integer linear program, henceforth referred to by ILP_{orig}.

$$\textbf{min} \sum_{i=1}^{m} x_i \tag{1}$$

subject to :

$$\sum_{i=1}^{m} M1_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{m} M2_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \ldots, n \tag{3}$$

$$x_i \in \{0, 1\} \quad \text{for } i = 1, \ldots, m$$

Hereby, the objective function minimizes the number of selected common blocks. Constraints (2) make sure that the strings corresponding to the selected common blocks do not overlap in input string s_1, while constraints (3) make sure that the strings corresponding to the selected common blocks do not overlap in input string s_2. Moreover, note that constraints (2) and (3) implicitly ensure that the sum of the length of the strings corresponding to the selected common blocks is equal to *n*.

2.3. Example

As an example, consider the small problem instance from Section 1.1. The complete set of common blocks (*B*) as induced by

[1] The way in which *B* is ordered is of no importance.

Table 1
Results for the 10 instances of GROUP1.

id	GREEDY	ACO	TRESEA	ILP_{orig}			
	Value	Best	Best	Value	Time (seconds)	Gap (percent)	$\|\mathbf{B}\|$
1	46	42	42	*41	1	0.0	4299
2	56	51	48	*47	2	0.0	6211
3	62	55	56	*52	34	0.0	8439
4	46	43	43	*41	1	0.0	4299
5	44	43	41	*40	1	0.0	4718
6	48	42	41	*40	3	0.0	4435
7	65	60	60	*55	38	0.0	8687
8	51	47	45	*43	2	0.0	4995
9	46	45	43	*42	2	0.0	4995
10	63	59	58	*54	51	0.0	9699
Avg.	52.7	48.7	47.7	45.5	13.5	0.0	6029.3

input strings $s_1 =$ **AGACTG** and $s_2 =$ **ACTAGG** is as follows:

$$B = \begin{Bmatrix} b_1 &= (\mathbf{ACT}, 3, 1) \\ b_2 &= (\mathbf{AG}, 1, 4) \\ b_3 &= (\mathbf{AC}, 3, 1) \\ b_4 &= (\mathbf{CT}, 4, 2) \\ b_5 &= (\mathbf{A}, 1, 1) \\ b_6 &= (\mathbf{A}, 1, 4) \\ b_7 &= (\mathbf{A}, 3, 1) \\ b_8 &= (\mathbf{A}, 3, 4) \\ b_9 &= (\mathbf{C}, 4, 2) \\ b_{10} &= (\mathbf{T}, 5, 3) \\ b_{11} &= (\mathbf{G}, 2, 5) \\ b_{12} &= (\mathbf{G}, 2, 6) \\ b_{13} &= (\mathbf{G}, 6, 5) \\ b_{14} &= (\mathbf{G}, 6, 6) \end{Bmatrix}$$

Given set B, matrices $M1$ and $M2$ are the following ones:

$$M1 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad M2 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The optimal solution to this instance is $\mathcal{S} = \{b_1, b_2, b_{14}\}$. It can easily be verified that this solution respects constraints (2)–(4) of the ILP model.

2.4. Experimental evaluation

In the following we will provide an experimental evaluation of model ILP_{orig}. The model was implemented in ANSI C++ using GCC 4.7.3 for compiling the software. Moreover, the model was solved with IBM ILOG CPLEX V12.1. The experimental results that we outline in the following were obtained on a cluster of PCs with "Intel(R) Xeon(R) CPU 5130" CPUs of 4 nuclei of 2000 megahertz and 4 gigabytes of RAM.

2.4.1. Problem instances

For testing model ILP_{orig} we chose the same set of benchmark instances that was used by in Ferdous and Sohel Rahman (2013) for the experimental evaluation of their ant colony optimization approach. This set contains, in total, 30 artificial instances and 15 real-life instances consisting of DNA sequences. Remember, in this context, that each problem instance consists of two related input strings. Moreover, the benchmark set consists of four subsets of instances. The first subset (henceforth labeled GROUP1) consists of 10 artificial instances in which the input strings are maximally of length 200. The second set (GROUP2) consists of 10 artificial instances with input string lengths between 201 and 400. In the third set (GROUP3) the input strings of the 10 artificial instances have lengths between 401 and 600. Finally, the fourth set (REAL) consists of 15 real-life instances of various lengths from [200, 600].

2.4.2. Results

The results are shown in Tables 1–4, in terms of one table per instance set. The structure of these tables is as follows. The first column provides the instance identifiers. The second column contains the results of the greedy algorithm from Chrobak et al. (2004) (results were taken from Ferdous and Sohel Rahman (2013)). The third column provides the value of the best solution found in four independent runs per problem instance (with a CPU time limit of 7200 seconds per run) by the Aco approach by Ferdous and Sohel Rahman (2014, 2013).[2] The fourth column provides the value of the best solution found in 10 independent runs per problem instance (with a CPU time limit of 1000 seconds per run) by the probabilistic tree search algorithm (henceforth labeled TRESEA) by Blum et al. (2014). TRESEA was run on the same machines as the ones used for the current work. Finally, the last four table columns are dedicated to the presentation of the results provided by solving model ILP_{orig}. The first one of these columns provides the value of the best solution found within 3600 CPU seconds. In case the optimality of the corresponding solution was proved by CPLEX, the value is marked by an asterisk. The second column dedicated to ILP_{orig} provides the computation time (in seconds). In case of having solved the corresponding problem to optimality, this column only displays one value indicating the time needed by CPLEX to solve the problem. Otherwise, this column provides two values in the form X/Y, where X corresponds to the time at which CPLEX was able to find the first valid solution, and Y corresponds to the time at which CPLEX found the best solution within 3600 CPU seconds. The third one of the columns dedicated to ILP_{orig} shows the optimality gap, which refers to the gap between the value of the best valid solution and the current lower bound at the time of stopping a run. Finally, the last column indicates the size of set B, that is, the size of the complete set of common blocks. Note that this value corresponds to the number

[2] In this context, note that the experiments for Aco were performed on a computer with an "Intel(R) 2 Quad" CPU with 2.33 gigahertz and 4 gigabytes of RAM.

Table 2
Results for the 10 instances of Group2.

id	Greedy	Aco	TreSea	ILP_{orig}			
	Value	Best	Best	Value	Time (seconds)	Gap (percent)	$\|\mathbf{B}\|$
1	119	113	111	98	50/2067	2.9	37743
2	122	118	114	106	80/1046	7.5	47174
3	114	111	107	97	35/1220	2.7	36979
4	116	115	111	102	48/891	4.9	40960
5	135	132	127	116	83/2703	6.7	52697
6	108	105	102	93	39/1476	5.6	35650
7	108	98	96	88	31/3107	6.0	30839
8	123	118	114	104	61/3248	5.1	42668
9	124	119	113	104	49/1563	5.2	42998
10	105	101	98	89	27/1397	3.6	31169
Avg.	117.4	113.0	109.3	99.7	50/1872	5.0	39887.7

Table 3
Results for the 10 instances of Group3.

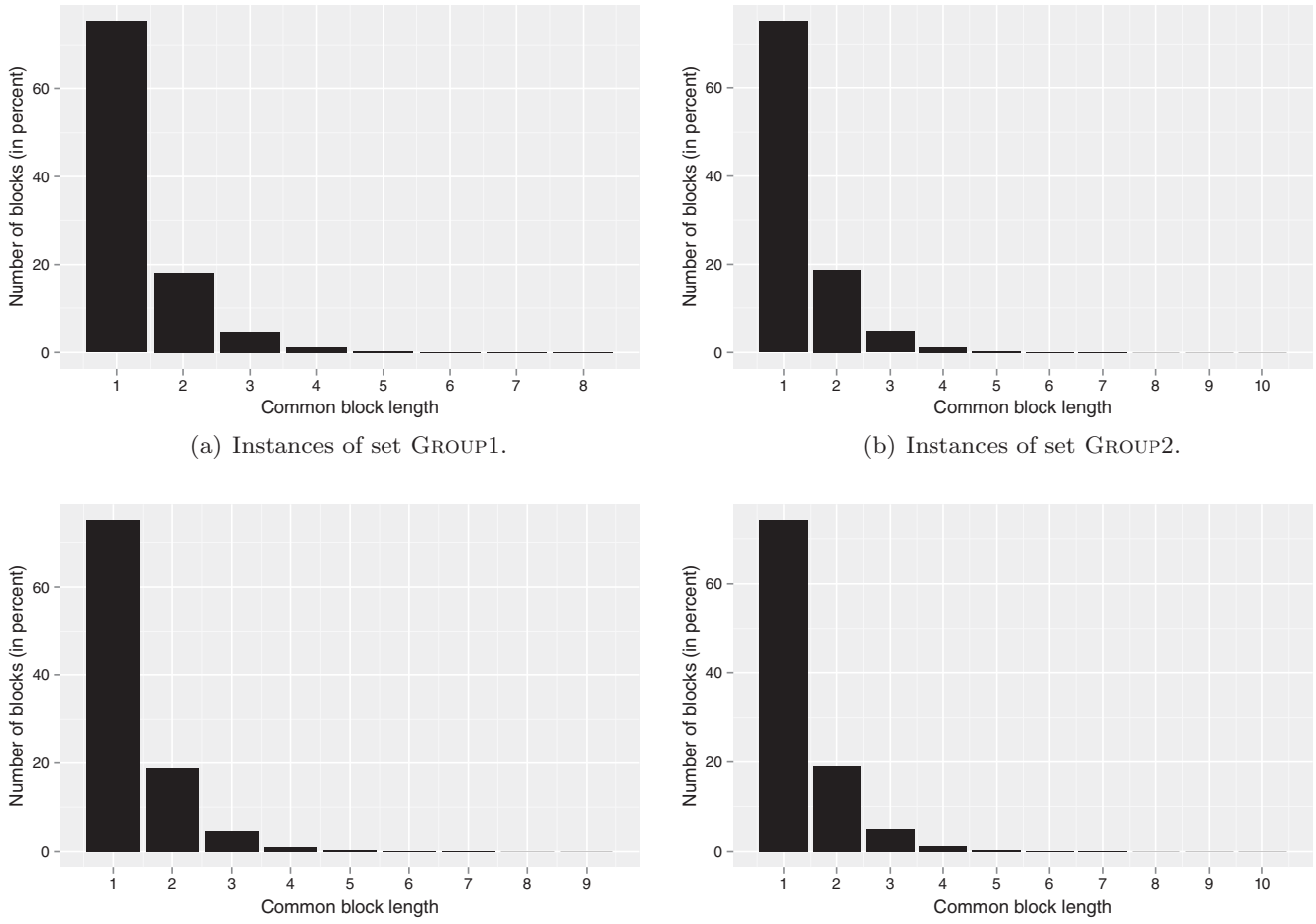
id	Greedy	Aco	TreSea	ILP_{orig}			
	Value	Best	Best	Value	Time (seconds)	Gap (percent)	$\|\mathbf{B}\|$
1	182	177	171	155	333/858	7.5	110973
2	175	175	168	155	345/693	7.7	102670
3	196	187	185	166	462/2063	8.5	119287
4	192	184	179	159	458/976	6.9	114975
5	176	171	163	150	279/682	9.7	99775
6	170	160	162	147	239/573	9.1	88839
7	173	167	161	149	253/620	9.8	95765
8	185	175	169	151	312/3591	6.7	97400
9	174	172	169	158	352/1022	10.9	104186
10	171	167	161	148	343/1334	9.1	98237
Avg.	179.4	173.5	168.8	153.8	338/1241	8.6	103211.0

Table 4
Results for the 15 instances of set Real.

id	Greedy	Aco	TreSea	ILP_{orig}			
	Value	Best	Best	Value	Time (seconds)	Gap (percent)	$\|\mathbf{B}\|$
1	95	87	86	*78	968	0.0	22799
2	161	155	154	139	196/441	9.2	80523
3	121	116	113	104	61/3575	5.6	45869
4	173	164	158	144	301/1353	6.5	91663
5	172	171	165	150	379/1998	7.9	108866
6	153	145	143	128	170/3584	6.5	70655
7	140	140	131	121	180/1814	6.9	73502
8	134	130	128	116	127/3268	6.8	65560
9	149	146	142	131	191/358	8.8	75833
10	151	148	144	130	144/3429	6.1	69560
11	126	124	121	110	114/3591	4.8	56160
12	143	137	138	126	178/651	9.8	70861
13	180	180	171	157	469/2236	7.1	115810
14	152	147	146	130	161/3099	6.7	73449
15	157	160	152	139	295/1430	7.7	91060
Avg.	147.1	143.3	139.5	126.9	212/2120	6.7	74163.9

of variables used by ILP_{orig}. The best result (among all algorithms) for each problem instance is marked by a grey background, and the last row of each table provides averages over the whole table.

The following conclusions can be drawn when analyzing the results. First, CPLEX is able to solve all instances of Group1 to optimality. This is done, on average, in about 13 seconds. Moreover, none of the existing algorithms was able to find any of these optimal solutions. Second, CPLEX was also able to find new best-known solutions for all remaining 35 problem instances, even though it was not able to prove optimality within 3600 CPU seconds, which is indicated by the positive optimality gaps. An exception is instance 1 of set Real which also could be solved to optimality. Third, the improvements over the competitor algorithms obtained by solving ILP_{orig} with CPLEX are remarkable. In particular, the average improvement (in percent) over



(a) Instances of set GROUP1. (b) Instances of set GROUP2.

(c) Instances of set GROUP3. (d) Instances of set REAL.

Fig. 1. Distribution of the string lengths corresponding to the complete set of common blocks. The distributions are shown averaged over all instances for each of the four sets of problem instances.

TRESEA, the best competitor from the literature, is 4.8 percent in the case of GROUP1, 9.2 percent in the case of GROUP2, 9.7 percent in the case of GROUP3, and 9.9 percent in the case of REAL.

In order to study the limits of solving ILP_{orig} with CPLEX we randomly generated larger DNA instances. In particular, we generated one random instance for each input string size from {800, 1000, 1200, 1400, 1600, 1800, 2000}. CPLEX was stopped when at least 3600 CPU seconds had passed and at least one feasible solution had been found. However, if after 12 CPU hours still no feasible solution was found, the execution was stopped as well. The results are shown in Table 5. The first column of this table provides the length of the corresponding random instance. The remaining four columns contain the same information as already explained in the context of Tables 1–4, just that column Time (seconds) simply provides the computation time (in seconds) at which the best solution was found. Analyzing the results we can observe that the application of CPLEX to ILP_{orig} quickly becomes unpractical with growing input string size. For example, the first valid solution for the instance with string length 1600 was found after 10987 seconds. Concerning the largest problem instance, no valid solution was found within 12 CPU hours.

Table 5
Results of applying CPLEX to ILP_{orig} in the context of larger instances.

Length	Value	Time (seconds)	Gap (percent)	$\lvert\mathbf{B}\rvert$
800	210	2531	10.7	214622
1000	304	1673	26.4	334411
1200	342	3435	22.6	480908
1400	401	6459	24.9	653401
1600	442	10987	24.1	854500
1800	486	18276	24.0	1084533
2000	n.a.	n.a.	n.a.	1335893

3. A MIP-based heuristic

As shown at the end of the previous section, the application of CPLEX to ILP_{orig} reaches its limits starting from an input string size of about 1200. However, if it were possible to considerably reduce the size of the set of common blocks (B), mathematical programming might still be an option to obtain good (heuristic) solutions. With this idea in mind we studied the distribution of the lengths of the strings of the common blocks in B for all 45 problem instances. This distribution is shown—averaged over the instances of each of the four instance sets—in Fig. 1. Analyzing these distributions it can be observed, first of all, that the distribution does not seem to depend on instance size.[3] However, the important aspect to observe is that around 75 percent of all the common blocks contain strings of length 1. Moreover, only a very small portion of these common blocks will form part of an optimal solution. In comparison, it is reasonable to assume that a much larger percentage of the blocks corresponding to large strings will form part of an optimal solution. These observations gave rise to the heuristic which is outlined in the following.

[3] Most probably the distribution would change in some way when changing the size of the alphabet.

3.1. Heuristic

The proposed heuristic works in two phases. In the first phase, a subset of B (the complete set of common blocks) must be chosen. For this purpose, let $B_{\geq l}$ (where $l \geq 1$) denote the subset of B that contains all common blocks b_i from B with $|t_i| \geq l$, that is, all blocks whose corresponding string is longer or equal than l. Note, in this context, that $B_{\geq 1} = B$. Moreover, note that $|B_{\geq 1}| \geq |B_{\geq 2}| \geq |B_{\geq 3}| \geq \cdots \geq |B_{\geq \infty}|$. Let $l_{\max}$ be the smallest value for l such that $|B_{\geq l_{\max}}| > 0$. Observe that $B_{\geq l_{\max}}$ only contains the common blocks with the longest strings. Having chosen a specific value for l from $[2, l_{\max}]$, the following ILP, henceforth referred to as ILP_{ph1}, may be solved.

$$\textbf{min} \sum_{b_i \in B_{\geq l}} x_i + y \tag{4}$$

subject to :

$$\sum_{b_i \in B_{\geq l}} M1_{i,j} \cdot x_i \leq 1 \quad \text{for } j = 1, \ldots, n \tag{5}$$

$$\sum_{b_i \in B_{\geq l}} M2_{i,j} \cdot x_i \leq 1 \quad \text{for } j = 1, \ldots, n \tag{6}$$

$$\sum_{b_i \in B_{\geq l}} |t_i| \cdot x_i = n - y \tag{7}$$

$$x_i \in \{0, 1\} \quad \text{for } b_i \in B_{\geq l}$$
$$y \in \{0, 1, \ldots, n\}$$

ILP_{ph1} is based on a binary variable x_i for each common block $b_i \in B_{\geq l}$, and an additional variable $y \in \{0, 1, \ldots, n\}$ that counts the number of positions that are not covered by any chosen block. Moreover, matrices $M1$ and $M2$ are the same as the ones introduced in Section 2.2, that is, they are defined over the whole set B. The objective function minimizes the number of chosen blocks plus the number of uncovered positions. The constraints (5)–(6) are the same as in ILP_{orig} (see Section 2.2), apart from the fact that all equality symbols are replaced by the $\leq$-symbol. Moreover, in constraints (7) variable y is added to the right-hand side in order to account for the number of uncovered positions. In short, the idea of ILP_{ph1} is to produce a partial solution for the original MCSP that covers as much as possible of both input strings, while choosing as few common blocks as possible.

Solving ILP_{ph1} will henceforth be referred to as *phase 1* of the proposed heuristic. Let us denote by $\mathcal{S}_{ph1}$ the solution provided by phase 1.[4] Due to the constraints of ILP_{ph1} this solution is a valid partial solution to the original MCSP problem. The idea of the second phase is then to produce the best complete solution possible that contains $\mathcal{S}_{ph1}$. This is done by solving the following ILP, henceforth referred to as ILP_{ph2}.

$$\textbf{min} \sum_{b_i \in B_{ph2} \cup \mathcal{S}_{ph1}} x_i \tag{8}$$

subject to :

$$\sum_{b_i \in B_{ph2} \cup \mathcal{S}_{ph1}} M1_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \ldots, n \tag{9}$$

$$\sum_{b_i \in B_{ph2} \cup \mathcal{S}_{ph1}} M2_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \ldots, n \tag{10}$$

$$x_i = 1 \quad \text{for } b_i \in \mathcal{S}_{ph1} \tag{11}$$
$$x_i \in \{0, 1\} \quad \text{for } b_i \in B_{ph2}$$

Hereby, $B_{ph2} := B(\mathcal{S}_{ph1}) \subset B$ is the set of common blocks that may be added to $\mathcal{S}_{ph1}$ without violating any constraints.[5] Note that model ILP_{ph2} is the same as model ILP_{orig}, just that ILP_{ph2} only considers common blocks from B_{ph2} and that it forces any solution to contain all common blocks from $\mathcal{S}_{ph1}$; see constraints (11). This completes the description of the heuristic.

[4] Remember that solutions are subsets of B.

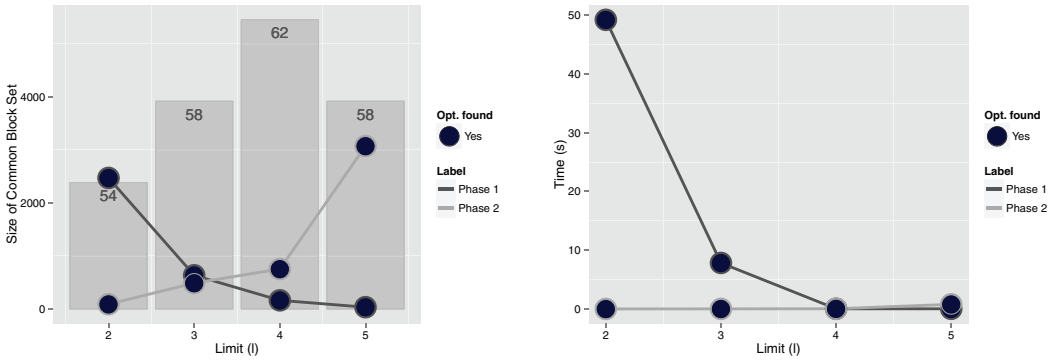
[5] See Section 2.1 for the definition of $B(\cdot)$.

Table 6
Results of the heuristic. Each of the four subtables deals with one of the four problem instance sets. Note that Best Known refers to the best heuristic results known from the literature.

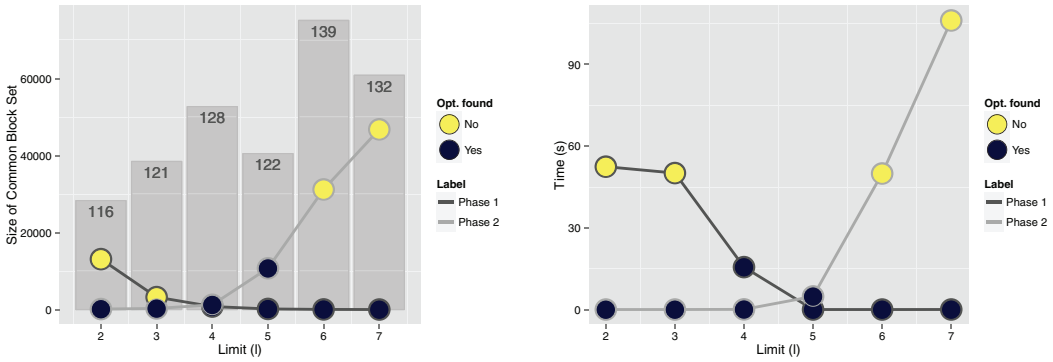
id	Best Known	Heuristic	
		Value	Time (seconds)
(a) Instances of Group1.			
1	42	+**41**	1
2	48	+**47**	2
3	55	+**52**	43
4	43	+**41**	2
5	41	+**40**	2
6	41	+**40**	3
7	60	+**55**	7
8	45	+**43**	3
9	43	+**42**	2
10	58	+**54**	58
Avg.	47.7	**45.5**	12.3
(b) Instances of Group2.			
1	111	**100**	209
2	114	**107**	248
3	107	**98**	297
4	111	**103**	268
5	127	+**116**	280
6	102	**96**	224
7	96	**90**	164
8	114	+**104**	232
9	113	**108**	157
10	98	**92**	159
Avg.	109.3	**101.4**	223.8
(c) Instances of Group3.			
1	171	**161**	674
2	168	**162**	479
3	185	**171**	980
4	179	**167**	786
5	163	**153**	431
6	160	**151**	499
7	161	**154**	228
8	169	**160**	315
9	169	**158**	779
10	161	**154**	459
Avg.	168.6	**159.1**	563.0
(d) Instances of Real.			
1	86	**80**	130
2	154	+**139**	563
3	113	**105**	223
4	158	+**142**	560
5	165	**161**	494
6	143	**133**	401
7	131	**123**	433
8	128	**121**	404
9	142	+**131**	398
10	144	**133**	448
11	121	**113**	367
12	137	**129**	319
13	171	**161**	1105
14	146	**136**	553
15	152	**144**	405
Avg.	139.4	**130.1**	453.5

3.2. Experimental evaluation

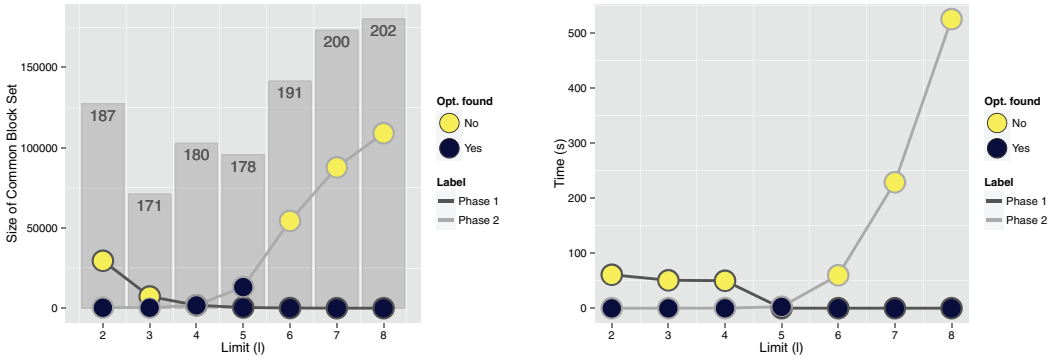
Just like model ILP_{orig}, the heuristic was implemented in ANSI C++ using GCC 4.7.3 for compiling the software. The two ILP models were solved with IBM ILOG CPLEX V12.1, and the same machines as for the experimental evaluation of ILP_{orig} were used for running the experiments.



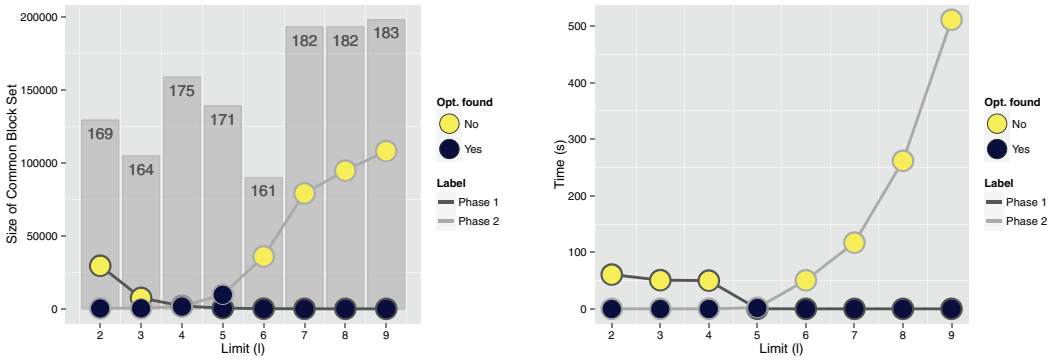
(a) Instance 10 of GROUP1, results (left) and computation time (right)



(b) Instance 5 of GROUP2, results (left) and computation time (right)



(c) Instance 3 of GROUP3, results (left) and computation time (right)



(d) Instance 13 of REAL, results (left) and computation time (right)

Fig. 2. Detailed information on the results of the proposed heuristic for four chosen problem instances. The description of the information content of the graphics is provided in the text.

As mentioned before, the heuristic may be applied for any value of l from the interval $[2, l_{max}]$. In fact, we applied the heuristic to each of the 45 problem instances from sets GROUP1, GROUP2, GROUP3, and REAL, with all possible values for l. In order not to spend too much computation time the following stopping criterion was used for each call to CPLEX concerning any of the two involved ILP models. CPLEX was stopped (1) in case a provenly optimal solution was obtained or (2) in case at least 50 CPU seconds were spent and the first valid solution was obtained. The overall result of the heuristic for a specific problem instance is the value of the best solution found for any value of l. Moreover, as computation time we provide the sum of the computation times spend for all applications for different values of l.

The results are shown in Table 6, which contains one subtable for each of the four instance sets. Each subtable has the following format. The first column provides the instance identifier. The second column contains the value of the best heuristic solution found in the literature. Finally, the last two table columns present the results of our heuristic. The first one of these columns contains the value of the best solution generated by the heuristic, while the second column provides the total computation time (in seconds). The last row of each subtable presents averages over the whole subtable. Moreover, the best result for each instance is indicated in bold font, and those cases in which the result of applying CPLEX to ILP_{orig} could be matched are marked by a "+" symbol.

The results allow to make the following observations. First, our heuristic is able to improve the best-known heuristic result from the literature in all 45 cases. Overall, the heuristic improves by 6.5 percent (on average) over the best known heuristic results from the literature. On the downside, the heuristic is only able to match the results of applying CPLEX to model ILP_{orig} in 15 out of 45 cases. However, this changes with growing instance size, as we will show later in Section 3.4.

3.3. Gaining insight into the behavior of the heuristic

With the aim of gaining more insight into the behavior of the heuristic with respect to the choice of a value for parameter l, the following information is presented in graphical form in Fig. 2. Two graphics are shown for each of the four chosen problem instances. More precisely, we chose to present information for the largest problem instances from each of the four instance sets (see subfigures (a) to (d) of Fig. 2). The left graphic of each subfigure has to be read as follows. The x-axis ranges over the possible values for l, while the y-axis indicates the size of the set of common blocks that is used for solving models ILP_{ph1} and ILP_{ph2}. The graphic shows two curves. The one with a black line concerns solving model ILP_{ph1} in phase 1 of the heuristic, while the other one (shown by means of a grey line) concerns solving model ILP_{ph2} in phase two of the heuristic. The dots indicate for each value of l the size of the set of common blocks used by the corresponding models. Moreover, in case the interior of a dot is light-grey (yellow in the online version) this means that the corresponding model could not be solved to optimality within 50 CPU seconds, while a black interior of a dot indicates that the corresponding model was solved to optimality. Finally, the bars in the background of the graphic present the values of the solutions that were generated with different values of l. The graphics on the right hand side present the corresponding computation times required by solving the different models.

The following observations can be made. When the value of l is close to the lower or the upper bound—that is, either close to 2 or close to l_{max}—one of the two involved sets of common blocks is quite large, and, therefore, the computation time needed for solving the corresponding ILP may be large, in particular when the input instance is rather large. On the contrary, for intermediate values of l, the size of both involved sets of common blocks is moderate, and, therefore, CPLEX is rather fast in providing solutions, even if the optimal solution is not found (or cannot be proven) within 50 CPU seconds. Moreover, the value of l with which the best results are obtained seems difficult to be predicted. While rather small values of l are required for the first three of the chosen problem instances, an intermediate value of l is best in case of instance 13 of REAL.

Table 7
Results of applying the heuristic in the context of larger instances.

Length	CPLEX	Heuristic				
	Value	Value	Time (seconds)	$\lvert B_{\geq 5}\rvert$	$\lvert B_{ph2}\rvert$	Percent of B
800	210	225	10	801	13169	6.1
1000	304	276	55	1385	16318	4.9
1200	342	330	57	1785	17192	3.6
1400	401	384	60	2535	21158	3.2
1600	442	416	63	3244	21512	2.5
1800	486	473	91	4416	45770	4.2
2000	n.a.	518	138	5132	68785	5.1

3.4. Results of heuristic for larger instances

With the aim of reducing the required computation time as much as possible, the heuristic was applied with an intermediate value of $l = 5$ to all problem instances from the set of larger instances described at the end of Section 2.4.2. The results are shown in Table 7. The first table column provides the length of the input strings of the corresponding random instance. The second column indicates the result of applying CPLEX with a computation time limit of 3600 CPU seconds to ILP_{orig}.[6] The remaining five columns contain the results of heuristic. The first one of these columns provides the value of the solution generated by the heuristic, while the second column shows the corresponding computation time. The next two columns provide the size of the sets of common blocks used in phase 1, respectively phase 2, of the heuristic. Finally, the last column gives information about the number of common blocks considered by the heuristic in comparison to the size of the complete set of common blocks (which can be found in Table 5). In particular, summing the common block set sizes from phases 1 and 2 of the heuristic and comparing this number with the size of the complete set of common blocks, the percentage of the common blocks considered by the heuristic can easily be calculated. This percentage is given in the last table column. As before, the best result per table row is indicated by a grey background.

The following observations can be made. First, apart from the smallest problem instance, the heuristic outperforms the application of CPLEX to model ILP_{orig}. Moreover, this is achieved in a fraction of the time needed by CPLEX. Finally, it is reasonable to assume that the success of the heuristic is due to an important reduction of the common blocks that are considered (see last table column). In general, the heuristic only considers between 2.5 percent and 6.1 percent of all common blocks. This is why the computation times are rather low in comparison to CPLEX.

4. Conclusions and future work

In this paper we considered a problem with applications in bioinformatics known as the minimum common string partition problem. First, we introduced an integer linear programming model for this problem. By applying the IBM ILOG CPLEX solver to this model we were able to improve all best-known solutions from the literature for a problem instance set consisting of 45 instances of different sizes. The smallest ones of these problem instances could even be solved to optimality in very short computation time. The second contribution

[6] Remember that the results of applying CPLEX to ILP_{orig} were described in detail in Section 2.4.2.

of the paper concerned a 2-phase heuristic which is strongly based on the developed integer linear programming model. The results have shown that, first, the heuristic outperforms competitor algorithms from the literature, and second, that it is applicable to larger problem instances.

Concerning future work, we aim at studying the incorporation of mathematical programming strategies based on the introduced integer linear programming model into metaheuristic techniques such as GRASP and iterated greedy algorithms. Moreover, we aim at identifying other string-based optimization problems for which a 2-phase strategy such as the one introduced in this paper might work well.

Acknowledgements

C. Blum was supported by project TIN2012-37930 of the Spanish Government. In addition, support is acknowledged from Ikerbasque, Basque Foundation for Science. J. A. Lozano was partially supported by the Saiotek and IT609-13 programs (Basque Government), TIN2013-41272-P (Spanish Ministry of Science and Innovation), COMBIOMED network in computational bio-medicine (Carlos III Health Institute). Our experiments have been executed in the High Performance Computing environment managed by RDlab (http://rdlab.lsi.upc.edu) and we would like to thank them for their support.

References

Blum, C., Lozano, J. A., & Pinacho Davidson, P. (2014). Iterative probabilistic tree search for the minimum common string partition problem. In M. J. Blesa, C. Blum, S. Voss (Eds.), *Proceedings of HM 20104– 9th international workshop on hybrid metaheuristics, Vol. 8457 of Lecture notes in computer science* (p. 154). Verlag, Berlin, Germany: Springer.

Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., et al. (2005). Computing the assignment of orthologous genes via genome rearrangement. In *Proceedings of the Asia Pacific bioinformatics conference 2005* (pp. 363–378).

Chrobak, M., Kolman, P., & Sgall, J. (2004). The greedy algorithm for the minimum common string partition problem. In K. Jansen, S. Khanna, J. D. P. Rolim, & D. Ron (Eds.), *Proceedings of APPROX 2004 – 7th international workshop on approximation algorithms for combinatorial optimization problems, Vol. 3122 of Lecture notes in computer science* (pp. 84–95). Berlin Heidelberg: Springer.

Cormode, G., & Muthukrishnan, S. (2007). The string edit distance matching problem with moves. *ACM Transactions on Algorithms*, *3*(2), 1–19.

Damaschke, P. (2008). Minimum common string partition parameterized. In K. A. Crandall, & J. Lagergren (Eds.) *Proceedings of WABI 2008 – 8th international workshop on algorithms in bioinformatics, Vol. 5251 of Lecture notes in computer science* (pp. 87–98). Berlin Heidelberg: Springer.

Ding, L., & Fu, B. (2013). Algebrization and randomization methods. In P. M. Pardalos, D.-Z. Du, & R. L. Graham (Eds.), *Handbook of combinatorial optimization* (pp. 171–220). New York: Springer.

Ferdous, S. M., & Sohel Rahman, M. (2014). A MAX-MIN ant colony system for minimum common string partition problem, CoRR abs/1401.4539, http://arxiv.org/abs/1401.4539.

Ferdous, S. M., Sohel Rahman, M. (2013). Solving the minimum common string partition problem with the help of ants. In Y. Tan, Y. Shi, & H. Mo (Eds.), *Proceedings of ICSI 2013 – 4th international conference on advances in swarm intelligence, Vol. 7928 of Lecture notes in computer science* (pp. 306–313). Berlin Heidelberg: Springer Berlin Heidelberg.

Fu, B., Jiang, H., Yang, B., & Zhu, B. (2011). Exponential and polynomial time algorithms for the minimum common string partition problem. In W. Wang, X. Zhu, & D.-Z. Du (Eds.), *Proceedings of COCOA 2011 – 5th international conference on combinatorial optimization and applications, Vol. 6831 of Lecture notes in computer science* (pp. 299–310). Berlin Heidelberg: Springer.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman.

Goldstein, A., Kolman, P., Zheng, J. (2005). Minimum common string partition problem: Hardness and approximations. In R. Fleischer, & G. Trippen (Eds.), *Proceedings of ISAAC 2004 – 15th international symposium on algorithms and computation, Vol. 3341 of lecture notes in computer science* (pp. 484–495). Berlin Heidelberg: Springer.

Goldstein, I., Lewenstein, M. (2011). Quick greedy computation for minimum common string partitions. In R. Giancarlo, G. Manzini (Eds.), *Proceedings of CPM 2011 – 22nd annual symposium on combinatorial pattern matching, Vol. 6661 of Lecture notes in computer science* (pp. 273–284). Berlin Heidelberg: Springer.

Gusfield, D. (1997). *Algorithms on strings, trees, and sequences, computer science and computational biology*. Cambridge: Cambridge University Press.

He, D. (2007). A novel greedy algorithm for the minimum common string partition problem. In I. Mandoiu, & A. Zelikovsky (Eds.), *Proceedings of ISBRA 2007 – Third international symposium on bioinformatics research and applications, Vol. 4463 of Lecture notes in computer science* (pp. 441–452). Berlin Heidelberg: Springer.

Hsu, W. J., & Du, M. W. (1984). Computing a longest common subsequence for a set of strings. *BIT Numerical Mathematics*, *24*(1), 45–59. doi:10.1007/BF01934514.

Jiang, H., Zhu, B., Zhu, D., & Zhu, H. (2012). Minimum common string partition revisited. *Journal of Combinatorial Optimization*, *23*(4), 519–527.

Kaplan, H., & Shafrir, N. (2006). The greedy algorithm for edit distance with moves. *Information Processing Letters*, *97*(1), 23–27.

Kolman, P. (2005). Approximating reversal distance for strings with bounded number of duplicates. In J. Jedrzejowicz, & A. Szepietowski (Eds.), *Proceedings of MFCS 2005 – 30th international symposium on mathematical foundations of computer science, Vol. 3618 of Lecture notes in computer science* (pp. 580–590). Berlin Heidelberg: Springer.

Kolman, P., & Waleń, T. (2007). Reversal distance for strings with duplicates: Linear time approximation using hitting set. In T. Erlebach, & C. Kaklamanis (Eds.), *Proceedings of WAOA 2007 – 4th international workshop on approximation and online algorithms, Vol. 4368 of Lecture notes in computer science* (pp. 279–289). Berlin Heidelberg: Springer.

Meneses, C., Oliveira, C., & Pardalos, P. (2005). Optimization techniques for string selection and comparison problems in genomics. *IEEE Engineering in Medicine and Biology Magazine*, *24*(3), 81–87.

Mousavi, S., Babaie, M., & Montazerian, M. (2012). An improved heuristic for the far from most strings problem. *Journal of Heuristics*, *18*, 239–262.

Pappalardo, E., Pardalos, P. M., & Stracquadanio, G. (2013). *Optimization approaches for solving string selection problems*. New York: Springer.

Rajasekaran, S., Hu, Y., Luo, J., Nick, H., Pardalos, P. M., Sahni, S., Shaw, G. (2001). Efficient algorithms for similarity search. *Journal of Combinatorial Optimization*, *5*(1), 125–132.

Rajasekaran, S., Nick, H., Pardalos, P. M., Sahni, S., & Shaw, G. (2001). Efficient algorithms for local alignment search. *Journal of Combinatorial Optimization*, *5*(1), 117–124.

Shapira, D., & Storer, J. A. (2002). Edit distance with move operations. In A. Apostolico, & M. Takeda (Eds.), *Proceedings of CPM 2002 – 13th annual symposium on combinatorial pattern matching, Vol. 2373 of Lecture notes in computer science* (pp. 85–98). Berlin Heidelberg: Springer.

Smith, T., & Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, *147*(1), 195–197.

7

Construct, Merge, Solve & Adapt: A New General Algorithm For Combinatorial Optimization

Computers & Operations Research 68 (2016) 75–88

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Construct, Merge, Solve & Adapt A new general algorithm for combinatorial optimization

Christian Blum [a,b,*], Pedro Pinacho [a,c], Manuel López-Ibáñez [d], José A. Lozano [a]

[a] Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, San Sebastian, Spain
[b] IKERBASQUE, Basque Foundation for Science, Bilbao, Spain
[c] Escuela de Informática, Universidad Santo Tomás, Concepción, Chile
[d] Alliance Manchester Business School, University of Manchester, UK

ARTICLE INFO

Available online 2 November 2015

Keywords:
Metaheuristics
Exact solver
Hybrid algorithms
Minimum common string partition
Minimum covering arborescence

ABSTRACT

This paper describes a general hybrid metaheuristic for combinatorial optimization labelled Construct, Merge, Solve & Adapt. The proposed algorithm is a specific instantiation of a framework known from the literature as Generate-And-Solve, which is based on the following general idea. First, generate a reduced sub-instance of the original problem instance, in a way such that a solution to the sub-instance is also a solution to the original problem instance. Second, apply an exact solver to the reduced sub-instance in order to obtain a (possibly) high quality solution to the original problem instance. And third, make use of the results of the exact solver as feedback for the next algorithm iteration. The minimum common string partition problem and the minimum covering arborescence problem are chosen as test cases in order to demonstrate the application of the proposed algorithm. The obtained results show that the algorithm is competitive with the exact solver for small to medium size problem instances, while it significantly outperforms the exact solver for larger problem instances.

1. Introduction

In this paper we introduce a general algorithm for combinatorial optimization labelled Construct, Merge, Solve & Adapt (CMSA). The proposed algorithm belongs to the class of hybrid metaheuristics [1–4], which are algorithms that combine components of different techniques for optimization. Examples are combinations of metaheuristics with dynamic programming, constraint programming, and branch and bound. In particular, the proposed algorithm is based on the following general idea. Imagine it were possible to identify a substantially reduced sub-instance of a given problem instance such that the sub-instance contains high-quality solutions to the original problem instance. This would allow applying an exact technique—such as, for example, a mathematical programming solver—with little computational effort to the reduced sub-instance in order to obtain a high-quality solution to the original problem instance. This is for the following reason. For many combinatorial optimization problems the field of mathematical programming—and integer linear programming (ILP) in particular—provides powerful tools; for a comprehensive introduction into this area see, for example, [5]. ILP-solvers are in general based on a tree search framework but further include the solution of linear programming relaxations of a given ILP model for the problem at hand (besides primal heuristics) in order to obtain lower and upper bounds. To tighten these bounds, various kinds of additional inequalities are typically dynamically identified and added as cutting planes to the ILP-model, yielding a branch & cut algorithm. Frequently, such ILP approaches are highly effective for small to medium sized instances of hard problems, even though they often do not scale well enough to large instances relevant in practice. Therefore, in those cases in which a problem instance can be sufficiently reduced, a mathematical programming solver might be very efficient in solving the reduced problem instance.

1.1. Related work

The general idea described above is present in several works from the literature. For example, it is the underlying idea of the general algorithm framework known as Generate-And-Solve (GS) [6–9]. In fact, our algorithm can be seen as an instantiation of this framework. The GS framework decomposes the original optimization problem into two conceptually different levels. One of the two levels makes use of a component called *Solver of Reduced Instances* (SRI), in which an exact method is applied to sub-instances of the original problem instance that maintain the conceptual structure of the original instance, that is,

* Corresponding author.
E-mail addresses: christian.blum@ehu.es (C. Blum), ppinacho@santotomas.cl (P. Pinacho), manuel.lopez-ibanez@manchester.ac.uk (M. López-Ibáñez), ja.lozano@ehu.es (J.A. Lozano).

http://dx.doi.org/10.1016/j.cor.2015.10.014
0305-0548/

any solution to the sub-instance is also a solution to the original instance. At the other level, a metaheuristic component deals with the problem of generating sub-instances that contain high quality solutions. In GS, the metaheuristic component is called *Generator of Reduced Instances* (GRI). Feedback is provided from the SRI component to the GRI component, for example, by means of the objective function value of the best solution found in a sub-instance. This feedback serves for guiding the search process of the GRI component.

Even though most existing applications of the GS framework are in the context of cutting, packing and loading problems—see, for example, [7–11]—other successful applications include the ones to configuration problems arising in wireless networks [12–14]. Moreover, it is interesting to note that the applications of GS published to date generate sub-instances in the GRI component using either evolutionary algorithms [10,14] or simulated annealing [11,13]. Finally, note that in [10] the authors introduced a so-called density control operator in order to control the size of the generated sub-instances. This mechanism can be seen as an additional way of providing feedback from the SRI component to the GRI component.

Apart from the GS framework, the idea of solving reduced problem instances to optimality has also been explored in earlier works. In [15,16], for example, the authors tackle the classical traveling salesman problem (TSP) by means of a two-phase approach. The first phase consists in generating a bunch of high-quality TSP solutions using a metaheuristic. These solutions are then merged, resulting in a reduced problem instance, which is then solved to optimality by means of an exact solver. In [17] the authors present the following approach for the prize-collecting Steiner tree problem. First, the given problem instance is reduced in such a way that it still contains the optimal solution to the original problem instance. Then, a memetic algorithm is applied to this reduced problem instance. Finally, a mathematical programming solver is applied to find the best solution to the problem instance obtained by merging all solutions of the first and the last population of the memetic algorithm. Massen et al. [18,19] use an ant colony optimization algorithm to generate a large number of feasible routes for a vehicle routing problem with feasibility constraints, then apply an exact solver to a relaxed set-partitioning problem in order to select a subset of the routes. This subset is used to bias the generation of new routes in the next iteration.

Finally, note that a first, specific, application of the general algorithm proposed in this work has been published in [20] in the context of the minimum weight arborescence problem.

1.2. Contribution of this work

Even though—as outlined above—there is important related work in the literature, the idea of iteratively solving reduced problem instances to optimality has not yet been explored in an exhaustive manner. In this work we introduce a generally applicable algorithm labelled Construct, Merge, Solve & Adapt (CMSA) for tackling combinatorial optimization problems. The algorithm can be seen as a specific instantiation of the GS framework. It is designed to take profit from ILP solvers such as CPLEX even in the context of large problem instances to which these solvers can not be applied directly. In particular, the main feature of the algorithm is the generation of sub-instances of the original problem instance by repeated probabilistic solution constructions, and the application of an ILP solver to the generated sub-instances. Hereby, the way of generating sub-instances by merging the solution components found in probabilistically constructed solutions distinguishes our algorithm from other instantiations of the GS framework from the literature. This feature is actually quite appealing, because our algorithm can easily be applied to any problem for which (1) a constructive heuristic and (2) an exact solver are known.

We consider two test cases for the proposed algorithm: (1) the *minimum common string partition* (MCSP) problem [21], and a *minimum covering arborescence* (MCA) problem, which is an extension of the problem tackled in [20]. For both problems, ILP solvers such as CPLEX are very effective in solving small to medium size problem instances. However, their performance deteriorates (1) in the context of the MCSP problem when the length of the input strings exceeds 600, and (2) in the context of the MCA problem when the number of nodes of the input graph exceeds 1000. We will show that the CMSA algorithm is a new state-of-the-art algorithm for the MCSP problem, especially for benchmark instances for which the application of CPLEX to the original ILP model is not feasible. In the context of the MCA problem we will show that our algorithm is able to match the performance of CPLEX for small and medium size problem instances. Moreover, when large size instances are tackled, the algorithm significantly outperforms a greedy approach.

1.3. Organization of the paper

The remaining part of the paper is organized as follows. The CMSA algorithm is outlined in general terms in Section 2. The application of this algorithm to the minimum common string partition problem is described in Section 3, whereas its application to the minimum covering arborescence problem is outlined in Section 4. An extensive experimental evaluation is provided in Section 5. Finally, in Section 6 we provide conclusions and an outlook to future work.

2. Construct, Merge, Solve & Adapt

In the following we assume that, given a problem instance $\mathcal{I}$ to a generic problem $\mathcal{P}$, set C represents the set of all possible components of which solutions to the problem instance are composed. C is henceforth called the complete set of solution components with respect to $\mathcal{I}$. Note that, given an integer linear (or non-linear) programming model for problem $\mathcal{P}$, a generic way of defining the set of solution components is to say that each combination of a variable with one of its values is a solution component. Moreover, in the context of this work a valid solution S to $\mathcal{I}$ is represented as a subset of the solution components C, that is, $S \subseteq C$. Finally, set $C' \subseteq C$ contains the solution components that belong to a restricted problem instance, that is, a sub-instance of $\mathcal{I}$. For simplicity reasons, C' will henceforth be called a sub-instance. Imagine, for example, the input graph in case of the TSP. The set of all edges can be regarded as the set of all possible solution components C. Moreover, the edges belonging to a tour S—that is, a valid solution—form the set of solution components that are contained in S.

The Construct, Merge, Solve and Adapt (CMSA) algorithm works roughly as follows. At each iteration, the algorithm deals with the incumbent sub-instance C'. Initially this sub-instance is empty. The first step of each iteration consists in *generating* a number of feasible solutions to the original problem instance $\mathcal{I}$ in a probabilistic way. In the second step, the solution components involved in these solutions are added to C' and an exact solver is applied in order to *solve* C' to optimality. The third step consists in *adapting* sub-instance C' by removing some of the solution components guided by an aging mechanism. In other words, the CMSA algorithm is applicable to any problem for which (1) a way of (probabilistically) generating solutions can be found and (2) a strategy for solving the problem to optimality is known.

In the following we describe the CMSA algorithm, which is pseudo-coded in Algorithm 1, in more detail. The main loop of the proposed algorithm is executed while the CPU time limit is not reached. It consists of the following actions. First, the best-so-far solution S_{bsf} is initialized to NULL, and the restricted problem instance (C') to the empty set. Then, at each iteration a number of n_a solutions is probabilistically generated (see function ProbabilisticSolutionGeneration(C) in line 6 of

Algorithm 1). The components of all these solutions are added to set C'. The age of a newly added component c (age[c]) is set to 0. After the construction of n_a solutions, an exact solver is applied to find the best solution S'_{opt} in the restricted problem instance C' (see function ApplyExactSolver(C') in line 12 of Algorithm 1). In case S'_{opt} is better than the current best-so-far solution S_{bsf}, solution S'_{opt} is stored as the new best-so-far solution (line 13). Next, sub-instance C' is adapted, based on solution S'_{opt} and on the age values of the solution components. This is done in function Adapt(C', S'_{opt}, age$_{max}$) in line 14 as follows. First, the age of each solution component in C' is increased by one, and, subsequently, the age of each solution component in $S'_{opt} \subseteq C'$ is re-initialized to zero. Finally, those solution components from C' whose age has reached the maximum component age (age$_{max}$) are deleted from C'. The motivation behind the aging mechanism is that components which never appear in an optimal solution of C' should be removed from C' after a while, because they slow down the exact solver. On the other side, components which appear in optimal solutions seem to be useful and should therefore remain in C'. In general, the average size of set C' depends on the parameter values. For example, the higher the value of age$_{max}$, the higher the average size of C' during a run of the algorithm. In summary, the behavior of the general CMSA algorithm depends on the values of two parameters: the number of solution construction per iteration (n_a) and the maximum allowed age (age$_{max}$) of solution components. Moreover, as long as the mechanism for probabilistically generating solutions has a non-zero probability for generating an optimal solution, the probability to find an optimal solution converges to one with a growing computation time limit. This completes the general description of the algorithm.

Algorithm 1. Construct, Merge, Solve & Adapt (CMSA).

1: **input**: problem instance $\mathcal{I}$, values for parameters n_a and age$_{max}$
2: S_{bsf}:=NULL, C':=$\emptyset$
3: age[c]:=0 for all $c \in C$
4: **while** CPU time limit not reached **do**
5: **for** $i = 1, \ldots, n_a$ **do**
6: S:=ProbabilisticSolutionGeneration(C)
7: **for** all $c \in S$ **and** $c \notin C'$ **do**
8: age[c]:=0
9: C':=$C' \cup \{c\}$
10: **end for**
11: **end for**
12: S'_{opt}:=ApplyExactSolver(C')
13: **if** S'_{opt} is better than S_{bsf} **then** S_{bsf}:=S'_{opt}
14: Adapt(C', S'_{opt}, age$_{max}$)
15: **end while**
16: **output**: S_{bsf}

3. Application to the MCSP problem

The MCSP problem is an *NP*-hard string problem from the bioinformatics field. String problems are very common in bioinformatics. This family of problems includes, among others, string consensus problems such as the far-from most string problem [22,23], the longest common subsequence problem and its variants [24,25], and alignment problems [26]. These problems are often computationally very hard, if not even *NP*-hard [27].

The MCSP problem can technically be described as follows. Given are two input strings s^1 and s^2 of length n over a finite alphabet Σ. The two strings are *related*, which means that each letter appears the same number of times in each of them. This definition implies that s^1 and s^2 have the same length n. A valid solution to the MCSP problem is obtained by partitioning s^1 into a set P_1 of non-overlapping substrings, and s^2 into a set P_2 of non-overlapping substrings, such that $P_1 = P_2$. Moreover, the goal is to find a valid solution such that $|P_1| = |P_2|$ is minimal. Consider the following example. Given are DNA sequences $s^1 =$ **AGACTG** and $s^2 =$ **ACTAGG**. As **A** and **G** appear twice in both input strings, and **C** and **T** appear once, the two strings are related. A trivial valid solution can be obtained by partitioning both strings into substrings of length 1, that is, $P_1 = P_2 = \{\mathbf{A}, \mathbf{A}, \mathbf{C}, \mathbf{T}, \mathbf{G}, \mathbf{G}\}$. The objective function value of this solution is 6. However, the optimal solution, with objective function value 3, is $P_1 = P_2 = \{\mathbf{ACT}, \mathbf{AG}, \mathbf{G}\}$.

The MCSP problem was introduced by Chen et al. [21] due to its relation to genome rearrangement. More specifically, it has applications in biological questions such as: May a given DNA string possibly be obtained by rearrangements of another DNA string? The general problem has been shown to be *NP*-hard even in very restrictive cases [28]. Approximation algorithms are described, for example, in [29]. Recently, Goldstein and Lewenstein [30] proposed a greedy algorithm for the MCSP problem that runs in $O(n)$ time. He [31] introduced a greedy algorithm with the aim of obtaining better average results. To our knowledge, the only metaheuristic approaches that have been proposed in the related literature for the MCSP problem are (1) the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System by Ferdous and Sohel Rahman [32,33] and (2) the probabilistic tree search algorithm by Blum et al. [34]. In these works the proposed algorithm is applied to a range of artificial and real DNA instances from [32]. Finally, the first ILP model for the MCSP problem, together with an ILP-based heuristic, was proposed in [35].

The remainder of this section describes the application of the CMSA algorithm presented in the previous section to the MCSP. For this purpose we define the set C of solution components and the structure of *valid subsets* of C as follows. Henceforth, a *common block* c_i of input strings s^1 and s^2 is denoted as a triple (t_i, k_i^1, k_i^2) where t_i is a string which can be found starting at position $1 \leq k_i^1 \leq n$ in string s^1 and starting at position $1 \leq k_i^2 \leq n$ in string s^2. Moreover, let $C = \{c_1, \ldots, c_m\}$ be the arbitrarily ordered set of all possible common blocks of s^1 and s^2, i.e., C is the set of all solution components. Given the definition of C, a subset S of C is called a valid subset iff the following conditions hold:

1. $\sum_{c_i \in S} |t_i| \leq n$, that is, the sum of the length of the strings corresponding to the common blocks in S is smaller or equal to the length of the input strings.
2. For any two common blocks $c_i, c_j \in S$ it holds that their corresponding strings neither overlap in s^1 nor in s^2.

Given a valid subset $S \subset C$, set $Ext(S) \subset C \setminus S$ denotes the set of common blocks that may be used in order to extend S such that the result is again a valid subset. Note that in case $Ext(S) = \emptyset$ it necessarily holds that $\sum_{c_i \in S} |t_i| = n$. In this case S is a valid subset which corresponds to a complete (valid) solution to the problem.

3.1. Probabilistic solution generation

Next we describe the implementation of function ProbabilisticSolutionGeneration(C) in line 6 of Algorithm 1. The construction of a complete (valid) solution (see Algorithm 2) starts with the empty subset S:=$\emptyset$. At each construction step, a solution component c^* from $Ext(S)$ is chosen and added to S. This is done until $Ext(S) = \emptyset$. The choice of c^* is done as follows. First, a value $\delta \in [0, 1]$ is chosen uniformly at random. In case $\delta \leq d_{rate}$, c^* is chosen such that $|t_{c^*}| \geq |t_c|$ for all $c \in Ext(S)$, that is, one of the common blocks whose substring is of maximal size is chosen. Otherwise, a candidate list L containing the l_{size} longest common blocks from $Ext(S)$ is built, and c^* is chosen from L uniformly at random (ties are broken randomly). In case the number of remaining blocks in $Ext(S)$ is lower than l_{size}, all the blocks are selected. In other words, the

greediness of this procedure depends on the pre-determined values of d_{rate} (determinism rate) and l_{size} (candidate list size). Both are input parameters of the algorithm.

Algorithm 2. Probabilistic solution generation (MCSP problem).

1: **input**: s^1, s^2, d_{rate}, l_{size}
2: $S := \emptyset$
3: **while** $Ext(S) \neq \emptyset$ **do**
4: Choose a random number $\delta \in [0, 1]$
5: **if** $\delta \leq d_{\text{rate}}$ **then**
6: Choose c^* such that $|t_{c^*}| \geq |t_c|$ for all $c \in Ext(S)$
7: $S := S \cup \{c^*\}$
8: **else**
9: Let $L \subseteq Ext(S)$ contain the (at most) l_{size} longest common blocks from $Ext(S)$
10: Choose c^* uniformly at random from L
11: $S := S \cup \{c^*\}$
12: **end if**
13: **end while**
14: **output**: The complete (valid) solution S

3.1.1. Solving reduced sub-instances

The last component of Algorithm 1 which remains to be described is the implementation of function ApplyExactSolver(C') in line 12. In the case of the MCSP problem we make use of the ILP model proposed in [35] and the ILP solver CPLEX for solving it. The model for the complete set C of solution components can be described as follows. First, two binary $m \times n$ matrices M^1 and M^2 are defined. In both matrices, row $1 \leq i \leq m$ corresponds to common block $c_i \in C$. Moreover, a column $1 \leq j \leq n$ corresponds to position j in input string s^1, respectively s^2. In general, the entries of matrix M^1 are set to zero. However, in each row i, the positions that string t_i (of common block c_i) occupies in input string s^1 are set to one. Correspondingly, the entries of matrix M^2 are set to zero, apart from the fact that in each row i the positions occupied by string t_i in input string s^2 are set to one. Henceforth, the position (i,j) of a matrix M is denoted by $M_{i,j}$. Finally, we introduce for each common block $c_i \in C$ a binary variable x_i. With these definitions the MCSP can be expressed in terms of the following ILP model.

$$\min \sum_{i=1}^{m} x_i \tag{1}$$

subject to:

$$\sum_{i=1}^{m} M^1_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{m} M^2_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \ldots, n \tag{3}$$

$$x_i \in \{0, 1\} \quad \text{for } i = 1, \ldots, m$$

The objective function minimizes the number of selected common blocks. Constraints (2) make sure that the strings corresponding to the selected common blocks do not overlap in input string s^1, while constraints (3) make sure that the strings corresponding to the selected common blocks do not overlap in input string s^2. The condition that the length of the strings corresponding to the selected common blocks is equal to n is implicitly obtained from these two constraint sets.

As an example, let us consider the small problem instance that was mentioned at the start of Section 3. The complete set of common blocks (C), as induced by input strings $s^1 =$ **AGACTG** and $s^2 =$ **ACTAGG**, is as follows:

$$C = \left\{ \begin{array}{l} c_1 = (\mathbf{ACT}, 3, 1) \\ c_2 = (\mathbf{AG}, 1, 4) \\ c_3 = (\mathbf{AC}, 3, 1) \\ c_4 = (\mathbf{CT}, 4, 2) \\ c_5 = (\mathbf{A}, 1, 1) \\ c_6 = (\mathbf{A}, 1, 4) \\ c_7 = (\mathbf{A}, 3, 1) \\ c_8 = (\mathbf{A}, 3, 4) \\ c_9 = (\mathbf{C}, 4, 2) \\ c_{10} = (\mathbf{T}, 5, 3) \\ c_{11} = (\mathbf{G}, 2, 5) \\ c_{12} = (\mathbf{G}, 2, 6) \\ c_{13} = (\mathbf{G}, 6, 5) \\ c_{14} = (\mathbf{G}, 6, 6) \end{array} \right\}$$

Given set C, matrices M^1 and M^2 are the following ones:

$$M^1 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad M^2 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The optimal solution to this instance is $S^* = \{c_1, c_2, c_{14}\}$. It can easily be verified that this solution respects constraints (2) and (3) of the ILP model.

Note that this ILP model can also be solved for any subset C' of C. This is achieved by replacing all occurrences of C with C', and by replacing m with $|C'|$. The solution of such an ILP corresponds to a feasible solution to the original problem instance as long as C' contains at least one feasible solution to the original problem instance. Due to the way in which C' is generated (see Section 3.1) this condition is fulfilled.

4. Application to the MCA problem

The MCA problem considered in this section belongs to the family of *minimum weight rooted arborescence* (MWRA) problems [36]. In this type of problem we are given a directed (acyclic) graph with integer weights on the arcs. In some of these problems the weight values might be restricted to be positive, while in other problems positive and negative weights are allowed. Valid solutions to such a problem correspond to subgraphs of the input graph that are arborescences rooted in the pre-defined root node. In this context, a rooted arborescence is a directed, rooted (not necessarily spanning) tree in which all arcs point away from the root node (see [37]). The goal is to find, among all valid solutions, one with minimal weight. Hereby, the weight of an arborescence is defined as the sum of the weights of its arcs. These type of

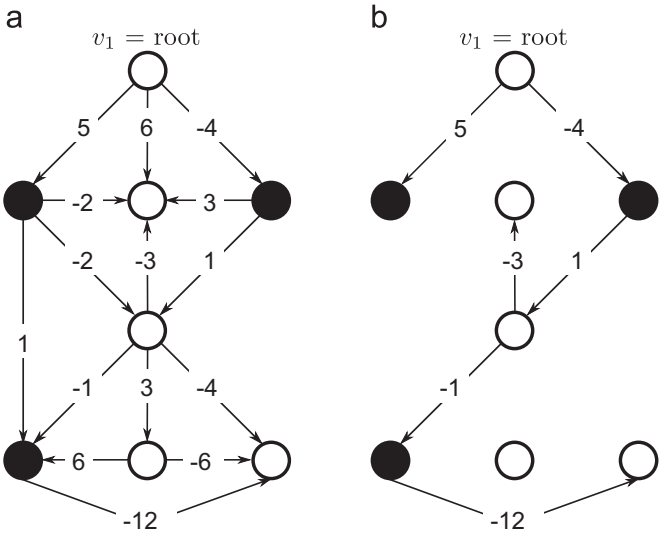
problems have applications, for example, in computer vision and in multistage production planning.

The specific problem tackled in this work—henceforth called *minimum covering arborescence* (MCA) problem—is an extension of the MWRA problem considered in [20] and the minimum covering arborescence problem is described on page 535 of [38]. The MCA problem is formally defined as follows. Given is a directed acyclic graph (DAG) denoted by $G=(V,A)$. Hereby, $V=\{v_1,\ldots,v_n\}$ is the set of n nodes and $A\subseteq\{(i,j)|i\neq j\in V\}$ is a set of m directed arcs. Without loss of generality it is assumed that v_1 is the designated root node. Each arc $a\in A$ has assigned an integer weight $w(a)\in\mathbb{Z}$. Moreover, a pre-defined subset $X\subseteq V$ of the nodes of the input graph must be included in a valid solution. Any arborescence $T=(V(T),A(T))$—where$V(T)\subseteq V$ is the node set of T and $A(T)\subseteq A$ is the arc set of T—rooted in v_1 with $X\subseteq V(T)$ is a valid solution to the problem. Let $\mathcal{A}$ be the set of all such arborescences. The objective function value (that is, the weight) $f(T)$ of an arborescence $T\in\mathcal{A}$ is defined as follows:

$$f(T):=\sum_{a\in A(T)} w(a). \tag{4}$$

The goal of the MCA problem is to find an arborescence $T^*\in\mathcal{A}$ such that the weight of T^* is smaller or equal to the weight of any arborescence in $\mathcal{A}$. In other words, the goal is to minimize objective function $f(\cdot)$. An example of the MCA problem is shown in Fig. 1. As the problem version in which $X=\emptyset$ is already *NP*-hard [20], the more general problem in which $X\neq\emptyset$ is also *NP*-hard. An example for the MCA problem is shown in Fig. 1.

The remainder of this section describes the application of the CMSA algorithm to the MCA problem. For this purpose we define the set of solution components and the structure of valid subsets of the complete set of solution components as follows. First, in the case of the MCA problem, the complete set of solution components corresponds to the set A of arcs of the input graph, that is, $C:=A$. However, for the sake of maintaining the readability of the following description of the algorithm components we continue to use notation A instead of C. Second, a subset S of A is called a valid subset iff $T=(V(S),S)$ is an arborescence of the input graph G rooted in v_1. Hereby, $V(S)\subseteq V$ refers to the subset of nodes that is obtained by joining all the heads and tails of the arcs in S. Given a valid subset $S\subset A$, $Ext(S)\subset A\backslash S$ refers to all arcs that can be added to S such that the result is again a valid subset. More in detail, $Ext(S):=\{a=(v_i,v_j)\in A|v_i\in V(S),v_j\in V\backslash V(S)\}$. In the special case of $S=\emptyset$, $Ext(S):=\mathrm{Out}(v_1)$, where $\mathrm{Out}(v)$, given $v\in V$, denotes the set of outgoing arcs of v, that is, the set of arcs that have v as *tail*. In the same way, $\mathrm{In}(v)$ denotes the set of incoming arcs of v, that is, the set of arcs that have v as *head*. Finally, a valid subset corresponds to a (valid) solution to the problem in case $X\subseteq V(S)$.



Example input graph. Opt. solution of value -14.

Fig. 1. (a) shows an input graph with eight nodes and 15 arcs. The uppermost node is the root node v_1. Moreover, the nodes colored in black form set X, that is, they must be included in any valid solution. (b) shows the optimal solution with value -14.

4.1. Probabilistic solution generation

Next, the implementation of function ProbabilisticSolutionGeneration(C) in line 6 of Algorithm 1 is described. The pseudo-code of this procedure is outlined in Algorithm 3. Starting from the root node v_1, at each step an arc—that is, a solution component—is chosen from set $\hat{A}$ (see lines 3 and 7 of Algorithm 3). For the choice of the first solution component in line 3, $\hat{A}$ is defined as the set of outgoing arcs of the root node v_1. For all further construction steps, $\hat{A}$ is defined as $Ext(S)$. However, instead of considering the whole set of arcs connecting one of the nodes of the current arborescence with one of the remaining nodes, function Reduce($\hat{A}$) is applied before choosing one of the arcs from $\hat{A}$ (see line 8). This function chooses from each set $\{(v_j,v_i)|v_j\in V(S)\}\subseteq\hat{A}$, for all $v_i\in V\backslash V(S)$, the arc with minimal weight. The chosen arc remains in $\hat{A}$, while the other ones are deleted. In other words, if a node $v_i\in V\backslash V(S)$ may be connected via several arcs with the current arborescence $T=(V(S),S)$, only the arc with minimal weight is considered. Finally, the process of constructing a solution finishes when $\hat{A}=\emptyset$, that is, when all nodes are already included in the constructed arborescence. In principle, the construction process could already be stopped once all nodes from X are included in the constructed arborescence. However, experimental tests have shown that generating spanning arborescences leads, overall, to better results.

Algorithm 3. Probabilistic solution generation (MCA problem).

1: **input**: a DAG $G=(V,A)$ with root node v_1, $d_{\min}$, $d_{\max}$
2: $S:=\emptyset$
3: $\hat{A}:=\mathrm{Out}(v_1)$
4: **while** $\hat{A}\neq\emptyset$ **do**
5: $a^*:=\mathrm{Choose}(\hat{A},d_{\min},d_{\max})$
6: $S:=S\cup\{a^*\}$
7: $\hat{A}:=Ext(S)$
8: $\hat{A}:=\mathrm{Reduce}(\hat{A})$
9: **end while**
10: **output:** valid subset S which induces arborescence $T=(V(S),S)$

The choice of an arc from $\hat{A}$ is done in function Choose($\hat{A}$, $d_{\min}$, $d_{\max}$)—see line 5 of the pseudo-code—based on heuristic information. The heuristic information $\eta(a)$ of an arc $a\in\hat{A}$—which will be used below in Eq. (7)—is computed as follows. First, let

$$w_{\max}:=\max\{w(a)|a\in A\}. \tag{5}$$

Based on this maximal weight of all arcs in G, the heuristic information is defined as follows:

$$\eta(a):=w_{\max}+1-w(a) \tag{6}$$

In this way, the heuristic information of all arcs is a positive integer number. Moreover, the arc with minimal weight has the highest heuristic value.

Given the current valid subset S—corresponding to arborescence $T=(V(S),S)$—and the non-empty set of arcs $\hat{A}$ that may be used for extending S, the probability for choosing arc $a\in\hat{A}$ is

defined as follows:

$$\mathbf{p}(a|S) := \frac{\eta(a)}{\sum_{a' \in \hat{A}} \eta(a')} \quad (7)$$

At the start of each arborescence construction, a so-called *determinism rate* δ is chosen uniformly at random from $[d_{\min}, d_{\max}]$, where $0 \leq d_{\min} \leq d_{\max} \leq 1$. The chosen value for $\delta$is then used during the arborescence construction as follows. At each construction step, first, a value $r \in [0,1]$ is chosen uniformly at random. Second, in case $r \leq \delta$, the arc $a^* \in \hat{A}$ with the maximum probability is deterministically chosen, that is: $a^* := \arg\max_{a \in \hat{A}} \{\mathbf{p}(a|S)\}$. Otherwise, that is, when $r > \delta$, arc $a^* \in \hat{A}$ is chosen probabilistically according to the probability values.

4.1.1. Solving reduced sub-instances

The last component of Algorithm 1 which remains to be described is the implementation of function ApplyExactSolver(C') in line 12. In the case of the MCA problem we make use of the following ILP model, which is a slight modification of models that can be found in [20,39,40] for related problems. The model works on an augmented graph $G^+ = (V^+ := V \cup \{v_0\}, A^+ := A \cup \{(v_0, v_1)\})$, where v_0 is an additional dummy node and (v_0, v_1) is a dummy arc connecting v_0 with the root node v_1. The weight $w(v_0, v_1)$ of arc (v_0, v_1) is zero. Henceforth, let $Pred(a) \in A^+$ denote for each $a \in A$ the set of predecessor arcs, that is, the set of arcs pointing to the tail of arc a. The ILP model works on a set of binary variables which contains for each arc $a \in A^+$ a binary variable $x_a \in \{0,1\}$. The ILP itself can then be stated as follows.

$$\min \sum_{a \in A} w(a) \cdot x_a \quad (8)$$

subject to:

$$\sum_{a \in In(v_i)} x_a \leq 1 \quad \text{for } v_i \in V \setminus (X \cup \{v_1\}) \quad (9)$$

$$\sum_{a \in In(v_i)} x_a = 1 \quad \text{for } v_i \in X \quad (10)$$

$$x_a - \sum_{a' \in Pred(a)} x_{a'} \leq 0 \quad \text{for } a \in A \quad (11)$$

$$x_{(v_0,v_1)} = 1 \quad (12)$$

$$x_a \in \{0,1\} \quad \text{for } a \in A^+$$

Hereby, constraints (9) ensure that for each node $v_i \in V \setminus (X \cup \{v_1\})$ (that is, all nodes of the original graph without the nodes from X and the root node) at most one incoming arc is chosen to form part of the arborescence. For all nodes in X, constraints (10) make sure that exactly one incoming arc is chosen. Constraints (11) ensure that if an arc a from the original graph is chosen for the arborescence, then also one predecessor arc of the tail of a must be chosen for the arborescence. Finally, constraint (12) forces the arborescence to start in dummy arc (v_0, v_1), which means that v_1 is forced to be the root node of the arborescence in the original graph G.

This ILP model can also be solved for any subgraph G' of G which is, itself, a DAG with root node v_1. Note that set C' (see Algorithm 1) in case of the MCA problem induces such a subgraph. The optimal solution to such a *reduced* ILP corresponds to a feasible solution to the original problem instance as long as G' contains at least one feasible solution to the original problem instance. Due to the way in which C' is generated (see Section 4.1) this condition is fulfilled.

5. Experimental evaluation

The proposed applications of CMSA to the MCSP problem and the MCA problem were implemented in ANSI C++ using GCC 4.7.3 for compiling the software. Moreover, both the complete ILP models and the reduced ILP models within CMSA were solved with IBM ILOG CPLEX 12.1. The experimental evaluation was conducted on a cluster of 32 PCs with Intel(R) Xeon(R) X5660 CPUs with 2 cores at 2.8 GHz and 48 GB of RAM.

5.1. Experiments concerning the MCSP problem

The following algorithms were considered for the comparison: Greedy, the greedy approach from [31]; TreSea, the probabilistic tree search approach from [34]; $I_{LP_{compl}}$, the application of CPLEX to the complete ILP for each considered problem instance; HeurIlp, the application of an ILP-based heuristic from [35];[1] and Cmsa, our proposed CMSA approach. Moreover, in the context of the existing benchmark instances from the literature a comparison to the ant colony optimization approach from [32,33] (labelle Aco) is also included.

5.1.1. Benchmark instances

Both existing as well as new benchmark instances were used for the experimental evaluation. As a first benchmark set we chose the one that was introduced by Ferdous and Sohel Rahman [32] for the experimental evaluation of their ant colony optimization approach. This set contains, in total, 30 artificial instances and 15 real-life instances consisting of DNA sequences, that is, the size of the alphabet is four. Remember, in this context, that each problem instance consists of two related input strings. Moreover, the benchmark set is divided into four subsets of instances. The first subset (henceforth labelled Group1) consists of 10 artificial instances in which the input strings are maximally of length 200. The second set (Group2) consists of 10 artificial instances with input string lengths between 201 and 400. In the third set (Group3) the input strings of the 10 artificial instances have lengths between 401 and 600. Finally, the fourth set (Real) consists of 15 real-life instances of various lengths between 200 and 600.

The second benchmark set that we used is new. It consists of 20 randomly generated instances for each combination of $n \in \{200, 400, \ldots, 1800, 2000\}$, where n is the length of the input strings, and alphabet size $|\Sigma| \in \{4, 12, 20\}$. 10 of these instances are generated with an equal probability for each letter of the alphabet. More specifically, the probability for each letter $l \in \Sigma$ to appear at a certain position of the input strings is $\frac{1}{|\Sigma|}$. The resulting set of 300 benchmark instances of this type are labelled Linear. The other 10 instances per combination of n and $|\Sigma|$ are generated with a probability for each letter $l \in \Sigma$ to appear at a certain position of the input strings of $l / \sum_{i=1}^{|\Sigma|} i$. The resulting set of 300 benchmark instances of this second type are labelled Skewed.

5.1.2. Tuning of Cmsa and TreSea

Cmsa has several parameters for which well-working values must be found: (n_a) the number of solution constructions per iteration, (age_{max}) the maximum allowed age of solution components, (d_{rate}) the determinism rate, (l_{size}) the candidate list size, and (t_{max}) the maximum time in seconds allowed for CPLEX per application to a sub-instance. The last parameter is necessary, because even when applied to reduced problem instances, CPLEX might still need too much computation time for solving such sub-

[1] HeurIlp has a parameter l that needs to be given a value. In our experiments we chose $l = \min\{5, l_{max}\}$, where l_{max} denotes the length of the longest common block. This is following a suggestion of the authors of [35].

instances to optimality. In any case, CPLEX always returns the best feasible solution found within the given computation time.

We decided to make use of the automatic configuration tool irace [41] for the tuning of the five parameters. In fact, irace was applied to tune CMSA separately for each instance size from $\{200, 400, \ldots, 1800, 2000\}$. For each of these 10 different instance sizes we generated 12 *training instances* for tuning: two instances of type Linear and two instances of type Skewed for each alphabet size from $\{4, 12, 20\}$. The tuning process for each instance size was given a budget of 1000 runs of Cmsa, where each run was given a computation time limit of 3600 CPU seconds. Finally, the following parameter value ranges were chosen concerning the five parameters of Cmsa:

- $n_a \in \{10, 30, 50\}$
- $\text{age}_{\max} \in \{1, 5, 10, inf\}$, where *inf* means that solution components are never removed from C'.
- $d_{\text{rate}} \in \{0.0, 0.5, 0.9\}$, where a value of 0.0 means that the selection of solution component c^* (see line 6 of Algorithm 2) is always done randomly from the candidate list, while a value of 0.9 means that solution constructions are nearly deterministic.
- $l_{\text{size}} \in \{3, 5, 10\}$
- $t_{\max} \in \{60, 120, 240, 480\}$ (in seconds)

The 10 applications of iraceproduced the 10 configurations of Cmsa shown in Table 1a. The following trends can be observed. First of all, with growing instance size, more time ($t_{\max}$) should be given to individual applications of CPLEX to sub-instances of the original problem instance. Second, irrespective of the instance size, candidate list sizes (l_{size}) smaller than five seem to be too restrictive. Third, also irrespective of the instance size, less than 30 solution constructions per iteration (n_a) seem to be insufficient. Presumably, when only few solution constructions per iteration are performed, the resulting change in the corresponding sub-instances is not large enough and, therefore, some applications of CPLEX result in wasted computation time. Finally, considering the obtained values of d_{rate} for instance sizes from 200 to 1600, the trend is that with growing instance size the degree of greediness in the solution construction should grow. However, the settings of d_{rate} for $n \in \{1800, 2000\}$ is not in accordance with this observation.

Table 1
Parameter settings produced by irace for the 10 different instance sizes.
(a) Tuning results for CMSA

n	n_a	$\text{age}_{\max}$	d_{rate}	l_{size}	$t_{\max}$
200	50	*inf*	0.0	10	60
400	50	10	0.0	10	60
600	50	10	0.5	10	60
800	50	10	0.5	10	240
1000	50	10	0.9	10	480
1200	50	10	0.9	10	480
1400	50	*inf*	0.9	5	480
1600	50	5	0.9	10	480
1800	30	10	0.5	5	480
2000	50	10	0.0	10	480

(b) Tuning results for TreSea

n	d_{rate}	l_{size}
200	0.9	5
400	0.9	3
600	0.9	10
800	0.5	5
1000	0.5	5
1200	0.5	3
1400	0.5	5
1600	0.0	5
1800	0.0	10
2000	0.0	10

In addition to tuning experiments for Cmsa, we also performed tuning experiments for TreSea. In fact, TreSea constructs solutions in the same way in which they are constructed in Cmsa. The parameters involved in TreSea are, therefore, d_{rate} and l_{size}. For the tuning of TRESEA we used the same training instances, the same budget of 1000 runs, and the same parameter value ranges as for the tuning of CMSA. The obtained parameter values per instance size are displayed in Table 1b.

5.1.3. Results

In the following we present the experimental results for the two benchmark data sets described in Section 5.1.1, which are different from the data sets used for tuning the algorithms.

The first benchmark set, as outlined above, consists of four subsets of instances labelled GROUP1, GROUP2, GROUP3, and REAL. The results for these groups of instances are shown in the four Table 2a–d. The structure of these four tables is as follows. The first column provides the instance identifiers. The second column contains the results of GREEDY. The third column provides the value of the best solution found in four independent runs per problem instance (with a CPU time limit of 7200 s per run) by ACO; results are taken from [32,33]. The fourth column contains the value of the best solution found in 10 independent runs per problem instance (with a CPU limit of 3600 s per run) by TRESEA. The next three table columns are dedicated to the presentation of the results provided by solving the complete ILP model $I_{\text{LP}_{\text{compl}}}$. The first one of these columns provides the value of the best solution found within 3600 CPU seconds. The second column provides the computation time (in seconds). In case of having solved the corresponding problem to optimality, this column only displays one value indicating the time needed by CPLEX to solve the problem. Otherwise, this column provides two values in the form X/Y, where X corresponds to the time at which CPLEX was able to find the first valid solution, and Y corresponds to the time at which CPLEX found the best solution within 3600 CPU seconds. Finally, the third one of the columns dedicated to $I_{\text{LP}_{\text{compl}}}$ shows the optimality gap, which refers to the gap between the value of the best valid solution and the current lower bound at the time of stopping a run. The next two columns display the results of the ILP-based, deterministic heuristic HEURILP. The first column contains the results, and the second column the computation time. Finally, the last three columns of each table are dedicated to the presentation of the results obtained by CMSA. The first column provides the value of the best solutions found by CMSA within 3600 CPU seconds. The second column provides the average (mean) results over 10 independent runs per problem instance. The last column indicates the average time needed by CMSA to find the best solution of a run. The best result for each problem instance is marked by a grey background and the last row of each table provides averages over the whole table.

Analyzing the results it can be observed that the results of CMSA are very similar to the ones of applying CPLEX to $I_{\text{LP}_{\text{compl}}}$. In fact, the application of the non-parametric Wilcoxon test for all four instance subsets did not reveal differences of statistical significance between both techniques (for an α-value of 0.05). In comparison to the other techniques (GREEDY, ACO, TRESEA and HEURILP) both CMSA and the application of CPLEX to $I_{\text{LP}_{\text{compl}}}$ significantly outperform the competitors.

As described in Section 5.1.1, the second benchmark set which was specifically generated for this paper, consists of 300 instances of type LINEAR and another 300 instances of type SKEWED. The results for instances of type LINEAR are presented in the three Table 3a–c and the ones of type SKEWED are presented in the three Table 4a–c, in terms of one table per alphabet size. In contrast to the first

Table 2
Results for the instances of the first benchmark set (consisting of Group1, Group2, Group3 and Real).

id	Greedy	Aco	TreSea	ILP_{compl}			HeurIlp		Cmsa		
	Value	**Value**	**Value**	**Value**	**Time**	**Gap**	**Value**	**Time**	**Best**	**Mean**	**Time**
(a) Results for Group1.											
1	46	42	42	41	1	0.0%	42	< 1	41	41.0	2
2	54	51	48	47	3	0.0%	48	< 1	47	47.0	6
3	60	55	55	52	30	0.0%	54	< 1	52	52.0	298
4	46	43	43	41	2	0.0%	43	< 1	41	41.0	23
5	44	43	41	40	1	0.0%	43	< 1	40	40.0	2
6	48	42	41	40	3	0.0%	41	< 1	40	40.0	2
7	64	60	59	55	38	0.0%	59	< 1	56	56.0	29
8	47	47	45	43	3	0.0%	44	< 1	43	43.0	1027
9	42	45	43	42	2	0.0%	48	< 1	42	42.0	28
10	63	59	58	54	50	0.0%	58	< 1	54	54.0	36
Avg.	51.4	48.7	47.5	45.5	13.3	0.0%	48.0	< 1	45.6	45.6	145
(b) Results for Group2											
1	118	113	111	98	66/1969	2.9%	108	3	101	101.2	2045
2	121	118	114	106	129/1032	7.5%	111	2	104	104.6	1677
3	114	111	107	97	55/1216	2.7%	105	3	97	97.1	1883
4	116	115	110	102	63/949	4.9%	111	3	102	102.5	1187
5	132	132	127	116	146/3299	6.7%	125	4	117	117.8	1581
6	107	105	102	93	56/1419	5.6%	101	< 1	94	95.4	1587
7	106	98	95	88	41/2776	6.0%	96	2	88	89.0	2103
8	122	118	114	104	101/2980	5.1%	116	2	103	105.2	1858
9	123	119	113	104	81/1630	5.2%	112	2	104	104.9	2010
10	102	101	97	89	32/1458	3.6%	94	3	89	89.8	1550
Avg.	116.1	113.0	109.0	99.7	77/1873	5.0%	107.9	2	99.9	100.8	1748
(c) Results for Group3											
1	181	177	171	155	733/1398	7.5%	173	5	157	157.9	1842
2	173	175	168	155	553/869	7.7%	165	9	156	157.5	1702
3	195	187	185	166	746/2183	8.5%	180	6	166	167.3	1805
4	191	184	179	159	731/1200	6.9%	171	15	160	161.8	2057
5	174	171	162	150	485/886	9.7%	164	4	149	151.1	1224
6	169	160	162	147	399/764	9.1%	155	4	148	149.3	2027
7	171	167	159	149	524/990	9.8%	160	4	146	147.8	2265
8	185	175	170	151	492/3584	6.7%	166	7	153	154.2	1790
9	174	172	169	158	571/1186	10.9%	169	5	154	155.3	2468
10	171	167	160	148	547/1446	9.1%	160	4	148	149.0	1768
Avg.	178.4	173.5	168.5	153.8	578/1451	8.6%	166.3	6	153.7	155.1	1895
(d) Results for Real											
1	93	87	86	78	972	0.0%	85	< 1	78	78.9	1192
2	160	155	153	139	432/752	9.2%	150	3	138	140.0	1960
3	119	116	113	104	125/3580	5.6%	112	2	103	104.7	1126
4	171	164	156	144	577/1730	6.5%	158	6	143	143.7	2037
5	172	171	166	150	778/2509	7.9%	161	5	151	152.9	1557
6	153	145	143	128	257/3578	6.5%	139	3	126	127.6	1469
7	135	140	131	121	359/2187	6.9%	132	2	122	122.7	1657
8	133	130	128	116	275/3365	6.8%	123	3	118	118.4	1576
9	149	146	142	131	399/613	8.8%	139	2	130	130.7	1790
10	151	148	143	131	311/1771	7.2%	144	3	131	131.7	1500
11	124	124	120	110	205/3711	4.8%	122	2	111	111.9	1658
12	143	137	138	126	299/793	9.8%	136	2	126	127.5	1903
13	180	180	172	156	784/1130	7.1%	171	5	158	158.6	2066
14	150	147	146	134	370/2456	9.7%	147	6	133	134.0	1789
15	157	160	152	139	560/1762	7.7%	148	3	141	141.7	1424
Avg.	146	143.3	139.3	127.1	409/2131	7.0%	137.8	3	127.3	128.3	1647

benchmark set, for which the probabilistic algorithms such as TreSea and Cmsa were applied for 10 independent runs, results for instances of type Linear and Skewed are presented in these tables in terms of averages over 10 random instances of the same characteristics. Each algorithm included in the comparison was applied exactly once to each problem instance. Note that in addition to different alphabet sizes ($|\Sigma| \in \{4, 12, 20\}$) this second benchmark set also contains much larger instances than the first benchmark set (input strings with a length of up to $n = 2000$).

The analysis of the results permits to draw the following conclusions:

- Surprisingly, hardly any differences can be observed in the relative performance of the algorithms for instances of type Linear and instances of type Skewed. Therefore, all the following statements hold both for Linear and Skewed instances.
- Concerning the application of CPLEX to ILP_{compl}, the alphabet size has a strong influence on the problem difficulty. A value of "n/a" denotes that CPLEX was not able to find a feasible solution

Table 3
Results for the instances of set Linear.

n	Greedy	TreSea	$I_{LP_{compl}}$			HeurILP		Cmsa	
	Mean	**Mean**	**Mean**	**Time**	**Gap**	**Mean**	**Time**	**Mean**	**Time**
(a) Results for instances with $\Sigma = 4$									
200	75.0	68.7	63.5	4/104	0.0%	69.0	< 1	63.7	608
400	133.4	126.1	115.7	108/2081	6.8%	124.3	3	116.4	1381
600	183.7	177.5	162.2	513/1789	9.4%	174.1	8	162.9	1918
800	241.1	232.7	246.8	1671/1671	23.8%	229.1	17	212.4	2434
1000	287.0	280.4	n/a	n/a	n/a	277.2	1095	256.9	2623
1200	333.8	330.4	n/a	n/a	n/a	324.8	1803	303.3	2369
1400	385.5	378.9	n/a	n/a	n/a	373.1	1807	351.0	2452
1600	432.3	427.1	n/a	n/a	n/a	416.7	1811	400.6	1973
1800	477.4	474.2	n/a	n/a	n/a	464.4	1813	445.4	2194
2000	521.6	520.7	n/a	n/a	n/a	512.7	1820	494.0	1744
Avg.	307.1	301.7	n/a	n/a	n/a	296.5	1018	280.7	1969
(b) Results for instances with $\Sigma = 12$									
200	127.3	122.1	119.2	1/1	0.0	123.0	< 1	119.2	22
400	228.9	223.5	208.9	7/51	0.0	215.7	6	209.4	892
600	322.2	318.7	291	47/1277	0.9	296.2	691	293.8	1433
800	411.4	408.1	368.7	147/2405	1.6	373.9	1546	373.2	1484
1000	499.2	494.9	453.4	395/2084	3.8	452.0	1802	449.9	2651
1200	586.0	585.6	536.6	784/3188	4.7	542.4	1803	531.0	2318
1400	666.0	664.6	684.1	1667/1667	15.8	653.3	1864	606.9	2467
1600	754.4	754.6	773.5	2648/2648	16.0	749.7	2045	694.8	2392
1800	827.3	833.0	n/a	n/a	n/a	850.7	3301	773.6	1484
2000	913.5	916.2	n/a	n/a	n/a	939.6	5052	849.6	2967
Avg.	533.6	532.1	n/a	n/a	n/a	519.7	1811	490.1	1811
(c) Results for instances with $\Sigma = 20$									
200	149.2	146.6	146.2	1/1	0.0%	146.4	< 1	146.2	2
400	274.5	268.8	261.5	2/2	0.0%	263.8	< 1	261.9	80
600	389.2	383.5	362.3	10/15	0.0%	369.3	4	366.6	364
800	495.8	492.3	456.1	43/700	0.0%	464.7	121	463.1	804
1000	600.6	597.5	547.1	125/1737	0.6%	562.5	205	555.0	529
1200	706.1	707.8	642.2	296/2732	1.3%	658.8	415	648.5	1372
1400	801.1	804.0	737.9	559/2314	3.1%	745.7	812	737.7	2334
1600	899.8	903.1	861.3	966/2885	6.6%	872.6	1015	825.7	2251
1800	996.8	1000.1	1012.9	1559/1845	12.6%	994.4	1336	917.6	2437
2000	1097.8	1102.6	1136.0	2349/2349	14.4%	1120.7	1773	1024.9	2924
Avg.	641.1	640.6	616.35	591/1458	3.9%	619.9	568	594.4	1310

Table 4
Results for the instances of set Skewed.

n	Greedy	TreSea	$I_{LP_{compl}}$			HeurILP		Cmsa	
	Mean	**Mean**	**Mean**	**Time**	**Gap**	**Mean**	**Time**	**Mean**	**Time**
(a) Results for instances with $\Sigma = 4$									
200	68.7	62.8	57.4	10/217	0.0%	64.6	< 1	57.5	903
400	120.3	115.0	105.3	168/1330	7.6%	116.5	3	105.1	1314
600	170.6	163.8	149.7	938/2193	10.1%	165.2	38	150.4	1500
800	219.8	213.3	224	2600/2600	22.9%	211.7	1334	196.5	2303
1000	268.6	261.7	n/a	n/a	n/a	260.1	1798	240.2	2692
1200	313.8	309.0	n/a	n/a	n/a	302.1	1807	285	2785
1400	358.7	352.2	n/a	n/a	n/a	346.0	1801	327.6	2888
1600	400.9	397.9	n/a	n/a	n/a	394.4	1807	376.0	2171
1800	440.6	442.1	n/a	n/a	n/a	431.7	1808	417.7	2162
2000	485.0	481.2	n/a	n/a	n/a	468.9	1814	470.2	1222
Avg.	284.7	279.9	n/a	n/a	n/a	276.1	1221	262.6	1994
(b) Results for instances with $\Sigma = 12$									
200	117.9	112.7	108.5	1/1	0.0%	112.7	< 1	108.6	12
400	216.1	208.5	193.4	10/136	0.0%	197.6	32	194.3	1002
600	304.8	301.7	274.5	71/1081	1.2%	277.9	650	277.2	1711
800	389.3	385.4	347.0	248/2725	2.3%	348.8	1533	351.0	2177
1000	471.6	468.9	429.4	650/2582	4.9%	428.7	1805	424.4	2648
1200	551.1	549.9	559.4	1351/1804	14.9%	535.0	1686	500.1	2597
1400	625.7	626.3	645.1	2693/2693	16.7%	638.4	1879	570.0	2962
1600	705.6	706.4	n/a	n/a	n/a	715.1	2981	643.8	2434
1800	788.4	788.9	n/a	n/a	n/a	810.1	4689	723.3	2329
2000	857.8	858.0	n/a	n/a	n/a	879.9	6072	797.3	2805
Avg.	502.8	500.7	n/a	n/a	n/a	494.4	2133	459.0	2068
(c) Results for instances with $\Sigma = 20$									
200	140.4	135.9	134.7	1/1	0.0%	136.5	< 1	134.7	8
400	255.5	251.3	240.3	3/4	0.0%	246.1	2	240.6	1080
600	366.8	361.2	336.1	19/101	0.0%	344.6	15	341.1	764
800	466.3	462.7	424.4	80/1119	0.2%	429.9	442	429.8	753
1000	567.6	566.6	514.7	202/2253	0.9%	525.0	1130	520.9	1121
1200	661.8	662.4	604.2	469/2064	2.1%	608.2	1633	605.7	1869
1400	762.3	760.7	694.4	719/2511	2.8%	696.1	1837	693.2	1743
1600	851.2	855.2	863.3	1378/1828	12.3%	838.9	1804	780.4	2681
1800	948.7	948.8	969.8	1774/1976	13.9%	964.7	1713	870.2	2815
2000	1034.3	1037.7	1061.6	2589/2589	14.4%	1066.6	2547	967.1	2978
Avg.	605.5	604.3	584.4	723/1844	4.7%	585.7	1112	558.4	1581

within 3600 CPU seconds. For instances with $|\Sigma| = 4$, CPLEX is only able to provide feasible solutions for input strings of length up to 800, both in the context of instances Linear and Skewed. When $|\Sigma| = 12$, CPLEX can provide feasible solutions for input strings of length up to 1600 (Linear), respectively 1400 (Skewed). However, starting from $n = 1000$ CPLEX is not competitive with Cmsa anymore. Finally, even though CPLEX can provide feasible solutions for all instance sizes concerning the instances with $|\Sigma| = 20$, starting from $n = 1400$ the results are inferior to the ones of Cmsa.

- For instances smaller than those for which Cmsa outperforms CPLEX, the differences between the results of Cmsa and the ones of applying CPLEX to $I_{LP_{compl}}$ are, again, very small.

In summary, we can state that Cmsa is competitive with the application of CPLEX to the original ILP model when the size of the input instances is rather small. Moreover, the larger the size of the input instances, and the smaller the alphabet size, the greater is the advantage of Cmsa over the other algorithms. The validity of these statements can be conveniently observed in the graphics of Fig. 2.

Finally, we also provide information about the average sizes of the sub-instances tackled within Cmsa, in comparison to the sizes of the original problem instances. In particular, the average sizes of the sub-instances are shown in Fig. 3 in percent of the original problem instance sizes. For example, in the case $|\Sigma| = 4$, Linear, and input strings of length $n = 200$, the considered average size of the tackled sub-instances within Cmsa is approximately 58% of the size of the original instances. It can be observed that this percentage is getting smaller and smaller with growing size of the original instances. This is why CPLEX can either solve the sub-instances to optimality or provide nearly optimal solutions in little computation time, even in the context of large original problem instances.

5.2. Experiments concerning the MCA problem

The following algorithms were considered for the comparison:

1. SPGreedy: this is a cut-down version of Cmsa in which solutions are probabilistically generated, while other algorithmic components such as the ageing mechanism and the application of the ILP solver to reduced instances are not used. Remember that in the Cmsa implementation for the MCA problem, solutions are generated that are not extensible, that is, the generated solutions cover all reachable nodes. This implies that, during the construction process, once all nodes from X are included in the current arborescence, other complete (and valid) solutions are encountered. For each solution construction, PGreedy returns the best solution encountered in the process of generating a non-extensible solution.
2. $I_{LP_{compl}}$: the application of CPLEX to the complete ILP for each considered problem instance.
3. Cmsa: the proposed CMSA approach.

5.2.1. Benchmark instances

A diverse set of benchmark instances was generated in the following way. Each benchmark instance is characterized by three different parameters. First, the size n (number of nodes) of each generated DAG is taken from $\{500, 1000, 5000\}$. In the process of randomly generating a DAG $G=(V,A)$ with n nodes, the arc probability $\mathbf{p}_{\text{arc}}$ is used to determine for each possible arc a pointing from a node $v_i \in V$ to another node $v_j \in V \setminus \{v_i\}$—where $i < j$—if a is added to A or not. Three different arc probabilities were considered: $\mathbf{p}_{\text{arc}} \in \{0.1, 0.3, 0.5\}$. Finally, the third parameter determines the size of set X. For this purpose we used a parameter *perc*, which refers to the percentage of the number of nodes of the respective DAG, that is, in case $perc = 20\%$, for example, set X contains 20% of the nodes of the respective DAG. Note that the nodes in X are randomly selected from the set of reachable nodes. Values for *perc* were chosen from $\{1\%, 10\%, 20\%\}$.

Finally, note that the arc weights for all problem instances were chosen in the following way. In 99% of all arcs, a weight is chosen uniformly at random from $\{0, \ldots, 1000\}$. In the remaining cases, a negative arc weight is chosen from $\{-1000, \ldots, -1\}$. This was

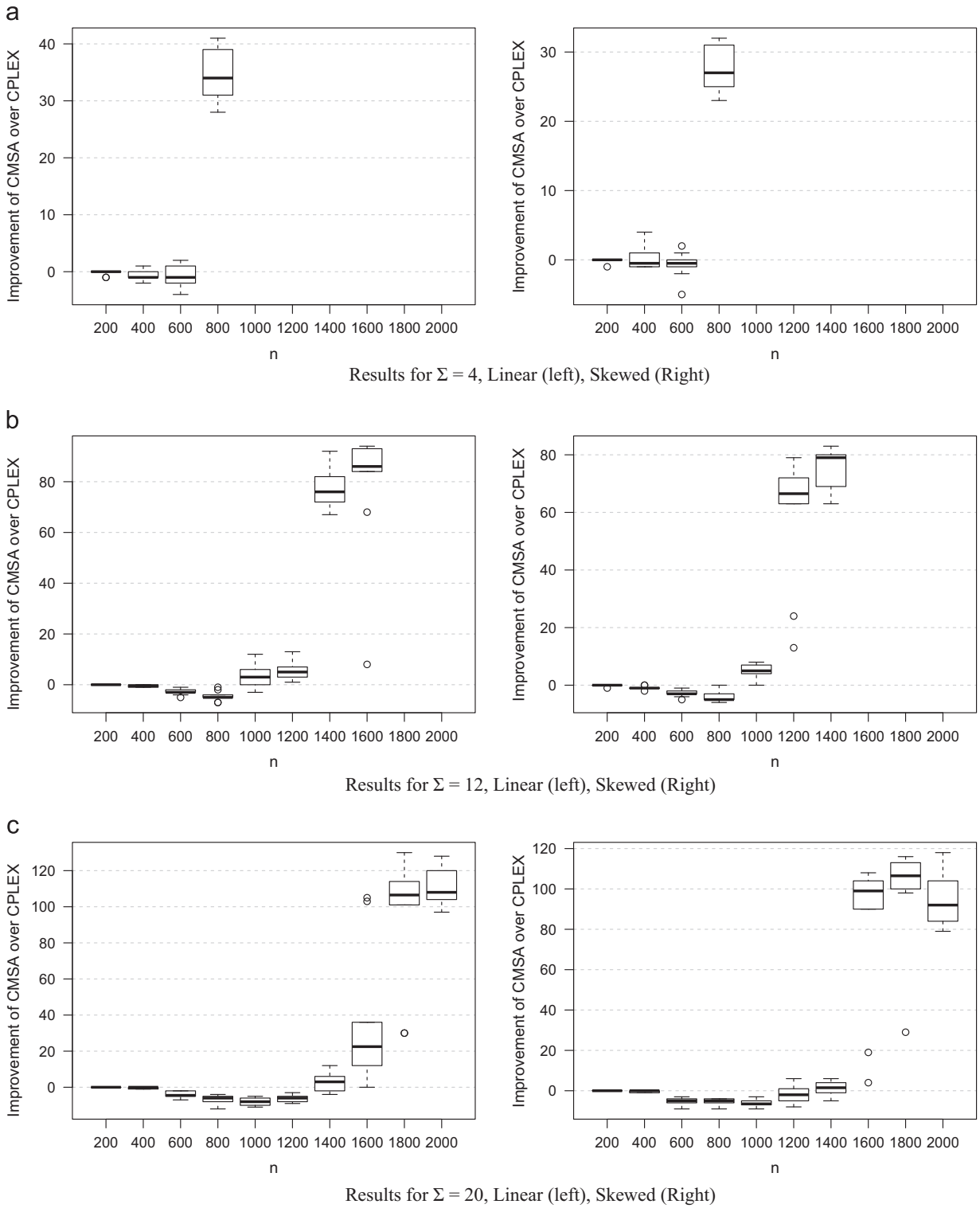


Fig. 2. Differences between the results of CMSA and the ones obtained by applying CPLEX to $I_{\text{LP}_{\text{compl}}}$ concerning the 600 instances of the second benchmark set. Each box shows these differences for the corresponding 10 instances. Note that negative values indicate that CPLEX obtained a better result than CMSA. (a) Results for $\Sigma = 4$, LINEAR (left), SKEWED (right). (b) Results for $\Sigma = 12$, LINEAR (left), SKEWED (right). (c) Results for $\Sigma = 20$, LINEAR (left), SKEWED (right).

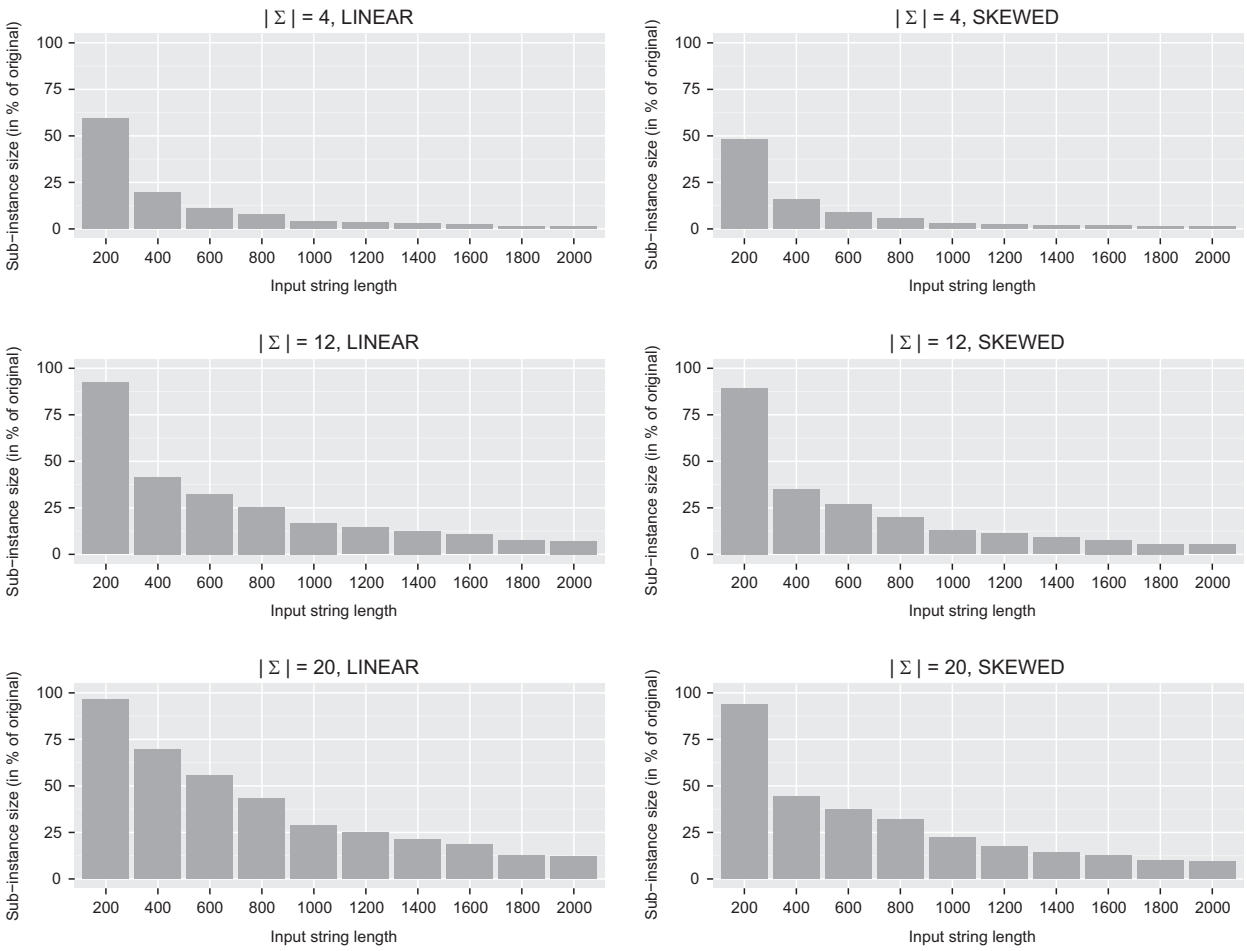


Fig. 3. Average sizes of the sub-instances tackled within CMSA concerning the 600 instances of the second benchmark set. In particular, sub-instance sizes are shown in percent of the original instances. For example, in the case $|\Sigma| = 4$, LINEAR, and $n = 200$, the considered average size of the tackled sub-instances within CMSA is approx. 58% of the size of the original instances.

done in this way, because initial experiments indicated that a low percentage of arcs with negative weights leads to more difficult problem instances. For each possible combination of values for n, $\mathbf{p}_{\text{arc}}$, and *perc*, 10 problem instances were randomly generated. This makes a total of 270 problem instances.

5.2.2. Tuning of CMSA and PGREEDY

CMSA has several parameters for which well-working values must be found: (n_a) the number of solution constructions per iteration, ($\text{age}_{\max}$) the maximum allowed age of solution components, and $d_{\min}$—respectively $d_{\max}$—which determine the degree of greediness which is employed during the process of constructing a non-extensible arborescence. Note that, in the case of the MCA problem, parameter $t_{\max}$—the maximum time in seconds allowed for CPLEX per application to a sub-instance—was not subject to parameter tuning. This is because, in all cases, applications of CPLEX to sub-instances used very few CPU seconds. Therefore, we used a problem instance independent value of 50 for $t_{\max}$.

As in the case of the MCSP problem, we make use of the automatic configuration tool irace [41] for the tuning of the three parameters. More specifically, irace was applied to tune CMSA separately for each combination of n and $\mathbf{p}_{\text{arc}}$. For each of these 9 combinations we randomly generated 12 *training instances*: four instances for each possible value of *perc*. The tuning process for each instance size was given a budget of 1000 runs of CMSA, where each run was given a computation time limit of $n/2$ CPU seconds. Finally, the following parameter value ranges were chosen concerning the three parameters of CMSA:

- $n_a \in \{10, 30, 50\}$
- $\text{age}_{\max} \in \{1, 5, 10, inf\}$, where *inf* means that solution components are never removed from the sub-instance.
- $(d_{\min}, d_{\max})$

$$\in \{(0.0, 0.0), (0.5, 0.5), (0.9, 0.9), (0.0, 0.5), (0.5, 0.9), (0.0, 0.9)\}$$

The 9 applications of iraceproduced the configurations of CMSA as shown in Table 5a. The following trends can be observed. First of all, the desired number of solution constructions per iteration seems to decrease with increasing instance size (in terms of the number of nodes). The same trend can be observed for the values of parameter $\text{age}_{\max}$, whose desired value tends to decrease with increasing instance size. Concerning the greediness of the solution constructions process, rather low greediness seems to be indicated. This is with the exception of the instances with $n = 500$ and $\mathbf{p}_{\text{arc}} = 0.5$ for which the obtained values for $d_{\min}$ and $d_{\max}$ are 0.5, respectively 0.9.

In addition to tuning experiments for CMSA, we also performed tuning experiments for PGREEDY. As PGREEDY constructs solutions in the same way in which they are constructed in CMSA, the parameters involved in PGREEDY are $d_{\min}$ and $d_{\max}$. For the tuning of PGREEDY we

Table 5
Parameter settings produced by irace for the 9 different combinations of n and $\mathbf{p}_{arc}$.
(a) Tuning results for CMSA

$(n, \mathbf{p}_{arc})$	n_a	age_{max}	(d_{min}, d_{max})
(500, 0.1)	50	10	(0.0, 0.0)
(500, 0.3)	30	10	(0.5, 0.5)
(500, 0.5)	30	5	(0.5, 0.9)
(1000, 0.1)	10	5	(0.5, 0.5)
(1000, 0.3)	30	1	(0.5, 0.5)
(1000, 0.5)	10	1	(0.5, 0.5)
(5000, 0.1)	10	5	(0.0, 0.5)
(5000, 0.3)	10	1	(0.0, 0.5)
(5000, 0.5)	10	5	(0.0, 0.5)

(b) Tuning results for PGreedy

$(n, \mathbf{p}_{arc})$	(d_{min}, d_{max})
(500, 0.1)	(0.9, 0.9)
(500, 0.3)	(0.5, 0.9)
(500, 0.5)	(0.9, 0.9)
(1000, 0.1)	(0.5, 0.9)
(1000, 0.3)	(0.5, 0.9)
(1000, 0.5)	(0.0, 0.5)
(5000, 0.1)	(0.0, 0.5)
(5000, 0.3)	(0.0, 0.5)
(5000, 0.5)	(0.0, 0.5)

used the same training instances and the same parameter value combinations as for the tuning of CMSA. The obtained parameter values per combination of n and $\mathbf{p}_{arc}$ are displayed in Table 5b .

5.2.3. Results

In the following we present the experimental results for the benchmark set described in Section 5.2.1. The results are shown in the three Table 6a–c. Note that each table row provides average results over 10 problem instances, and that each considered algorithm was applied exactly once to each problem instance. The layout of the three tables is as follows. The first column provides the size of the input graphs in terms of the number of nodes, whereas the second column indicates the graph density in terms of the edge probability used to generate the corresponding graphs. The third and fourth column provide the results and computation times of PGREEDY. The next three table columns are dedicated to the presentation of the results provided by solving the complete ILP model $I_{LP_{compl}}$ described in Section 4.1.1. The first one of these columns provides the value of the best solution found within $n/2$ CPU seconds, where n is the number of nodes of the respective graphs. The second column provides the computation time (in seconds) needed to solve the problems to optimality (if possible). Finally, the third one of the columns dedicated to $I_{LP_{compl}}$ shows the optimality gap, which refers to the gap between the value of the best valid solution and the current lower bound at the time of stopping a run. The last two

Table 6
Results for the MCA problem instances.

n	$\mathbf{p}_{arc}$	PGREEDY		$I_{LP_{compl}}$			CMSA	
		Mean	**Time**	**Mean**	**Time**	**Gap**	**Mean**	**Time**
(a) Results for instances with $perc = 1\%$								
500	0.1	10,011.7	95.4	−940.1	0.4	0.0	−940.1	7.6
	0.3	2429.6	123.9	−13,450.8	3.5	0.0	−13,450.8	2.6
	0.5	−6244.6	151.8	−28,030.8	12.8	0.0	−28,030.8	4.7
1000	0.1	33,128.9	272.4	−15,263.9	4.5	0.0	−15,251.2	63.0
	0.3	−7501.4	235.4	−62,414.8	42.3	0.0	−62,414.5	86.4
	0.5	−42,531.7	292.9	−106,522.3	152.9	0.0	−106,522.3	58.4
5000	0.1	−114,469.8	1073.0	n/a	n/a	n/a	−530,515.0	1572.8
	0.3	−757,318.8	1167.1	n/a	n/a	n/a	−1,380,184.7	938.1
	0.5	−1,203,516.6	1716.8	n/a	n/a	n/a	−1,959,379.6	365.6
(b) Results for instances with $perc = 10\%$								
500	0.1	47,753.9	100.1	5648.3	0.3	0.0	5653.6	29.8
	0.3	12,247.8	153.9	−11,338.3	3.6	0.0	−11,338.3	3.3
	0.5	6.8	142.9	−26,982.4	11.4	0.0	−26,982.4	11.2
1000	0.1	62,650.5	130.1	−9115.1	3.5	0.0	−9025.9	119.3
	0.3	1152.3	248.4	−61,065.8	38.5	0.0	−61,065.8	51.2
	0.5	−39,504.9	205.3	−105,633.0	145.6	0.0	−105,633.0	51.5
5000	0.1	−104,899.7	1760.7	n/a	n/a	n/a	−526,539.8	1922.3
	0.3	−758,425.1	853.0	n/a	n/a	n/a	−1,379,684.4	861.9
	0.5	−1,203,092.2	1239.2	n/a	n/a	n/a	−1,959,193.4	247.2
(c) Results for instances with $perc = 20\%$								
500	0.1	51,730.7	139.2	10,887.8	0.3	0.0	10,899.6	41.9
	0.3	14,726.6	126.4	−9801.0	3.2	0.0	−9801.0	3.2
	0.5	2258.7	104.4	−25,827.6	11.1	0.0	−25,827.6	32.0
1000	0.1	66,223.5	237.3	−4228.4	2.9	0.0	−4190.0	168.7
	0.3	4079.5	258.1	−59,319.6	36.4	0.0	−59,319.6	73.2
	0.5	−39,905.1	218.8	−104,945.8	136.7	0.0	−104,941.1	57.1
5000	0.1	−105,791.8	1304.2	n/a	n/a	n/a	−522,990.6	1831.9
	0.3	−751,317.6	1768.9	n/a	n/a	n/a	−1,379,042.0	754.9
	0.5	−1,204,557.1	967.2	n/a	n/a	n/a	−1,959,042.5	399.5

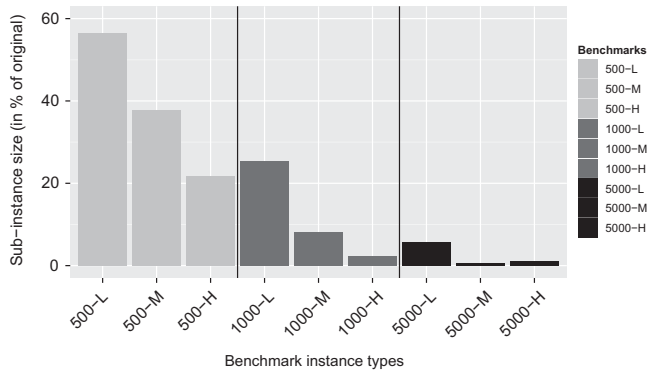


Fig. 4. Average sizes of the sub-instances tackled within CMSA concerning the 270 instances of the benchmark set, categorized into nine different subsets (see the text for a more detailed description). Sub-instance sizes are shown in percent of the original instances. For example, in the case 500-H, that is, graphs with 500 nodes of high density, the considered average size of the tackled sub-instances within CMSA is approx. 22% of the size of the original instances.

columns display the results obtained by CMSA. The first one of these columns provides the results and the second one the average time needed by CMSA to obtain these results. The best-performing algorithm for each table row is marked by a grey background.

The analysis of the results permits to draw the following conclusions:

- Concerning the application of CPLEX to $I_{\text{LP}_{\text{compl}}}$, the size of the input graphs has—as expected—a strong influence on the problem difficulty. In fact, CPLEX was able to solve all problem instances with $n \in \{500, 1000\}$ to optimality. In contrast, CPLEX was not even able to come up with a feasible solution within the allowed computation time in the case of input graphs with $n=5000$. The percentage of nodes that must be included in a solution (*perc*) apparently has no influence on CPLEX. With growing value of *perc*, CPLEX seems even faster in solving the corresponding problem instances.
- For input graphs with $n \in \{500, 1000\}$ CMSA is nearly always able to provide optimal solutions, and is, therefore, competitive with CPLEX. With growing graph density, CMSA is considerably faster than CPLEX. For instances with $n=5000$, CMSA outperforms both CPLEX, which is not able to provide feasible solutions, and the probabilistic greedy algorithm PGREEDY.

Again, as in the case of the MCSP problem, we also provide in the case of the MCA problem information about the average sizes of the sub-instances tackled within CMSA, in comparison to the sizes of the original problem instances. These average sub-instance sizes are shown in Fig. 4 in percent of the original problem instance sizes. More in detail, the 270 benchmark instances are categorized into nine different subsets concerning the number of nodes and the density of the graph. A notation X–Y is used, where X refers to the number of nodes of the graphs, that is, $X \in \{500, 1000, 5000\}$, and Y refers to low, medium and high density, that is, $Y \in \{\text{L}, \text{M}, \text{H}\}$. For example, in the case 500-H, that is, graphs with 500 nodes of high density, the considered average size of the tackled sub-instances within CMSA is approximately 22% of the size of the original instances. It can be observed that this percentage is getting smaller and smaller with growing size of the input graphs and growing density. This is why CPLEX can either solve the sub-instances to optimality or provide nearly-optimal solutions in little computation time, even in the context of large original problem instances.

6. Conclusions and future work

In this paper we introduced a new, generally applicable, algorithm for solving combinatorial optimization problems. The algorithm is an instantiation of the Generate-and-Solve framework from the literature. It is based on the general idea of generating solutions in a probabilistic way, solving the sub-instances of the original problem instance that result from merging the solution components contained in the generated solutions to optimality, and adapting these sub-instances based on an aging mechanism. The proposed algorithm has been applied to two NP-hard combinatorial optimization problems—the minimum common string partition problem and the minimum covering arborescence problem—as test cases. The results have shown that the proposed algorithm is a state-of-the-art method for these problems, especially, for what concerns rather large problem instances.

In future work we will consider the following two lines of research. First, we would like to apply the algorithm to other types of combinatorial optimization problems such as, for example, permutation problems or scheduling problems. Second, we plan to study alternatives for the aging mechanism applied in this work. This is because the aging mechanism results in a binary decision whether a solution component is considered or not. It would be interesting to investigate more fine-grained mechanisms that take into account the quality of the solutions or interactions between solution components.

Acknowledgments

C. Blum was supported by project TIN2012-37930-02 of the Spanish Government. In addition, support is acknowledged from IKERBASQUE (Basque Foundation for Science). J.A. Lozano was partially supported by the IT609-13 program (Basque Government) and project TIN2013-41272P (Spanish Ministry of Science and Innovation).

References

[1] Blum C, Puchinger J, Raidl G, Roli A. Hybrid metaheuristics in combinatorial optimization: a survey. Appl Soft Comput 2011;11(6):4135–51.
[2] Talbi E-G, editor. Hybrid metaheuristics, Studies in computational intelligence, No. 434. Berlin, Germany: Springer Verlag; 2013.
[3] Raidl GR. Decomposition based hybrid metaheuristics. Eur J Oper Res 2015;244(1):66–76.
[4] Boschetti MA, Maniezzo V, Roffilli M, Bolufé Röhler A, Matheuristics: optimization, simulation and control. In: Blesa MJ, Blum C, Di Gaspero L, Roli A, Sampels M, Schaerf A (Eds.), Proceedings of HM 2009sixth international workshop on hybrid metaheuristics. Lecture notes in computer science. vol. 5818. Berlin, Heidelberg: Springer; 2009. pp. 171–77.
[5] Wolsey LA. Integer programming. Hoboken, NJ: Wiley-Interscience; 1998.
[6] Nepomuceno NV, Combinação de metaheurísticas e programação linear inteira: uma metodologia híbrida aplicada ao problema de carregamento de contêineres [Ph.D. thesis], [MSc Thesis]. University of Fortaleza; 2006.
[7] Nepomuceno NV, Pinheiro PR, Coelho ALV. Combining metaheuristics and integer linear programming: a hybrid methodology applied to the container loading problem. In: Proceedings of the XX congreso da sociedade brasileira de computação, concurso de teses e dissertações; 2007. p. 2028–32.
[8] Nepomuceno NV, Pinheiro PR, Coelho ALV. Tackling the container loading problem: a hybrid approach based on integer linear programming and genetic algorithms. In: Cotta C, van Hemert J (Eds.), Proceedings of EvoCOP 2007 - 7th European conference on evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol. 4446. Berlin, Heidelberg: Springer; 2007. p. 154–165.
[9] Nepomuceno, N, Pinheiro P, Coelho ALV. Recent advances in evolutionary computation for combinatorial optimization. In: Studies in Computational Intelligence. vol. 153. Berlin, Germany: Springer Verlag; 2008, p. 87–99. [Ch. A hybrid optimization framework for cutting and packing problems].
[10] Pinheiro PR, Coelho ALV, de Aguiar AB, Bonates TO. On the concept of density control and its application to a hybrid optimization framework: investigation into cutting problems. Comput Ind Eng 2011;61(3):463–72.
[11] Saraiva RD, Nepomuceno NV, Pinheiro PR. The generate-and-solve framework revisited: generating by simulated annealing. In: Middendorf M, Blum C, editors. Evolutionary computation in combinatorial optimization. Lecture notes in computer science, vol. 7832. Berlin, Heidelberg: Springer; 2013. p. 262–73.

[12] Coudert D, Nepomuceno NV, Tahiri I. Energy saving in fixed wireless broadband networks. In: Pahl J, Reiners T, Voß S, editors. Proceedings of INOC 2011 – fifth international conference on network optimization. Lecture notes in computer science, vol. 6701. Berlin, Heidelberg: Springer; 2011. p. 484–9.
[13] Coudert D, Nepomuceno N, Rivano H. Power-efficient radio configuration in fixed broadband wireless networks. Comput Commun 2010;33(8):898–906.
[14] Pinheiro PR, Coelho ALV, de Aguiar AB, de Menezes Sobreira Neto A. Towards aid by generate and solve methodology: application in the problem of coverage and connectivity in wireless sensor networks. Int J Distrib Sensor Netw 2012, article ID 790459.
[15] Applegate D, Bixby R, Chvátal V, Cook W. Finding tours in the TSP, Tech. rep., Forschungsinstitut für Diskrete Mathematik, Germany: University of Bonn; 1999.
[16] Cook W, Seymour P. Tour merging via branch-decomposition. INFORMS J Comput 2003;15(3):233–48.
[17] Klau GW, Ljubić I, Moser A, Mutzel P, Neuner P, Pferschy U, et al. Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In: Proceedings of GECCO 2004 – genetic and evolutionary computation conference. Lecture notes in computer science, vol. 3102. Berlin, Heidelberg: Springer; 2004. p. 1304–15.
[18] Massen F, Deville Y, Hentenryck P, Pheromone-based heuristic column generation for vehicle routing problems with black box feasibility. In: Beldiceanu N, Jussien N, Pinson É (Eds.), Proceedings of CP-AI-OR 2012 – integration of AI and OR techniques in constraint programming for combinatorial optimzation problems. Lecture notes in computer science. vol. 7298. Berlin, Heidelberg: Springer; 2012. p. 260–74.
[19] Massen F, López-Ibánez M, Stützle M, Deville Y, Experimental analysis of pheromone-based heuristic column generation using irace. In: Blesa MJ, Blum C, Festa P, Roli A, Sampels M (Eds.), Proceedings of HM 2013 – international workshop on hybrid metaheuristics. Lecture notes in computer science, vol. 7919. Berlin, Heidelberg: Springer; 2013. p. 92–106.
[20] Blum C, Calvo B. A matheuristic for the minimum weight rooted arborescence problem. J Heuristics 2015;21(4):479–99.
[21] Chen X, Zheng J, Fu Z, Nan P, Zhong Y, Lonardi S, et al. Computing the assignment of orthologous genes via genome rearrangement. In: Proceedings of the asia pacific bioinformatics conference 2005; 2005. p. 363–78.
[22] Mousavi S, Babaie M, Montazerian M. An improved heuristic for the far from most strings problem. J Heuristics 2012;18:239–62.
[23] Meneses C, Oliveira C, Pardalos P. Optimization techniques for string selection and comparison problems in genomics. IEEE Eng Med Biol Mag 2005;24 (3):81–7.
[24] Hsu WJ, Du MW. Computing a longest common subsequence for a set of strings. BIT Numer Math 1984;24(1):45–59.
[25] Smith T, Waterman M. Identification of common molecular subsequences. J Mol Biol 1981;147(1):195–7.
[26] Gusfield D. Algorithms on strings, trees, and sequences, computer science and computational biology. Cambridge: Cambridge University Press; 1997.
[27] Garey MR, Johnson DS. Computers and intractability; a guide to the theory of NP-completeness. San Francisco, LA: W. H. Freeman; 1979.
[28] Goldstein A, Kolman P, Zheng J, Minimum common string partition problem: hardness and approximations. In: Fleischer R, Trippen G (Eds.), Proceedings of ISAAC 2004 - 15th international symposium on algorithms and computation. Lecture notes in computer science, vol. 3341. Berlin, Heidelberg: Springer; 2005. p. 484–95.
[29] Kolman P, Waleń T. Reversal distance for strings with duplicates: linear time approximation using hitting set. In: Erlebach T, Kaklamanis C, editors. Proceedings of WAOA 2007 – fourth international workshop on approximation and online algorithms. Lecture notes in computer science, vol. 4368. Berlin, Heidelberg: Springer; 2007. p. 279–89.
[30] Goldstein I, Lewenstein M. Quick greedy computation for minimum common string partitions. In: Giancarlo R, Manzini G, editors. Proceedings of CPM 2011 – 22nd annual symposium on combinatorial pattern matching. Lecture notes in computer science, vol. 6661. Berlin, Heidelberg: Springer; 2011. p. 273–84.
[31] He D. A novel greedy algorithm for the minimum common string partition problem. In: Mandoiu I, Zelikovsky A, editors. Proceedings of ISBRA 2007 – third international symposium on bioinformatics research and applications. Lecture notes in computer science, vol. 4463. Berlin, Heidelberg: Springer; 2007. p. 441–52.
[32] Ferdous SM, Sohel Rahman M. Solving the minimum common string partition problem with the help of ants. In: Tan Y, Shi Y, Mo H, editors. Proceedings of ICSI 2013 – fourth international conference on advances in swarm intelligence. Lecture notes in computer science, vol. 7928. Berlin, Heidelberg: Springer; 2013. p. 306–13.
[33] Ferdous SM, Sohel Rahman M. A MAX-MIN ant colony system for minimum common string partition problem, CoRR abs/1401.4539. ⟨http://arxiv.org/abs/1401.4539⟩.
[34] Blum C, Lozano JA, Pinacho Davidson P. Iterative probabilistic tree search for the minimum common string partition problem. In: Blesa MJ, Blum C, Voss S, editors. Proceedings of HM 20104 – ninth international workshop on hybrid metaheuristics. Lecture notes in computer science, vol. 8457. Berlin, Germany: Springer Verlag; 2014. p. 154.
[35] Blum C, Lozano JA, Pinacho Davidson P. Mathematical programming strategies for solving the minimum common string partition problem. Eur J Oper Res 2015;242(3):769–77.
[36] Venkata Rao, V, Sridharan R. The minimum weight rooted arborescence problem: weights on arcs case. Tech. Rep., Indian Institute of Management Ahmedabad, Research and Publication Department; 1992.
[37] Tutte WT. Graph theory. Cambridge, UK: Cambridge University Press; 2001.
[38] Bang-Jensen J, Gutin GZ. Digraphs: theory, algorithms and applications. London, UK: Springer Science and Business Media; 2008.
[39] Venkata Rao V, Sridharan R. Minimum-weight rooted not-necessarily-spanning arborescence problem. Networks 2002;39(2):77–87.
[40] Duhamel C, Gouveia L, Moura P, Souza M. Models and heuristics for a minimum arborescence problem. Networks 2008;51(1):34–47.
[41] López-Ibáñez, M, Dubois-Lacoste J, Stützle T, Birattari M, The irace package, iterated race for automatic algorithm configuration, Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium; 2011.

8

The Weighted Independent Domination Problem: Integer Linear Programming Models and Metaheuristic Approaches

The Weighted Independent Domination Problem: Integer Linear Programming Models and Metaheuristic Approaches

Pedro Pinacho Davidson[a,b], Christian Blum[c,*], José A. Lozano[a,d]

[a] *Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, San Sebastian, Spain*
[b] *Escuela de Informática, Universidad Santo Tomás, Concepción, Chile*
[c] *Artificial Intelligence Research Institute (IIIA-CSIC), Campus of the UAB, Bellaterra, Spain*
[d] *Basque Center for Applied Mathematics (BCAM), Bilbao, Spain*

Abstract

This work deals with the so-called weighted independent domination problem, which is an *NP*-hard combinatorial optimization problem in graphs. In contrast to previous work, this paper considers the problem from a non-theoretical perspective. The first contribution consists in the development of three integer linear programming models. Second, two greedy heuristics are proposed. Finally, the last contribution is a population-based iterated greedy metaheuristic which is applied in two different ways: (1) the metaheuristic is applied directly to each problem instance, and (2) the metaheuristic is applied at each iteration of a higher-level framework—known as construct, merge, solve & adapt—to sub-instances of the tackled problem instances. The results of the considered algorithmic approaches show that integer linear programming approaches can only compete with the developed metaheuristics in the context of graphs with up to 100 nodes. When larger graphs are concerned, the application of the populated-based iterated greedy algorithm within the higher-level framework works generally best. The experimental evaluation considers graphs of different types, sizes, densities, and ways of generating the node and edge weights.

Keywords: combinatorial optimization, integer linear programming, heuristics, population-based iterated greedy, construct, merge, solve, and adapt

1. Introduction

The so-called weighted independent domination (WID) problem is a combinatorial optimization problem that was introduced in [1]. This problem is an extension of the well-known independent domination (ID) problem. Given an undirected graph

*Corresponding author
Email addresses: ppinacho@santotomas.cl (Pedro Pinacho Davidson), christian.blum@iiia.csic.es (Christian Blum), ja.lozano@ehu.es (José A. Lozano)

$G = (V,E)$, V is the set of nodes and E refers to the set of edges. An edge $e \in E$ that connects nodes $u \neq v \in V$ is equally denoted by (u,v) and by (v,u). The *neighborhood* $N(v)$ of a node $v \in V$ is defined as $N(v) := \{u \in V \mid (v,u) \in E\}$, the *closed neighborhood* $N[v]$ of a node $v \in V$ is defined as $N[v] := N(v) \cup \{v\}$, and the set of edges incident to a node $v \in V$ is denoted by $\delta(v)$. Note, in this context, that an edge $e \in E$ is called *incident* to a node v, if v forms one of the two endpoints of e. Given an undirected graph $G = (V,E)$, a subset $D \subseteq V$ of the nodes is called a *dominating set* if every node $v \in V \setminus D$ is adjacent to at least one node from D, that is, if for every node $v \in V \setminus D$ exists at least one node $u \in D$ such that $v \in N(u)$. Furthermore, a set $I \subseteq V$ is called an *independent set* if for any pair $v \neq v' \in I$ it holds that v and v' are not connected by an edge in G. Correspondingly, a subset $D \subseteq V$ is called an *independent dominating set* if D is both an independent set and a dominating set. Finally, given an independent dominating set $D \in V$, for all $v \in V \setminus D$ we define the *D-restricted neighborhood* $N(v \mid D)$ as $N(v \mid D) := N(v) \cap D$, that is, the neighborhood of v is restricted to all its neighbors that are in D.

In the WID problem we are given an undirected graph $G = (V,E)$ with node and edge weights. More specifically, for each $v \in V$, respectively $e \in E$, we are given an integer weight $w(v) \geq 0$, respectively $w(e) \geq 0$. The WID problem consists in finding an independent dominating set D in G that minimizes the following cost function:

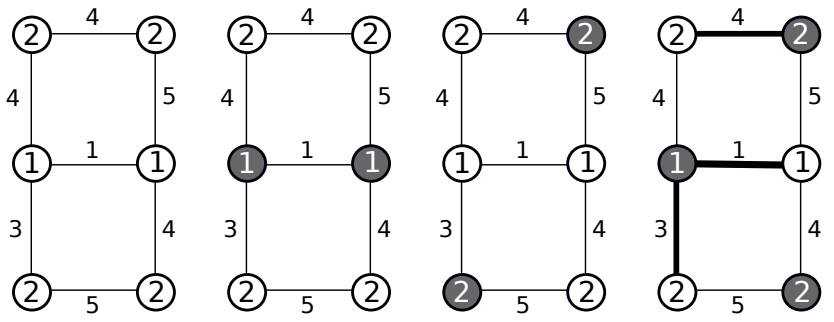
$$f(D) := \sum_{u \in D} w(u) + \sum_{v \in V \setminus D} \min\{w(v,u) \mid u \in N(v \mid D)\} \tag{1}$$

In words, the objective function value of D is obtained by the sum of the weights of the nodes in D plus the sum of the weights of the minimum-weight edges that connect the nodes that are not in D to nodes that are in D. As an example consider the graphics in Figure 1. The node weights are indicated inside the nodes and the edge weights are provided besides the edges. A possible input graph is shown in Figure 1a. An optimal *minimum weight dominating set* (the set of gray nodes) is shown in Figure 1b. However, note that this set is not an independent set because the two nodes that form the set are adjacent to each other. An optimal *minimum weight independent dominating set*[1] is given in Figure 1c. Note that for both, the minimum weight dominating set problem and the minimum weight independent dominating set problem, the edge weights are not considered. Finally, the optimal solution to the WID problem is shown in Figure 1d. The minimum weight edges that are chosen to connect nodes not in D to nodes in D are indicated with bold lines. The objective function value of this solution is 13, which is composed of the nodes weights (2 + 1 + 2) and the edge weights (4 + 1 + 3).

1.1. Our Contribution

So far, the WID problem has only been considered from a theoretical perspective. It is easy to see that the problem is *NP*-hard. This is because with $w(v) = 1$ for all

[1] In this problem, given an undirected graph with node weights, the goal is to find an independent dominating set for which the sum of the weights of the nodes is minimal.



(a) Graph with node and edge weights.

(b) Minimum weight dominating set.

(c) Minimum weight dominating independent set.

(d) Optimal solution to the WID problem.

Figure 1: Example that relates the WID problem with the *minimum weight dominating set* problem and with the *minimum weight independent dominating set* problem.

$v \in V$ and $w(e) = 0$ for all $e \in E$ it reduces to the independent domination problem which was shown to be *NP*-hard in [2]. A linear time algorithm for the WID problem in series-parallel graphs was proposed in [1]. In this work we consider the WID problem in general graphs from an algorithmic perspective. Our contributions are as follows. First, we present three integer linear programming (ILP) models for the WID problem. Second, we propose two different greedy heuristics for solving the problem. The first one is known from the minimum weight independent dominating set problem, while the second one is specifically developed for the WID problem. Third, we propose a so-called population-based iterated greedy (PBIG) algorithm. This algorithm employs an iterated greedy metaheuristic in a population-based fashion, and can therefore be seen as a hybrid between methods based on local search and population-based methods. Finally, in addition to applying the PBIG algorithm directly to all problem instances, we also apply it within a framework known as *construct, merge, solve & adapt* (CMSA) [3]. CMSA was initially introduced for being able to take profit from exact solvers—such as, for example, the general purpose ILP solver CPLEX—in the context of problem instances that are too large to be tackled directly by the respective exact solver. CMSA generates, at each iteration, a number of probabilistic solutions which are used to produce sub-instances to the tackled problem instances. The exact solver is then used to solve the corresponding sub-instance at each iteration of CMSA. In fact, in the context of the WID problem we tried to implement a standard CMSA algorithm based on all three ILP models proposed in this work. Unfortunately, these versions of CMSA were still not efficient enough in order to be able to deal with, for example, graphs with 1000 nodes. Therefore, the idea was to study if CMSA can also be used in order to improve the working of a standard metaheuristic such as PBIG. This gave rise to a CMSA-PBIG algorithm which makes use of the framework of CMSA and uses PBIG (instead of an exact solver) for deriving hopefully good solutions to the corresponding sub-instance at each iteration of CMSA. The obtained results show that this is, indeed, the case. Note that this work is a signficiant extension of a paper that appears in the conference proceedings of EvoCOP 2017 [4]. The extension concerns the

development of two additional ILP models, the application of PBIG within the CMSA framework, and the application to problem instances that represent different types of graphs.

1.2. Related Work

On one side, there is related work for problems similar to the one considered in this work. The *minimum independent dominating set* problem, for example, has recently been approached by a greedy randomized adaptive search procedure (GRASP) in [5]. Another related problem is the *minimum weight dominating set* problem. This problem has been quite popular in recent years as a test case for metaheuristics. The most recent research efforts for this problem have led to the development of an ant colony optimization approach and a genetic algorithm in [6], a hybrid evolutionary algorithm in [7], a hybrid approach combining iterated greedy algorithms and an ILP solver in a sequential way in [8], and a memetic algorithm in [9].

On the other side, there is related work concerning the employed metaheuristic, that is, PBIG. In general, iterated greedy (IG) algorithms have shown to be able to work very well in the context of problems for which a good and fast greedy heuristic is known. Prime examples include those to various scheduling problems such as [10, 11]. The first PBIG approach was proposed in the context of the minimum weight vertex cover problem in [12]. Later, PBIG was also applied to the delimitation and zoning of rural settlements [13] and, as mentioned above, to the minimum weight dominating set problem [8]. Moreover, the literature contains related work concerning applications of CMSA. More specifically, CMSA—using CPLEX as the exact solver— has been applied to hard combinatorial optimization problems such as minimim common string partition [3, 14] and the repetition-free longest common subsequence problem [15].

1.3. Organization

The remainder of this paper is organized as follows. In Section 2, three different ILP models for the WID problem are proposed. Two greedy heuristics are outlined in Section 3. Moreover, the PBIG approach and its application in the CMSA framework are described in Section 4. Finally, an extensive experimental evaluation is provided in Section 5 and conclusions as well as an outlook to future work is given in Section 6.

2. ILP Models

In the following we present three different ILP models for the WIDP problem. These models are experimentally evaluated in Section 5.

2.1. ILP-1: Model based on Indicator Variables

The first one of the proposed ILP models—henceforth called ILP-1—uses three sets of binary variables. For each node $v \in V$ it uses a binary variable x_v. Moreover, for each edge $e \in E$ the model uses a binary variable y_e and a binary variable z_e. Hereby, x_v indicates if v is chosen for the solution. Moreover, z_e indicates if $e \in E$ is selected for connecting a non-chosen node to a chosen one. Variable y_e is an indicator variable, which indicates if e is choosable, or not.

$$
\begin{aligned}
\text{(ILP-1)} \quad \min \quad & \sum_{v \in V} x_v w(v) + \sum_{e \in E} z_e w(e) && && (2)\\
\text{s.t.} \quad & x_v + x_u \leq 1 && \text{for } e = (u,v) \in E && (3)\\
& x_v + \sum_{u \in N(v)} x_u \geq 1 && \text{for } v \in V && (4)\\
& x_v + x_u = y_e && \text{for } e = (u,v) \in E && (5)\\
& z_e \leq y_e && \text{for } e \in E && (6)\\
& x_v + \sum_{e \in \delta(v)} z_e \geq 1 && \text{for } v \in V && (7)\\
& x_v \in \{0,1\} && \text{for } v \in V\\
& y_e \in \{0,1\} && \text{for } e \in E\\
& z_e \in \{0,1\} && \text{for } e \in E
\end{aligned}
$$

Hereby, constraints (3) are the independent set constraints, that is, they make sure that two adjacent nodes can not take part in the solution. Constraints (4) are the dominating set constraints. They ensure that for each node $v \in V$, either the node itself or at least one of its neighbors form part of the solution. These two sets of constraints will be the same in all three ILP models. Constraints (5) ensure the proper setting of the indicator variables. Note that edges that contribute to the objective function value must always connect a node that is not chosen for the solution with a node that is in the solution. Therefore, if—concerning an edge $e = (u,v)$—either v or u is in the solution, variable y_e is forced to take value one, which indicates that this edge is choosable. Constraints (6) relate the indicator variables with the variables that actually show which edges are chosen. In particular, if an indicator variable y_e has value zero, z_e is forced to take value zero, which means e cannot be chosen. Finally, constraints (7) ensure that each node $v \in V$ that does not form part of the solution—that is, when $x_v = 0$—is connected by an edge to a node that forms part of the solution. Due to the fact that the optimization goal concerns minimization, the edge with the lowest weight is chosen for this purpose.

2.2. ILP-2: Eliminating the Indicator Variables

The second ILP model—henceforth called ILP-2—follows the same idea as ILP-1, apart from the fact that it does not require the set of indicator variables. That is, model ILP-2 only makes use of binary variables x_v for all $v \in V$ and binary variables z_e for all $e \in E$. The meaning of these variables is described above.

$$
\begin{aligned}
\text{(ILP-2)} \quad \min \quad & \sum_{v \in V} x_v w(v) + \sum_{e \in E} z_e w(e) && && (8)\\
\text{s.t.} \quad & x_v + x_u \leq 1 && \text{for } e = (u,v) \in E && (9)\\
& x_v + \sum_{u \in N(v)} x_u \geq 1 && \text{for } v \in V && (10)\\
& x_v + x_u \geq z_e && \text{for } e = (u,v) \in E && (11)\\
& (1 - x_v) + (1 - x_u) \geq z_e && \text{for } e = (u,v) \in E && (12)\\
& x_v + \sum_{e \in \delta(v)} z_e \geq 1 && \text{for } v \in V && (13)\\
& x_v \in \{0,1\} && \text{for } v \in V\\
& z_e \in \{0,1\} && \text{for } e \in E
\end{aligned}
$$

Note that the independent set constraints (9), the dominating set constraints (10), and constraints (13) that ensure that each node $v \in V$ that does not form part of the solution is connected by an edge to a node that forms part of the solution, are the same as in model ILP-1. However, the two sets of constraints concerning the indicator variables from model ILP-1 (constraints (5) and (6)) are replaced by constraints (11) and (12). Note that when both x_v and x_u—concerning an edge $e = (u,v) \in E$—are set to zero, constraints (11) force variable z_e to take value zero, which means that an edge that connects two non-selected nodes can not be chosen for the solution. Furthermore, when both x_v and x_u—again concerning an edge $e = (u,v) \in E$—are set to one, constraints (17) force variable z_e to take value zero, which means that an edge that connects two selected nodes can not be chosen for the solution.

2.3. ILP-3: Expicit Variables for the Edge-Weight Contribution

The third ILP model—henceforth called ILP-3—is structurally different to ILP-1 and ILP-2. The main idea is to model the edge-weight contribution of each node in terms of an integer variable q_v for all $v \in V$. Obviously, the edge-weight contribution of a selected node $v \in V$—that is, when $x_v = 1$—must be zero, whereas the edge-weight contribution of a non-selected node $v \in V$ must be equal to the weight of the minimum-weight edge that connects this node to a selected node.

(ILP-3)

$$\min \sum_{v \in V} x_v w(v) + q_v \tag{14}$$

$$\text{s.t. } x_v + x_u \leq 1 \qquad \forall\, e = (u,v) \in E \tag{15}$$

$$x_v + \sum_{u \in N(v)} x_u \geq 1 \qquad \forall\, v \in V \tag{16}$$

$$q_v \leq (1 - x_v)M \qquad \forall\, v \in V \tag{17}$$

$$q_v \geq 0 \qquad \forall\, v \in V \tag{18}$$

$$q_v \geq x_u w(e) - \left(x_v M + \sum_{\substack{e'=(v,v') \in \delta(v) \\ \text{s.t. } w(e') < w(e)}} x_{v'} M \right) \qquad \begin{array}{l} \forall\, v \in V, \\ e = (v,u) \in \delta(v) \end{array} \tag{19}$$

$$x_v \in \{0,1\} \qquad \forall\, v \in V$$

$$q_v \in \{-|V| \cdot M, \ldots, M\} \qquad \forall\, v \in V$$

Observe that the independent set constraints (15) and the dominating set constraints (16) are again as in ILP-1 and ILP-2. In addition, constraints (17) set the upper bound of an edge-contribution variable to zero in case the corresponding node forms part of the solution, that is, of the independent dominating set. In case a node does not form part of the solution, constraints (17) set the upper bound to a large constant M, which we have set to the maximum weight of all edges of the graph in our implementation. Furthermore, constraints (18) set the lower bound of all edge contributions to zero. Finally, constraints (19) correctly set the lower bound of the edge contributions in order to be equal to the weight of the minimum-weight edge connecting the respective node with one of its selected neighbors.

3. Greedy Heuristics

The first one of two different greedy heuristics developed in this work is a simple extension of a well-known heuristic for the minimum weight independent dominating set problem. Given an input graph G, this heuristic starts with an empty solution $S = \emptyset$ and adds, at each step, exactly one node from the remaining graph G' to S. Initally, the *remaining graph* G' is a copy of G. After adding a node $v \in V'$ to S, all nodes from $N[v \mid G']$—that is, from the closed neighborhood of v in G'—are removed from V'. Moreover all their incident edges are removed from E'. In this way, only those nodes that maintain the property of S being an independent set may be added to S. At each step, the node $v \in V'$ that maximizes $\frac{|N(v|G')|}{w(v)}$ is chosen to be added to S, where $N(v \mid G')$ refers to the neighborhood of v in G'. In other words, nodes with a high degree in the remaining graph G' and with a low node weight are preferred. Note that this greedy heuristic does not take the edge weights into account. They are only considered when calculating the objective function value of the final solution S. The pseude-code of this

Algorithm 1 Greedy Heuristic (GREEDY1)

1: **input:** a undirected graph $G = (V, E)$ with node and edge weights
2: $S := \emptyset$
3: $G' := G$
4: **while** $V' \neq \emptyset$ **do**
5: $v^* := \text{argmax}\{\frac{|N(v|G')|}{w(v)} \mid v \in V'\}$ {Ties are randomly resolved}
6: $S := S \cup \{v^*\}$
7: Remove from G' all nodes from $N[v \mid G']$ and their incident edges
8: **end while**
9: **output:** An independent dominating set S of G

heuristic, henceforth referred to as GREEDY1, is shown in Algorithm 1.

In contrast to GREEDY1, the second greedy heuristic is designed to take into account the edge weights already during the process of constructing a solution. The algorithmic framework of this greedy heuristic—henceforth denoted by GREEDY2—is the same as the one of GREEDY1. However, the way in which a node is chosen at each step is different. For the description of this greedy heuristic the following notations are required. First, the maximum weight of any edge in E is denoted by $w_{\max}$. Then, let $S \in V$ be a partial solution, that is, S is an independent set which is not yet a dominating set, but which can be extended to be a dominating set. The *auxiliary objective function value* $f^{\text{aux}}(S)$ is defined as $\sum_{v \in V} c(v \mid S)$, where $c(v \mid S)$ is called the *contribution* of node v with respect to partial solution S. Given S, these contributions are defined as follows:

1. If $v \in S$: $c(v \mid S) := w(v)$
2. If $v \notin S$ and $N(v) \cap S = \emptyset$: $c(v \mid S) := w_{\max}$
3. If $v \notin S$ and $N(v) \cap S \neq \emptyset$: $c(v \mid S) := \min\{w(e) \mid e = (v,u), u \in S\}$

Note that in the case of S being a complete solution, it holds that $f(S) = f^{\text{aux}}(S)$. Now, in order to obtain GREEDY2, line 5 of Algorithm 1 must be exchanged with the following one:

$$v^* := \text{argmin}\left\{f^{\text{aux}}(S \cup \{v\}) \mid v \in V'\right\} \tag{20}$$

4. Population-Based Iterated Greedy Algorithm

A high level description of the implemented PBIG approach—henceforth referred to as PBIG—is given in Algorithm 2. Apart from the input graph G, PBIG requires values for five parameters: (1) the population size $p_{\text{size}} \in \mathbb{Z}^+$, (2) the lower bound ($D^l$) and the upper bound ($D^u$) for the degree of destruction applied to each solution of the population at each iteration, (3) the determinism rate $d_{\text{rate}} \in [0,1]$, and (4) the candidate list size $l_{\text{size}} > 0$. The latter two parameters control the greediness of the probabilistic solution (re-)construction procedure. Moreover, note that for the values of the above-mentioned bounds it must hold that $0 \leq D^l \leq D^u \leq 1$. For the following description, each solution S is a subset of the nodes of V, has an objective function value $f(S)$, and

Algorithm 2 PBIG for the WID problem

1: **input:** input graph G, parameters $p_{\text{size}} > 0$, $D^l, D^u, d_{\text{rate}}, l_{\text{size}} \in [0,1]$
2: $\mathscr{P} :=$ GenerateInitialPopulation$(p_{\text{size}}, d_{\text{rate}}, l_{\text{size}})$
3: **while** termination condition not satisfied **do**
4: $\quad \mathscr{P}_{\text{new}} := \emptyset$
5: $\quad$ **for** each candidate solution $S \in \mathscr{P}$ **do**
6: $\quad\quad \hat{S} :=$ DestroyPartially(S)
7: $\quad\quad S' :=$ Reconstruct$(\hat{S}, d_{\text{rate}}, l_{\text{size}})$
8: $\quad\quad$ AdaptDestructionRate(S, S')
9: $\quad\quad \mathscr{P}_{\text{new}} := \mathscr{P}_{\text{new}} \cup \{S'\}$
10: $\quad$ **end for**
11: $\quad \mathscr{P} :=$ Accept$(\mathscr{P}, \mathscr{P}_{\text{new}})$
12: **end while**
13: **output:** argmin $\{f(S) \mid S \in \mathscr{P}\}$

an individual, possibly dynamic, destruction rate D_S.

The algorithm works as follows. First, the p_{size} solutions of the initial population are generated by function GenerateInitialPopulation$(p_{\text{size}}, d_{\text{rate}}, l_{\text{size}})$ (see line 2 of Alg. 2). Afterwards, each iteration consists of the following steps. First, an empty population $\mathscr{P}_{\text{new}}$, called offspring population, is created. Then, each solution $S \in \mathscr{P}$ is partially destroyed using procedure DestroyPartially(S) (see line 6 of Alg. 2). This results in a partial solution $\hat{S}$. On the basis of $\hat{S}$, a complete solution S' is then constructed using procedure Reconstruct$(\hat{S}, d_{\text{rate}}, l_{\text{size}})$ (see line 7 of Alg. 2). Then, the destruction rate D_S of solution S is adapted depending on the quality of solution S' in function AdaptDestructionRate(S, S'). Each newly obtained complete solution is stored in $\mathscr{P}_{\text{new}}$. Note that the two phases of destruction and re-construction are applied to all solutions from $\mathscr{P}$ independently of each other. When the iteration is completed, procedure Accept$(\mathscr{P}, \mathscr{P}_{\text{new}})$ selects the best p_{size} solutions from $\mathscr{P} \cup \mathscr{P}_{\text{new}}$ for the population of the next iteration. In the case of two solutions from $\mathscr{P} \cup \mathscr{P}_{\text{new}}$ being equal, the criterion used for tie-breaking is based on the individual destruction rates. More specifically, the solution S with the highest individual destruction rate D_S is preferred over the other one. Finally, the algorithm terminates when a predefined CPU time limit is reached, and the best found solution is returned. The four procedures that form the core of PBIG are described in more detail in the following.

GenerateInitialPopulation$(p_{\text{size}}, d_{\text{rate}}, l_{\text{size}})$: This function generates p_{size} solutions for the initial population. For this purpose it uses the mechanism of GREEDY2[2] (see Section 3) in a probabilistic way. At each construction step, first, a random number $\delta \in [0,1]$ is generated. In case $\delta \leq d_{\text{rate}}$, the best node according to the greedy function is chosen. Otherwise, a candidate list of size $\min\{|V'|, l_{\text{size}}\}$, where $V' \subseteq V$ are the

[2]Note that GREEDY2 is chosen over GREEDY1 because, as it will be shown later, GREEDY2 generally works better than GREEDY1.

nodes that can be selected at the current construction step, is generated, and one of the nodes from the candidate list is chosen uniformly at random. Note also that the initial destruction rate (D_S) of each solution S is set to the lower bound D^l for the destruction rates.

DestroyPartially(S): In this function, $\max\{3, \lfloor D_S \cdot |S| \rfloor\}$ randomly selected nodes are removed from S, where D_S is the current individual destruction rate of solution S.

Reconstruct($\hat{S}, d_{\text{rate}}, l_{\text{size}}$): Given as input a partial solution $\hat{S}$, this function re-constructs a complete solution S' in the same way in which solutions are probabilistically constructed in the context of generating the initial population (see above). Moreover, the initial destruction rate $D_{S'}$ of S' is set to D^l.

AdaptDestructionRate(S, S'): The individual destruction rate D_S of solution S (from which partial solution $\hat{S}$ was obtained) is updated on the basis of the lower bound D^l and the upper bound D^u as follows. If $f(S') < f(S)$, the value of D_S is set back to the lower bound D^l. Otherwise, the value of D_S is incremented by a certain amount. After initial experiments, we determined this amount to be 0.05. If the value of D_S, after this update, exceeds the upper bound D^u, it is set back to the lower bound D^l.

Note that the idea behind this way of dynamically changing the value of D_S is as follows. As long as the algorithm is able to improve a solution using a low destruction rate, this rate is kept low. In this way, the re-construction is faster. Only when the algorithm seems not to be able to improve over a solution, the individual destruction rate of this solution is increased in a step-wise manner.

4.1. Application of PBIG in the CMSA Framework

As mentioned before, the CMSA framework was introduced in [3] in order to be able to take profit from an efficient exact solver even in the context of problem instances that are too large to be solved directly by the exact solver. The general idea of CMSA is as follows. At each iteration, solutions to the tackled problem instance are generated in a probabilistic way. The solution components found in these solutions are then added to a sub-instance of the original problem instance. Subsequently, an exact solver such as, for example, CPLEX is used to solve the sub-instance to optimality. Moreover, the algorithm is equipped with a mechanism for deleting seemingly useless solution components from the sub-instance. This is done such that the sub-instance has a moderate size and can be solved rather quickly to optimality.

In the context of the WID problem, the set of solution components corresponds to the set of nodes of the input graph. Moreover, solutions can be probabilistically constructed by a probabilistic version of either GREEDY1 or GREEDY2. In fact, both greedy heuristics can be made probabilistic by the mechanism described in the context of PBIG in the previous section. For this purpose we require two parameters: (1) the determinism rate parameter (called d^{cmsa}_{rate} in the context of CMSA-PBIG) and (2) the candidate list size (called l^{cmsa}_{size} in the context of CMSA-PBIG). Our initial idea was to use one of the three proposed ILP models in order to solve the sub-instances within CMSA-PBIG to optimality. However, even though smaller than the original problem

instances, this idea turned out to be inefficient in the case of graphs with 500 and 1000 nodes. Therefore, we implemented the following option. Instead of applying an exact solver to each sub-instance, the PBIG algorithm is applied with a certain time limit to each sub-instance. The resulting pseudo-code of CMSA-PBIG is provided in Algorithm 3.

Each algorithm iteration works as follows. First, the best-so-far solution S_{bsf} is initialized to NULL, indicating that no such solution exists yet. Moreover, the current sub-instance $V' \subseteq V$ (where V is the set of nodes of the input graph G) is initialized to the empty set. Then, at each iteration, n_{a} solutions are probabilistically generated in function ProbabilisticSolutionGeneration($\text{opt}_{\text{greedy}}$, d^{cmsa}_{rate}, l^{cmsa}_{size}), either making use of GREEDY1 (in case $\text{opt}_{\text{greedy}} = 0$) or of GREEDY2 (in case $\text{opt}_{\text{greedy}} = 1$). The nodes found in the constructed solutions are then added to V'. Furthermore, each node $v \in V'$ has an *age*, labelled $age[v]$, which is initialized to zero. Next, PBIG is applied in function ApplyPBIG(V') to find a high-quality solution, however, restricted to the nodes from V'; that is, the solution (re-)construction process of PBIG is restricted to choose nodes from V'. If the resulting solution, labelled S'_{pbig}, is better than the current best-so-far solution S_{bsf}, solution S'_{pbig} is adopted as the new best-so-far solution. Next, sub-instance V' is adapted on the basis of solution S'_{pbig} in conjunction with the age values of the nodes in V'. This is done in function Adapt(V', S'_{pbig}, age_{max}) as follows. First, the age of each node in $V' \setminus S'_{\text{pbig}}$ is incremented while the age of each node in S'_{pbig} is re-initialized to zero. Subsequently, those nodes from V' with an age value greater than age_{max}—which is a parameter of the algorithm—are removed from V'. This causes that nodes that are never selected for the best solutions of PBIG do not slow down the working of PBIG in coming iterations.

5. Experimental Evaluation

The following seven algorithmic approaches are evaluated on a large variety of benchmark instances: (1–3) the application of CPLEX to the three ILP models (ILP-1, ILP-2, and ILP-3), (4) GREEDY1, (5) GREEDY2, (6) PBIG, and (7) CMSA-PBIG. All techniques were implemented in ANSI C++ using GCC 4.6.3 for compiling the software. Moreover, we used CPLEX version 12.6 in single-threaded execution for solving the ILP models. The experimental results that are presented in the following were obtained on a cluster of 32 computers with Intel® Xeon® X5660 CPUs of 6 nuclei of 2.8 GHz and (in total) 48 Gigabytes of RAM. For each run of CPLEX we allowed a maximum of 4 Gigabytes of RAM. In the following, first, the set of benchmark instances is described. Then, a detailed analysis of the experimental results is presented.

5.1. Benchmark Instances

Two types of graphs were considered for the experimental evaluation: (1) random graphs and (2) random geometric graphs. In each case, graphs of different properties—for what concerns, for example, the density—and different sizes were created. In particular, for each type we generated graphs of 100, 500 and 1000 nodes, that is, $|V| \in \{100, 500, 1000\}$. The random graphs were generated adding edges between nodes totally at random, with a given probability ep for each edge. This probability

Algorithm 3 CMSA-PBIG for the WID problem

input: input graph G, parameter values for PBIG and parameter values for d_{rate}^{cmsa}, l_{size}^{cmsa}, age_{max}, n_{a}, t_{max}, and $\text{opt}_{\text{greedy}}$
$S_{\text{bsf}} := \text{NULL}$
$V' := \emptyset$
$age[v] := 0$ for all $v \in V$
while CPU time limit not reached **do**
  for $i = 1, \ldots, n_{\text{a}}$ **do**
    $S :=$ ProbabilisticSolutionGeneration($\text{opt}_{\text{greedy}}$, d_{rate}^{cmsa}, l_{size}^{cmsa})
    for all $v \in S$ **and** $v \notin V'$ **do**
      $age[v] := 0$
      $V' \leftarrow V' \cup \{v\}$
    end for
  end for
  $S'_{\text{pbig}} \leftarrow$ ApplyPBIG(V')
  if $f(S'_{\text{pbig}}) < f(S_{\text{bsf}})$ **then** $S_{\text{bsf}} := S'_{\text{pbig}}$
  Adapt(V', S'_{pbig}, age_{max})
end while
output: S_{bsf}

controls the density of the graph. In particular, we considered $ep \in \{0.05, 0.15, 0.25\}$. The random geometric graphs were created as follows. First, the $|V|$ nodes were assigned to random coordinates from the unit square. Then, a *radius* (r) was fixed and each pair of nodes at a distance smaller or equal than the radius was connected by an edge. The radius controls the density of the graph, that is, the larger the radius the denser is the resulting graph. In order to produce graphs with densities comparable to the ones of the random graphs we considered $r \in \{0.14, 0.24, 0.34\}$. The main difference between random geometric graphs and random graphs is that in the former ones only nodes that are placed close together may be connected while in the latter ones any two nodes may be connected.

Three different schemes for generating the node and edge weights were considered. In the first scheme, both node and edge weights were drawn uniformly at random from $\{0, \ldots, 100\}$. Henceforth, we call the resulting graphs *neutral graphs*. In the second scheme, node weights were drawn uniformly at random from $\{0, \ldots, 1000\}$ and edge weights were drawn uniformly at random from $\{0, \ldots, 10\}$. In these graphs, henceforth called *node oriented* graphs, the choice of the nodes is presumably very important because of the nodes themselfs. Finally, in the third scheme node weights were drawn uniformly at random from $\{0, \ldots, 10\}$ and edge-weights were drawn uniformly at random from $\{0, \ldots, 1000\}$. In these *edge-oriented* graphs, the choice of the nodes is important due to edges that are made available for connecting non-chosen nodes to chosen nodes.

For each combination of a graph type, a number of nodes, an edge probability (respectively, a radius), and a weight generation scheme, we produced 10 problem instances. This makes a total of 540 graphs: 270 random graphs (this set is henceforth

called RG) and 270 random geometric graphs (this set is henceforth called RGG).

5.2. Tuning Experiments

The automatic configuration tool irace [16] was used in order to find well-working values for the parameters of PBIG and CMSA-PBIG. In this section we describe the experimental setup used for the tuning experiments, and the tuning results.

5.2.1. Tuning of PBIG

The following five parameters were considered in the case of PBIG: p_{size}, D^l, D^u, d_{rate} and l_{size}. The tuning tool was applied separately for each combination of graph type, number of nodes and the weight generation scheme. Note that no separate tuning was performed concerning the graph density (depending on ep in the context of random graphs, respectively r in the context of random geometric graphs). This is because, after initial runs, it was shown that the other parameters have a higher influence on the behavior of the algorithm. Summarizing, irace was applied 18 times with a budget of 1000 applications of PBIG per tuning run.

For each application of PBIG a time limit of $|V| \cdot 3$ CPU seconds was fixed. For each run of irace, two tuning instances were generated for each combination of graph type, number of nodes, graph density, and weight generation scheme. This gives a total of six tuning instances per run of irace. The following parameter value ranges were considered for each tuning run:

- $p_{\text{size}} \in \{1, 10, 50, 100\}$.
- For the lower and upper bound values of the destruction percentage, the following value combinations were considered: $(D^l, D^u) \in \{$(10,10), (20,20), (30,30), (40,40), (50,50), (60,60), (70,70), (80,80), (90,90), (10,50), (30,70), (50,90)$\}$. Note that in those cases in which both bounds have the same value, the percentage of deleted nodes is always the same.
- $d_{\text{rate}} \in \{0.0, 0.3, 0.5, 0.7, 0.9\}$.
- $l_{\text{size}} \in \{1, 3, 5, 10\}$.

The results of the tuning processes in the case of random graphs are presented in Table 1. The trends are as follows: the population size (p_{size}) should be rather high. Interestingly, the option of a dynamically changing value for the destruction rate (D^l, D^u) never resulted best. In most cases a fixed value greater than 0.5 is selected. The determinism rate (d_{rate}) should be rather low, specially when large graphs are concerned. Finally, the candidate list size (l_{size}) should be rather high.

In the same way, the results of the tuning processes for random geometric graphs are shown in Table 2. Here, the trends are as follows: the chosen population sizes are rather high, with one exception ($|V| = 100$, node-oriented). Interestingly, this is also the only case in which a dynamically changing destruction rate was selected. In the other cases a fixed value greater than 0.4 is selected for this parameter. The selected determinism rate tends to decrease with increasing graph size, and the candidate list size should be rather high.

Table 1: Results of tuning PBIG with irace for random graphs.

Weight scheme	$\|V\|$	p_{size}	(D^l, D^u)	d_{rate}	l_{size}
neutral	100	50	$(0.7, 0.7)$	0.0	5
	500	100	$(0.5, 0.5)$	0.3	10
	1000	100	$(0.5, 0, 5)$	0.0	10
node-oriented	100	10	$(0.8, 0.8)$	0.0	10
	500	100	$(0.6.0.6)$	0.0	5
	1000	100	$(0.6, 0.6)$	0.3	10
edge-oriented	100	100	$(0.6, 0.6)$	0.0	10
	500	50	$(0.6, 0.6)$	0.0	10
	1000	100	$(0.5, 0.5)$	0.0	10

Table 2: Results of tuning PBIG with irace for random geometrics graphs.

Weight scheme	$\|V\|$	p_{size}	(D^l, D^u)	d_{rate}	l_{size}
neutral	100	50	$(0.8, 0.8)$	0.3	10
	500	100	$(0.5, 0, 5)$	0.5	5
	1000	100	$(0.4, 0.4)$	0.3	5
node-oriented	100	1	$(0.5, 0.9)$	0.5	10
	500	100	$(0.5, 0.5)$	0.3	10
	1000	50	$(0.6, 0.6)$	0.3	5
edge-oriented	100	100	$(0.8, 0.8)$	0.9	10
	500	50	$(0.4, 0.4)$	0.0	10
	1000	100	$(0.4, 0.4)$	0.0	10

5.2.2. Tuning of CMSA-PBIG

In addition to the five parameters of PBIG, the tuning procedure for CMSA-PBIG must additionally consider the six parameters of the CMSA framework of CMSA-PBIG: d_{rate}^{cmsa}, l_{size}^{cmsa}, age_{max}, n_{a}, t_{max}, and $\text{opt}_{\text{greedy}}$. The parameter value ranges for the five PBIG-parameters were chosen as for the tuning procedure of the stand-alone PBIG. For the six additional parameters of CMSA-PBIG, the value ranges considered were the following:

- $d_{\text{rate}}^{cmsa} \in \{0.0, 0.3, 0.5, 0.7, 0.9\}$.
- $l_{\text{size}}^{cmsa} \in \{1, 3, 5, 10, 20\}$.
- $\text{age}_{\text{max}} \in \{1, 3, 5, 10, inf\}$.
- $n_{\text{a}} \in \{1, 10, 30, 50\}$.
- $t_{\text{max}} \in \{1, 2, 5, 10, 50\}$.
- $\text{opt}_{\text{greedy}} \in \{0, 1\}$, where value 0 represents the selection of GREEDY1 for the probabilistic construction of solutions, and value 1 the selection of GREEDY2 for this purpose.

Table 3: Results of tuning CMSA-PBIG with irace for random graphs.

Weight scheme	$\|V\|$	p_{size}	(D^l, D^u)	d_{rate}	l_{size}	d_{rate}^{cmsa}	l_{size}^{cmsa}	$\text{age}_{\max}$	n_{a}	$t_{\max}$	$\text{opt}_{\text{greedy}}$
neutral	100	50	(0.3, 0.7)	0.7	3	0.5	3	inf	30	1	1
	500	50	(0.3, 0.3)	0.3	5	0.3	5	inf	30	5	1
	1000	10	(0.4, 0.4)	0.0	5	0.3	20	5	10	5	0
node-oriented	100	1	(0.6, 0.6)	0.7	10	0.7	10	inf	30	1	0
	500	10	(0.3, 0.3)	0.3	10	0.5	20	3	50	10	0
	1000	10	(0.5, 0.5)	0.0	5	0.5	20	10	30	50	1
edge-oriented	100	1	(0.5, 0.9)	0.5	5	0.5	5	inf	30	5	1
	500	100	(0.3, 0.3)	0.0	5	0.3	20	3	50	10	0
	1000	50	(0.4, 0.4)	0.3	10	0.5	10	3	50	50	1

Table 4: Results of tuning CMSA-PBIG with irace for random geometrics graphs.

Weight scheme	$\|V\|$	p_{size}	(D^l, D^u)	d_{rate}	l_{size}	d_{rate}^{cmsa}	l_{size}^{cmsa}	$\text{age}_{\max}$	n_{a}	$t_{\max}$	$\text{opt}_{\text{greedy}}$
neutral	100	10	(0.3, 0.7)	0.3	3	0.3	20	10	30	2	1
	500	10	(0.3, 0.7)	0.3	3	0.3	20	10	30	2	1
	1000	100	(0.2, 0.2)	0.3	10	0.5	20	inf	30	50	1
node-oriented	100	10	(0.5, 0.5)	0.7	10	0.5	10	1	50	2	0
	500	1	(0.2, 0.2)	0.7	10	0.3	20	inf	30	5	0
	1000	50	(0.2, 0.2)	0.7	3	0.5	10	5	50	5	0
edge-oriented	100	1	(0.3, 0.3)	0.7	10	0.5	10	5	50	5	1
	500	1	(0.3, 0.3)	0.7	10	0.5	10	5	50	5	1
	1000	50	(0.3, 0.7)	0.0	10	0.5	10	10	1	5	1

The setup of the tuning processes for CMSA-PBIG was the same as for the ones of PBIG. That is, irace was applied 18 times with a budget of 1000 applications of CMSA-PBIG per tuning run. The time limit of $|V| \cdot 3$ CPU seconds per execution was applied, and the same tuning instances were used.

The results of the tuning processes in the case of random graphs are shown in Table 3. Note that when referring, in the following, to PBIG, we mean the application of PBIG within CMSA-PBIG. The following remarkable trends can be observed: the selected destruction rate of PBIG is generally lower than the one chosen for the stand-alone version of PBIG. The determinism rate of PBIG decreases with increasing graph size. Interestingly, the required candidate list size of PBIG is generally lower than the one of the probabilistic solution construction mechanism of the CMSA framework. Concerning the remaining parameters of the CMSA framework, the following can be observed: the determinism rate of the probabilistic solution construction mechanism of CMSA-PBIG tends to decrease with increasing graph size. The number of solutions constructed per iteration (n_{a}) is around 30. The time limit for the application of PBIG at each iteration ($t_{\max}$) increases with increasing graph size, and finally, the selection of GREEDY1 (value 0 of $\text{opt}_{\text{greedy}}$) is most common for graphs generated according to the node-oriented weight scheme, while GREEDY2 seems preferred for the remaining graphs.

In the same way, the results of the tuning processes concerning random geometric graphs are presented in Table 4. The trends that can be observed in this case are very similar to those already outlined for random graphs.

5.3. Numerical Results

The seven solution approaches were applied exactly once to each problem instance. The computation time limit for the applications of CPLEX, PBIG and CMSA-PBIG

was $|V| \cdot 3$ seconds for each graph. The results are presented in numerical form in two tables: Table 5 contains the results for all random graphs and Table 6 contains the results for all random geometric graphs. The two tables have the following format. The first three table columns indicate the number of nodes in the graph ($|V|$), the weight generation scheme, and the graph density in terms of the edge probability (ep) for random graphs and the radius (r) for random geometric graphs. The results of the seven approaches are presented in two columns for each approach. The first one of these columns (with heading **result**) provides, in all seven cases, the average result obtained for the corresponding 10 problem instances. In the case of GREEDY1 and GREEDY2, the second column provides the average computation times (in seconds). In the case of the application of CPLEX to the three ILP models, the second columns provide the average optimality gaps (in %) that correspond to the results shown in the first columns. Finally, in the case of PBIG and CMSA-PBIG, the second column shows the average time at which the best solutions of a run were found. Note that the best result of each table row is shown with gray background. In addition, we applied a statistical significance test to the results of each table row. More specifically, in each table row all approaches were compared to the best-performing approach and the results of those approaches who are statistically equivalent to the best-performing approach are marked by the ★ symbol (significance level of 0.05). The statistical differences have been assessed using the Friedman test and the p-values have been corrected for multiple comparison using Finner's procedure [17].

Additionally, we aimed for detecting the differences between the algorithms (if any) for large subsets of the problem instances. First, all the algorithms have been compared simultaneously using the Friedman test. Then, given that in all the cases the test rejects the hypothesis that all the algorithms perform equally, all the pairwise comparisons have been tested using the Nemenyi post-hoc test [17]. The corresponding results are shown in Figures 2 and 3 by means of so-called criticial difference plots. Briefly, each approach is positioned in the segment according to its average ranking concerning the considered subset of instances. Then, the critical difference (CD) is computed for a significance level of 0.05 and the performance of those algorithms that have a difference lower than CD are regarded as equal—that is, no difference of statistical significance can be detected. This is indicated in the graphic by horizontal bars joining the respective algorithms.[3]

The experimental results allow us to make the following observations:

- When considering all instances together—see Figure 2a—CMSA-PBIG is the best-performing algorithm, followed by PBIG. The next group of approaches is composed of the application of CPLEX to the three ILP models. Concerning the order between them, ILP-2 is generally the best-performing one, followed by ILP-3 and then ILP-1. Finally, the worst-performing group of algorithms is composed of the two greedy algorithms, with GREEDY2 outperforming GREEDY1. All differences are statistically significant.

[3]Note that all the tests and the plots have been generated using R's **scmamp** package [18], available at https://github.com/b0rxa/scmamp.

Table 5: Numerical results for random graphs.

$\|V\|$	Weight scheme	ep	GREEDY1		GREEDY2		ILP-1		ILP-2		ILP-3		PBIG		CMSA-PBIG	
			result	**time**	**result**	**time**	**result**	**gap**	**result**	**gap**	**result**	**gap**	**result**	**time**	**result**	**time**
100	N	0.05	3589.1	<0.1	3519.1	<0.1	3049.8	0.7	3049.8	0.0	3051.9★	0.5	3049.8	0.54	3049.8	8.0
		0.15	3014.4	<0.1	2981.3	<0.1	2445.2	30.4	2396.3	28.3	2398.4★	44.6	2330.9	28.31	2338.1★	48.3
		0.25	2883.5	<0.1	2796.1	<0.1	2161.1	31.3	2114.3★	24.3	2167.6	58.8	2070.9	0.16	2093.9★	54.1
	V	0.05	10465.6	<0.1	11756.6	<0.1	7715.4	0.0	7715.4	0.0	7715.4	0.0	7747.0★	76.47	7860.0★	59.4
		0.15	4891.6	<0.1	5845.4	<0.1	3046.6	0.0	3046.6	0.0	3046.6	0.0	3050.3★	16.9	3070.3★	40.6
		0.25	3297.5	<0.1	3488.9	<0.1	1808.4	0.0	1808.4	0.0	1808.4	0.0	1808.4	3.5	1808.4	15.5
	E	0.05	25698.7	<0.1	22269.3	<0.1	14378.7	0.0	14378.7	0.0	14378.7	0.0	14378.7	0.78	14378.7	37.9
		0.15	27528.4	<0.1	23404.5	<0.1	15473.0	42.4	15198.9★	39.6	15098.3★	57.7	14687.8★	0.19	14563.3	17.2
		0.25	25451.4	<0.1	21770.0	<0.1	16001.7	61.5	14557.7★	28.5	15346.3★	69.3	14506.6★	<0.1s	14382.2	37.7
500	N	0.05	14143.1	<0.1	13535.1	<0.1	11857.9	51.9	11809.3	50.5	12882.6	91.7	10327.3★	127.37	10140.6	667.3
		0.15	12268.5	<0.1	11558.0	<0.1	10050.1	69.4	9928.8	68.1	13196.7	98.1	8297.5★	47.33	8046.2	688.2
		0.25	11630.3	<0.1	10429.5	0.1	11341.1	78.0	9465.7★	73.9	12722.4	99.0	7633.7★	10.16	7443.0	538.6
	V	0.05	15501.5	<0.1	18298.1	<0.1	12557.6	52.6	11403.0★	47.1	12059.3	58.0	9822.6★	924.21	9588.2	912.1
		0.15	6496.3	<0.1	7300.1	<0.1	5940.1	61.3	6122.4	59.3	5474.6	80.5	3581.7★	219.52	3557.8	599.8
		0.25	4212.4	<0.1	4463.7	0.1	8166.2	79.4	10145.8	82.9	3628.7★	85.6	2590.6★	52.82	2586.5	521.1
	E	0.05	125357.6	<0.1	108178.0	<0.1	97915.3	82.9	89580.7	81.1	107463.5	98.7	70028.6★	1008.89	67528.9	713.6
		0.15	114951.0	<0.1	102365.1	<0.1	107834.7	93.4	91564.1★	91.7	117036.1	99.9	64673.8★	109.71	62950.1	798.1
		0.25	111012.3	<0.1	99018.2	0.1	100750.3	95.0	88687.7★	94.0	113798.0	99.9	64112.7★	33.26	61411.1	294.9
1000	N	0.05	25569.6	<0.1	23489.7	0.1	25156.1	69.5	25986.0	68.7	27158.7	97.0	17723.7	1750.12	17819.4★	1262.1
		0.15	20827.1	<0.1	20689.1	0.2	21117.8	81.1	18282.9★	76.5	22984.5	99.1	14731.3★	260.52	14461.9	813.3
		0.25	20858.8	<0.1	19280.5	0.4	158097.1	93.2	88053.8	88.7	21821.5	99.5	13968.5★	340.14	13695.6	1480.6
	V	0.05	18048.6	<0.1	20142.3	0.1	35766.3	83.3	39356.1	83.3	15464.1★	77.4	11301.7★	2018.01	11034.6	1363.5
		0.15	7408.3	<0.1	7987.4	0.2	35678.2	91.3	35904.3	97.0	11586.3	92.3	4540.3★	695.71	4456.4	1362.7
		0.25	4941.9★	<0.1	5566.6	0.4	35838.9	96.5	22569.5	100.0	11674.7	96.7	3519.0★	306.67	3460.4	1049.3
	E	0.05	238600.0	<0.1	202992.0	0.1	209391.8	91.1	198540.7	90.1	229778.7	99.7	133667.8★	2121.82	130889.2	1786.2
		0.15	209709.3	<0.1	182726.6	0.2	832437.1	97.7	191911.2	96.2	229888.1	100.0	123760.9★	99.67	120997.2	1478.6
		0.25	198537.0	<0.1	181150.0	0.4	1128240.1	98.6	2041102.4	99.7	221492.0	100.0	124193.3★	447.95	120288.7	1359.1

Table 6: Numerical results for random geometric graphs.

$\|V\|$	Weight scheme	r	GREEDY1		GREEDY2		ILP-1		ILP-2		ILP-3		PBIG		CMSA-PBIG	
			result	**time**	**result**	**time**	**result**	**gap**	**result**	**gap**	**result**	**gap**	**result**	**time**	**result**	**time**
100	N	0.14	3870.6	<0.1	3646.4	<0.1	3261.1	0.0	3261.1	0.0	3261.1	0.0	3261.1	11.93	3261.1	3.3
		0.24	3798.8	<0.1	3378.1	<0.1	2942.9	21.7	2917.5★	15.6	2884.5★	3.0	2882.5	3.04	2882.5	2.7
		0.34	3766.6	<0.1	3388.1	<0.1	2878.5★	26.5	2841.8★	8.6	2876.7★	17.6	2828.0	0.7	2828.0	27.9
	V	0.14	7364.9	<0.1	7514.3	<0.1	5731.8	0.0	5731.8	0.0	5731.8	0.0	5731.8	12.43	5740.0★	2.0
		0.24	2880.1	<0.1	2724.2	<0.1	1981.8	0.0	1981.8	0.0	1981.8	0.0	1981.8	<0.1s	1981.8	2.2
		0.34	1741.0	<0.1	1832.3	<0.1	940.5	0.3	940.5	0.0	940.5	0.0	968.8★	<0.1s	940.5	1.2
	E	0.14	29011.6	<0.1	24998.3	<0.1	19179.4	0.0	19179.4	0.0	19179.4	0.0	19313.2★	117.61	19179.4	10.1
		0.24	35312.1	<0.1	28647.1	<0.1	22519.7	20.5	22325.5★	16.5	22065.9	8.2	22108.3★	6.36	22065.9	5.5
		0.34	37929.4	<0.1	30503.4	<0.1	24295.9★	31.1	23717.6	7.0	24009.2★	19.0	23900.0★	1.14	23717.6	76.7
500	N	0.14	18408.3	<0.1	16208.4	<0.1	14942.7	56.0	15016.4	54.9	15457.2	85.1	13341.5★	835.38	13301.2	548.6
		0.24	18548.8	<0.1	15882.9	<0.1	19028.3	72.7	15668.7	66.5	16788.4	95.5	12943.1★	195.71	12783.3	129.6
		0.34	18311.4	<0.1	15497.8	0.1	18602.1	75.8	15468.3	67.8	18948.1	97.6	13065.5★	6.18	12954.8	327.5
	V	0.14	6807.8	<0.1	7087.8	<0.1	4377.0	0.4	4377.0	0.0	4381.6★	1.4	4400.9★	513.73	4389.7★	165.8
		0.24	3526.6	<0.1	3121.1	<0.1	2619.3★	7.6	2596.4★	5.8	2665.1	39.0	2573.7	9.71	2573.7	17.8
		0.34	2632.2	<0.1	2830.7	0.1	3933.7	51.2	2205.3★	17.0	2305.3	63.8	2181.6	7.17	2181.6	13.1
	E	0.14	177816.7	<0.1	148267	<0.1	128902.8	66.2	128784.8	65.7	129521.2	91.8	112404.8★	730.98	111638.2	680.5
		0.24	181739.2	<0.1	149980.2	<0.1	175979.6	76.6	147009.0	71.5	162967.5	98.3	118765.6★	16.28	117439.8	405.9
		0.34	190996.4	<0.1	155128.6	0.1	180632.5	78.4	149305.9	71.2	188131.4	99.0	122977.3★	5.21	120883.7	346.4
1000	N	0.14	36214.6	<0.1	32393.4★	0.1	38289.8	73.1	33123.7	68.5	38904.3	95.8	25892.9★	776.73	25719.0	1695.2
		0.24	36750.4	<0.1	31462.5	0.2	61533.9	86.9	33744.7	79.7	38610.1	98.3	25570.6★	524.78	25138.9	1081.1
		0.34	36913.2	<0.1	30752.1★	0.3	121566.1	100.0	48329.9	95.9	38549.2	98.9	25714.2★	17.24	25345.0	1698.0
	V	0.14	8552.3	<0.1	8613.7	0.1	6071.9	11.6	6614.5★	14.1	6144.7	43.9	5869.0	1778.52	5890.4★	950.4
		0.24	4977.4	<0.1	5146.2	0.2	11493.0	64.6	12485.5	92.3	4528.1★	74.7	4281.2	109.08	4283.6★	90.1
		0.34	4656.5	<0.1	4693.1	0.4	17662.5	100.0	9742.4	100.0	6214.7	88.7	3974.5★	63.35	3974.3	264.2
	E	0.14	358550.4	<0.1	305034.7	0.1	362985.2	78.8	311510.5	75.5	350946.2	98.4	236226.6★	2582.61	233127.8	1711.2
		0.24	357735.8	<0.1	295099.4 ★	0.2	347041.7	88.6	326670.5	83.2	375956.1	99.4	239560.7★	451.8	238364.5	1277.3
		0.34	369051.9	<0.1	303712.9★	0.3	1232171.4	99.7	476998.4	97.3	370791.1	99.6	244685.7	24.31	246171.8★	1195.2

- Concerning the two types of graphs that were studied—that is, random graphs vs. random geometric graphs—essentially no differences can be observed in the relative behaviour of the algorithms. This means that the studied techniques are not affected by local structures that are present in graphs.

- Concerning the comparison between GREEDY1 and GREEDY2 it can be observed that, while GREEDY2 outperforms GREEDY1 when all problem instances are considered together—in the context of node-oriented graphs, GREEDY1 has a better average ranking than GREEDY2 (not statistically significant). Concerning computation time, both algorithms are by far the fastest ones in the comparison. Naturally, the computation time of GREEDY2 is slightly higher than that of GREEDY1.

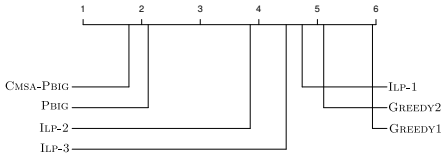
- Comparing the performance of the three ILP models, we can observe that ILP-2 generally achieves the best performance. In particular, for all considered subsets of instances (concerning density and weight generating scheme) ILP-2 outperforms ILP-1. This was to be expected, as the only difference in the two models is the elimination of a set of variables (also resulting in a change of a subset of the constraints). ILP-2 generally also outperforms ILP-3, with the exception of node-oriented graphs, for which ILP-3 achieves a better ranking than ILP-2. Finally, it can also be observed—when looking at the tables containing the numerical results—that all three ILP models are competitive with CMSA-PBIG and PBIG in the context of instances of 100 nodes.

- CMSA-PBIG and PBIG are clearly (and with statistical significance) the best-performing approaches in our set of compared approaches. This holds when considering all instances together, but also for all subsets of studied instances (see Figures 2 and 3). Concerning the comparison between the two, it can be observed that CMSA-PBIG has—for all considered subsets of instances—a better average ranking than PBIG. This difference is statistically signficiant when considering all instances together, and in the case of edge-oriented graphs. This relative performance between CMSA-PBIG and PBIG is of *general interest*, because it shows that by applying a metaheuristic in a framework such as CMSA, it is possible to improve the performance of the metaheuristic.

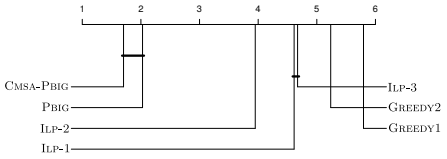
Summarizing, we can state that—in the context of the WID problem—the studied metaheuristics outperform the ILP-based approaches, which in turn outperform the greedy approaches. The ILP-based approaches can only compete with the metaheuristics in the case of instances of 100 nodes. Moreover, we would like to point out again the fact our results have shown it to be possible to improve the performance of a metaheuristic by repeatedly applying it to intelligently generated sub-instances of the tackled problem instances, as it is done in the CMSA framework.

6. Conclusions and Future Work

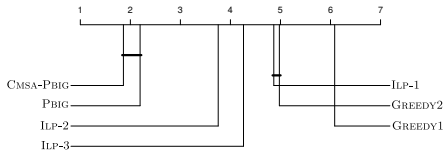
This paper has dealt with an NP-hard problem in graphs, the so-called weighted independent domination problem. We proposed three different integer linear program-



(a) All instances together.

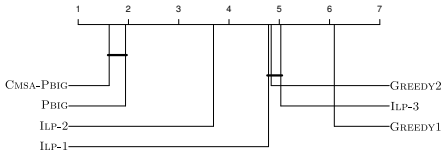


(b) Random graph instances (set RG).

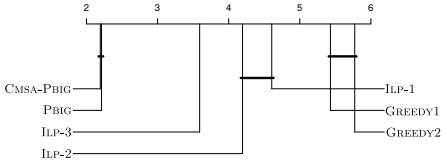


(c) Random geometric graph instances (set RGG).

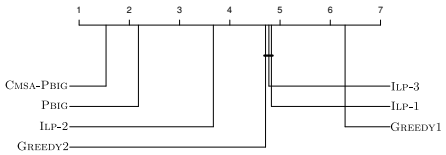
Figure 2: Criticial difference plots for all 540 problem instances together (a), the 270 instances of set RG (b), and the 270 instances of set RGG. The axis shows the average ranking of the seven considered techniques concerning the considered (sub)sets of instances. Horizontal bars connect techniques for which no statistical differences were found.



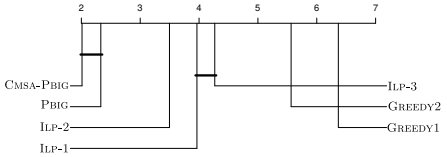
(a) All instances with *neutral* weighting scheme.



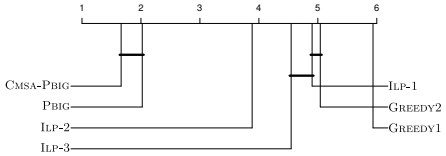
(b) All instances with *node-oriented* weighting scheme.



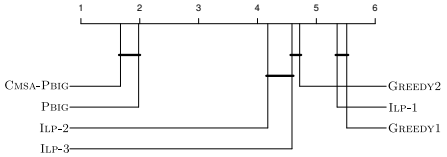
(c) All instances with *edge-oriented* weighting scheme.



(d) All low density instances.



(e) All medium density instances.



(f) All high density instances.

Figure 3: Criticial difference plots for six different subsets of problem instances concerning their weighting scheme and their density. The axis shows the average ranking of the seven considered techniques concerning the considered (sub)sets of instances. Horizontal bars connect techniques for which no statistical differences were found.

ming models for this problem. Additionally, two different greedy heuristics were presented. The first one of these heuristics is an adaptation of a known heuristic from a related problem to the weighted independent domination problem. This heuristic disregards the edge-weights during the solution construction process. The second heuristic was especially developed for the tackled problem. Finally, we presented a population-based iterated greedy algorithm which takes profit from the better one of the two greedy heuristics. In addition to a standalone application of the population-based iterated greedy algorithm, the algorithm was applied within the construct, merge, solve & adapt framework. The results have shown that the population-based iterated greedy algorithm applied within the before-mentioned framework is, with statistical significance, the best-performing approach in the comparison. This is especially interesting, because it shows that the performance of a standard metaheuristic may be improved by the application within a framework such as construct, merge, solve & adapt, which is based on reducing the size of the tackled problem instances.

In the near future we plan to investigate the application of other metaheuristics within the construct, merge, solve & adapt framework in the context of a diverse set of difficult combinatorial optimization problems.

Acknowledgements

This work was supported by project TIN2012-37930-C02-02 (Spanish Ministry for Economy and Competitiveness, FEDER funds from the European Union).

References

[1] S.-C. Chang, J.-J. Liu, Y.-L. Wang, The weighted independent domination problem in series-parallel graphs, in: Proceedings of ICS 2014 – The International Computer Symposium, Vol. 274 of Intelligent Systems and Applications, IOS Press, 2015, pp. 77–84.

[2] G. J. Chang, The weighted independent domination problem is NP-complete for chordal graphs, Discrete Applied Mathematics 143 (1) (2004) 351–352.

[3] C. Blum, P. Pinacho, M. López-Ibáñez, J. A. Lozano, Construct, merge, solve & adapt: A new general algorithm for combinatorial optimization, Computers & Operations Research 68 (2016) 75–88.

[4] P. Pinacho Davidson, C. Blum, J. A. Lozano, The weighted independent domination problem: ILP model and algorithmic approaches, in: B. Hu, M. López-Ibáñez (Eds.), Proceedings of EvoCOP 2017 – 17th European Conference on Evolutionary Computation in Combinatorial Optimisation, Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2017, in press.

[5] Y. Wang, R. Li, Y. Zhou, M. Yin, A path cost-based GRASP for minimum independent dominating set problem, Neural Computing and ApplicationsIn press.

[6] A. Potluri, A. Singh, Hybrid metaheuristic algorithms for minimum weight dominating set, Applied Soft Computing 13 (1) (2013) 76–88.

[7] S. N. Chaurasia, A. Singh, A hybrid evolutionary algorithm with guided mutation for minimum weight dominating set, Applied Intelligence 43 (3) (2015) 512–529.

[8] S. Bouamama, C. Blum, A hybrid algorithmic model for the minimum weight dominating set problem, Simulation Modelling Practice and Theory 64 (2016) 57–68.

[9] G. Lin, W. Zhu, M. M. Ali, An effective hybrid memetic algorithm for the minimum weight dominating set problem, IEEE Transactions on Evolutionary ComputationIn press.

[10] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research 177 (3) (2007) 2033–2049.

[11] L. Fanjul-Peyro, R. Ruiz, Iterated greedy local search methods for unrelated parallel machine scheduling, European Journal of Operational Research 207 (1) (2010) 55–69.

[12] S. Bouamama, C. Blum, A. Boukerram, A population-based iterated greedy algorithm for the minimum weight vertex cover problem, Applied Soft Computing 12 (6) (2012) 1632 – 1639.

[13] J. Porta, J. Parapar, R. Doallo, V. Barbosa, I. Santé, R. Crecente, C. Díaz, A population-based iterated greedy algorithm for the delimitation and zoning of rural settlements, Computers, Environment and Urban Systems 39 (2013) 12–26.

[14] C. Blum, Construct, merge, solve and adapt: Application to unbalanced minimum common string partition, in: M. J. Blesa, C. Blum, A. Cangelosi, V. Cutello, A. Di Nuovo, M. Pavone, E.-G. Talbi (Eds.), Proceedings of HM 2016 – 10th International Workshop on Hybrid Metaheuristics, Vol. 9668 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2016, pp. 17–31.

[15] C. Blum, M. J. Blesa, Construct, merge, solve & adapt: Application to the repetition-free longest common subsequence problem, in: F. Chicano, B. Hu (Eds.), Proceedings of EvoCOP 2016 – 16th European Conference on Evolutionary Computation in Combinatorial Optimization, Vol. 9595 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2016, pp. 46–57.

[16] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58.

[17] S. García, F. Herrera, An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons, Journal of Machine Learning Research 9 (2008) 2677 – 2694.

[18] B. Calvo, G. Santafé, scmamp: Statistical comparison of multiple algorithms in multiple problems, The R Journal 8 (1).