CHALMERS

Evaluating the in-the-middle heuristic in Graphical Model Discrete Optimization

Master's Thesis in Engineering Mathematics

Simon Sigurdhsson

Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden 2014

Master's Thesis 2014:?

REPORT NO. 2014:?

Evaluating the in-the-middle heuristic in Graphical Model Discrete Optimization

SIMON SIGURDHSSON

Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden 2014

Evaluating the in-the-middle heuristic in Graphical Model Discrete Optimization
SIMON SIGURDHSSON

Technical report no 2014:?
Department of Computer Science and Engineering
Chalmers University of Technology
SE–412 96 Göteborg
Sweden
Telephone +46 (0)31–772 1000

Department of Computer Science and Engineering
Göteborg, Sweden 2014

Evaluating the in-the-middle heuristic in Graphical Model Discrete Optimization
SIMON SIGURDHSSON
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

The field of graphical modelling has been very active in recent years, which is unsurprising given the large range of problems which may be described using graphical model. While many novel algorithms have been presented, there are still areas of the field in which improvements are possible. Recent reviews have uncovered strong links between the linear programming and graphical modelling fields, and it is therefore of interest to survey the possible application of linear programming methods to graphical model optimization.

The in-the-middle algorithm, an approximate solution method in linear programming which has seen extensive use in the industry, has recently been extended to max-sum problems. Graphical models are easily translated into max-sum problems, and the in-the-middle is therefore an ideal candidate in reformulating linear programming algorithms to the field of graphical model optimization.

This thesis presents an implementation of the in-the-middle algorithm applied to max-sum problems, which may be applied to general graphical model optimization instances. The implementation is benchmarked against three existing high-performance exact solvers in the field, using a problem set consisting of several hundred problems. Results indicate that the in-the-middle algorithm may have potential in the field of graphical model optimization, but that further research is required to make the algorithm competitive. Several avenues for further research on the algorithm are proposed.

Keywords: Combinatorial optimization, Constraint programming, Integer linear programming, Markov random fields, Max-SAT, Max-sum, Undirected graphical models, Wedelin heuristic, Weighted constraint satisfaction

Contents

Acronyms

CFN	cost function network.
CP	constraint programming.
CSP	constraint satisfaction problem.
CVPR	Computer Vision and Pattern Recognition.
DP	dynamic programming.
ILP	integer linear programming.
LP	linear programming.
MAP	maximum posterior.
max-SAT	maximum satisfiability.
MILP	mixed integer linear programming.
MRF	Markov random field.
SAT	Boolean satisfiability.
SCSP	semiring-based constraint satisfaction problem.
VCSP	valued constraint satisfaction problem.
WCSP	weighted constraint satisfaction problem.
WPMS	weighted partial max-SAT.

I

Introduction

Optimization is a very relevant topic in modern industry, with applications to planning, cost-efficiency and many other areas. Since many modern optimization problems are formulated through linear programming (LP) models, there are many mature and efficient algorithms aimed at solving such problems. However, some problems are most efficiently expressed through deterministic or probabilistic *graphical models* in which constraints and variables are expressed through a hypergraph. It is therefore of interest to examine the application of known LP methods to problems in the graphical model field.

This thesis will expand a known LP optimization algorithm which has seen extensive use in industry, the in-the-middle algorithm[1] (and its accompanying heuristic), to a wider set of problems in graphical model discrete optimization, specifically weighted constraint satisfaction problems (WCSPs) and related instances. This will be done by implementing a recently developed reformulation of the algorithm which applies to max-sum problems, with a suitable transformation of WCSP instances into max-sum instances. This implementation will then be benchmarked against solvers in the constraint programming (CP) and graphical model optimization fields.

The objective and purpose of the thesis is then to determine the usefulness of the max-sum in-the-middle algorithm within the field of graphical model discrete optimization, and to shed light on potential areas of improvement of the algorithm within that field. However, this thesis will not develop any significant variations of the algorithm, nor will it develop stronger theory on its functionality.

1. Originally proposed by Wedelin (1995).

1 BACKGROUND

The in-the-middle algorithm (Wedelin 1995) for LP problems can be interpreted as a coordinate descent algorithm, and with a simple heuristic it has been shown to solve integer programming problems with very good performance and quality in large scale applications. Recent work has extended this algorithm to also apply to max-sum problems (Grohe and Wedelin 2008; Wedelin 2013), but this application has not yet been benchmarked against other algorithms and software in that field. The primary aim of this thesis is therefore to implement and benchmark the in-the-middle algorithm applied to max-sum problems.

When formulated to solve max-sum problems, the algorithm is applicable to a host of problems which may be restated as such problems, including but not limited to (weighted, valued and regular) constraint satisfaction problems (CSPs), cost function networks (CFNs) and weighted partial max-SAT (WPMS) problems. Additionally, though a simple logarithmic transformation of the problem, Markov random field (MRF) problems may also be restated as max-sum problems. This makes the algorithm (including the approximate heuristic variant) interesting in fields ranging from scheduling, protein folding and *n*-queen puzzles to classification and pattern recognition.

The CSP and maximum satisfiability (max-SAT) communities have been very active in recent years, producing benchmarks of solvers in those fields at the *International Conference on Theory and Applications of Satisfiability Testing* (Argelich et al. 2011) and the *Annual Congress of the French Society of Operations Research and Decision Support* (Allouche et al. 2014). Using the results of such benchmarks, the in-the-middle algorithm on max-sum problems may be compared to existing algorithms in the mentioned fields, determining the interest of developing, analysing or specializing the algorithm further for such problems.

2 RELATED WORK

Since the purpose of this thesis is to extend an existing algorithm and benchmark it against known solvers, it is interesting to note previous work both related to the algorithm itself (either its LP formulation or the max-sum variant discussed in this thesis) as well as previous benchmarks and recent algorithms in the related fields.

Previous work on the algorithm itself, especially the LP variant, will be implemented and examined to determine if known modifications of the algorithm apply to the max-sum variant as well. Publications (primarily benchmarks and competitions) in the CSP field will be surveyed to collect suitable problem sets and to pick appropriate solvers to benchmark the algorithm against.

2.1 THE IN-THE-MIDDLE ALGORITHM

Although the in-the-middle algorithm has mostly been discussed by its principal author (Wedelin 1995; Grohe and Wedelin 2008; Wedelin 2013; Alefragis et al. 2000), there is some literature evaluating and improving upon the algorithm available.

Bastert, Hummel, and de Vries (2010) notes the practical relevance of the field of heuristics for integer linear programming (ILP) in recent years, and evaluates the in-the-middle algorithm applied to such problems. In addition to explicitly providing the generalizations mentioned by Wedelin (1995), they also introduce a “push” operation intended to improve the quality of the approximate solutions. Finally, they also compare the algorithm with commercial software, with fairly favourable results.

Ernst, Jiang, and Krishnamoorthy (2006) applied a method based on the Lagrangian relaxation used by the in-the-middle algorithm to the task allocation problem, with favourable results compared to the commercial CPLEX mixed integer linear programming (MILP) solver. Similarly, Mason (2001) applied a specialized variant of the in-the-middle algorithm to personnel scheduling, outperforming CPLEX on difficult commercial rostering problems.

2.2 MAX-SUM AND CONSTRAINT SATISFACTION PROBLEMS

Graphical models allow the modelling of a range of NP-hard optimization problems in a consistent manner (de Givry 2014), and recent articles on the subject expose strong connections between linear programming and graphical models (Werner 2007; Kolmogorov 2013). As such, there is reason to believe that the application of linear programming algorithms to graphical models may have practical relevance.

Although several specialized solvers — *e.g.* Toulbar2 (Allouche, de Givry, and Schiex 2010), WPM2 (Ansótegui et al. 2013), MaxHS (Davies and Bacchus 2013), Max-DPLL (Larrosa, Heras, and de Givry 2008) and several others — have been developed for graphical models in recent years, benchmarks still indicate that there are areas of graphical model optimization in which LP solvers perform better, and areas where no existing solvers excel.

Allouche et al. (2014) and de Givry (2014) evaluate several exact optimizers to graphical model optimization problems, and the results indicate that there may be several problem domains (most notably MRF, WPMS and Max-CSP) which could benefit from fast, approximate solvers. Additionally, several large problem sets from each domains are included in the benchmark, and these problem sets have also been made available online by the authors.[1]

1. `http://genoweb.toulouse.inra.fr/~degivry/evalgm` (Allouche et al. 2014, p. 7).

II

Theory

This chapter will introduce some of the theory behind max-sum problems, constraint satisfaction problems (CSPs) and related fields. There are significant similarities between these two problems, and a detailed translation of CSP (and similar problems) to equivalent max-sum formulations will be provided. This translation will be used to apply the in-the-middle algorithm to CSPs.

With this background given, the in-the-middle algorithm will be introduced. First, the original linear programming (LP) formulation will be briefly described. Then, the max-sum variant will be thoroughly explained,[1] and several extensions and modifications of the algorithm will be introduced. Finally, the theoretical framework surrounding the algorithm will be compared to theoretical results in a CSP context.

1 MAX-SUM PROBLEMS AND CONSTRAINT SATISFACTION

The formulation of the in-the-middle algorithm provided by Grohe and Wedelin (2008) is based on the definition of a max-sum problem. In order to apply this algorithm to general weighted constraint satisfaction problem (WCSP), it is necessary to provide a link between the max-sum problem and problem formulations within the graphical model optimization field, where WCSP is one of the more general formulations. This will be done by first introducing the max-sum problem along with some useful interpretations in other fields, followed by a similar introduction of the WCSP

1. The algorithm will be described in theory, with practical implementation issues and considerations being discussed in the next chapter.

family. Finally, a theoretical link between the two fields will be provided, along with an explicit method of translation between the formulations.

1.1 MAX-SUM PROBLEMS

The general max-sum problem is an NP-hard optimization problem with many applications in fields ranging from statistical physics to artificial intelligence and pattern recognition. Problems including set partitioning (Grohe and Wedelin 2008, p. 107), max-flow/min-cut and (as will be apparent later) several variants of constraint satisfaction may be restated as max-sum problems. Formally, the following definition (which is based on that of Grohe and Wedelin) may be used:

Definition 1 (Max-sum problem) *The max-sum problem is the optimization problem*

$$\max_x f(x) = \sum_{g_k \in G} g_k(x^k),$$

where $g_k(x^k) \in \mathbb{R}$ are distinct arbitrary functions over $x^k \subseteq x$.

The max-sum problem then has three distinct components by which it is defined:

- A finite set of *variables* $X = \{x_1, \ldots, x_n\}$. Let $X^k \subseteq X$ denote a specific subset of X, and let $x^k \in X^k$ and $x \in X$ be *assignments* of the variables.
- *Domains* of the variables, $D = \{D_1, \ldots, D_n\}$, such that $x_i \in D_i, \forall i$. Subsets $D^k \subseteq D$ may be defined analogously to the variable subsets.
- A finite set G of *cost functions*. Every cost function $g_k \in G$ is defined over a variable subset X^k, *i.e.* it is a map $g_k : D^k \mapsto \mathbb{R}_{\leq 0} \cup \{-\infty\}$.

The cost functions may additionally be divided into three kinds: those defined on the empty set (*constants*), those defined on singleton subsets $X^k = \{x_i\} \subseteq X$ (*variable components*) and those defined on larger subsets (*constraint components*) — this is the *component model* introduced by Grohe and Wedelin (2008, p. 98).

One should also note that the codomains of g_k need not be restricted to $\mathbb{R}_{\leq 0} \cup \{-\infty\}$, but for the purposes of this thesis that is the chosen output. This allows solutions x to the max-sum problem to be ordered by their cost, where the cost is defined as $\text{cost}\,(x) = f(x) = \sum_k g_k(x^k)$, and additionally allows the definition of *infeasible* solutions to the max-sum problem as those for which $\text{cost}\,(x) = -\infty$. This will be useful in the translation between WCSPs and max-sum problems.

There are several algorithms available for solving max-sum problems, with Werner (2007) mentioning the *augmented DAG algorithm*, the *max-sum diffusion algorithm* and a LP relaxation method. In addition to those direct methods, the relation to CSP provides many more (which will be mentioned later), and algorithms such as *belief propagation* and *message passing* are also applicable to some max-sum problems (in particular, max-sum problems without loops).

Markov Random Fields

A restricted variant of the max-sum problem called the *binary max-sum labelling problem* has direct applications to artificial intelligence and pattern recognition, where the problem is known as computing the maximum posterior (MAP) configuration of Markov random fields (p. 1165).

1.2 WEIGHTED CONSTRAINT SATISFACTION PROBLEMS

CSPs are very general decision problems, defined through a set of objects which must satisfy a set of constraints. Many problems in artificial intelligence and operations research (including planning and resource allocation) may be stated as CSPs, as well as several academic problems such as Boolean satisfiability (SAT), queens puzzles and map colouring. One of the corresponding combinatorial optimization problems,[1] the WCSP,[2] additionally introduces *weights* on the constraints, and classify these as *hard* (must be satisfied) or *soft*. All CSPs may of course be restated as WCSPs with only hard constraints, while the relaxed objective of the combinatorial optimization approach allows even over-constrained CSPs to be "solved". Additionally, many problems in complexity theory such as maximum satisfiability (max-SAT), max-clique, max-cut and minimal vertex cover may be modelled using WCSPs (Meseguer, Rossi, and Schiex 2006, p. 315). Due to these facts, WCSPs may be regarded as more interesting than CSPs, especially in an optimization context.

There are several ways to formally define a WCSP, but the one used here closely matches the definition of a max-sum problem, which simplifies the formal translation between the two. The definition is based on that presented by Meseguer, Rossi, and Schiex (2006), which defines WCSP (p. 284) in terms of a regular CSP (p. 281):

Definition 2 (Constraint satisfaction problem) *A CSP is a decision problem consisting of three parts:*

- *A finite set of* variables $X = \{x_1, \ldots, x_n\}$. *Let* $V \subseteq X$ *denote a specific subset of* X.
- Domains *of the variables,* $D = \{D_1, \ldots, D_n\}$, *such that* $x_i \in D_i, \forall i$. *Subsets* $D^V \subseteq D$ *may be defined analogously to the variable subsets.*
- *A finite set* C *of* constraints $R_V \in C$ *defined by a* relation R *defined on a subset of variables* $V \subseteq X$, *which specify the assignments of* V *allowed by the constraint.*

The problem is to find an assignment t *which is allowed by all constraints* $R_V \in C$.

1. Both the valued constraint satisfaction problem (VCSP) and semiring-based constraint satisfaction problem (SCSP) frameworks may be seen as the corresponding optimization problems (Meseguer, Rossi, and Schiex 2006; Bistarelli et al. 1999), but WCSPs may be described using either. 2. In some literature these problems are called cost function networks (CFNs), but the definitions are in essence equivalent.

The reformulation as an optimization problem mainly concerns the introduction of *weights*, and a reformulation of the objective:

Definition 3 (Weighted constraint satisfaction problem) *A WCSP (denoted by Meseguer, Rossi, and Schiex (2006, p. 284) as a* k-weighted constraint network*) is a 4-tuple $\langle X, D, C, k \rangle$, where X and D are variables and domains as in definition 2, C is a set of* weighted *constraints and k is an upper bound. A weighted constraint $f_V \in C$ maps a subset V of variables to the set* $[0,k]$, i.e. $f_V : D^V \mapsto [0,k]$. *The* cost *of an assignment t is defined as the (bounded) sum of all f_V, and the optimization problem amounts to*

$$\min_t \operatorname{cost}(t) = \sum_{f_V \in C} f_V(t^V).$$

The attentive reader will notice the similarity between definition 3 and definition 1. Using this definition, *feasible* solutions are assignments t for which $\operatorname{cost}(t) < k$. One may also make a distinction between *hard* constraints ($f_V(t^V) = k$ for some assignment(s) t^V) and *soft* constraints.

Max-CSP and (weighted) max-SAT
The special case of WCSP where all constraints have either unit or zero cost (*i.e.* $f_V : D^V \mapsto 0, 1$, regardless of choice of k) is normally referred to as max-CSP (p. 284). Here, the objective value is exactly the number of violated clauses, and as such it is the most natural formulation of existing CSP instances as optimization problems.

The special case where all constraints are clauses of a Boolean formula (and weights are unrestricted) yields the weighted partial max-SAT (WPMS) problem (de Givry 2014, p. 2). If the weights are restricted to unit or zero costs as above, the problem becomes the well-known max-SAT problem (Meseguer, Rossi, and Schiex 2006, p. 284).

This makes explicit the fact that many real-world and academic problems in satisfiability and operations research may be restated as WCSP (and therefore max-sum) problems, and consequently that the in-the-middle algorithm may present a viable alternative to solving these problems using LP formulations (Ansótegui and Gabàs 2013; Davies and Bacchus 2013) or special algorithms (Ansótegui et al. 2013; Larrosa, Heras, and de Givry 2008).

1.3 TRANSLATING WCSP TO MAX-SUM

The translation from WCSP to max-sum, when using the definitions given above, is fairly straight-forward. In addition to the superficial similarity between definitions 1 and 3, the two problems have deep connections and are in a sense equivalent. When regarding the WCSP formulation as an instance of SCSP (Meseguer, Rossi, and Schiex 2006, p. 285 sq.), the WCSP has an associated ordered semiring $\langle [0,k], +^k, \min, \leq \rangle$ (p. 290). Werner (2007, p. 1167) noted that the max-sum problem is also associated with a similar ordered semiring structure $\langle(-\infty,\infty),+,\max,\geq\rangle$ (although in our case the set is $\mathbb{R}_{\leq 0} \cup \{-\infty\}$). Werner additionally provides a connection between max-sum and the regular CSP through a labelling problem, of which they are both special cases.

In fact, it seems that the only differences between the two problems as stated is the definition of the set included in the semiring — while the WCSP from definition 3 considers a domain $\{0,\dots,k\}$, the max-sum problem according to definition 1 has domain $\mathbb{R}_{\leq 0} \cup \{-\infty\}$ — and the fact that one is a minimization problem while the other is a maximization problem. However, by transforming the weighted constraints of the WCSP problem in such a way that the ordering is preserved but reversed (*i.e.* by negating all costs), and additionally setting $k = -\infty$ (and changing the costs of all hard constraints accordingly), a formulation which has a semiring $\langle\mathbb{R}_{\leq 0} \cup \{-\infty\},+,\max,\geq\rangle$ is obtained, which is exactly what the max-sum problem has. From there, it is only a matter of translating the constraints of the WCSP into corresponding cost functions in the max-sum formulation.

The translation from WCSP to the equivalent max-sum problem may be summarized by a few steps:

1. Variable sets and domains are kept in the translation; the sets D, X and X^V in definitions 2 and 3 correspond directly to the sets D, X and X^k of definition 1. Additionally, the subsets V of the WCSP problem correspond to the subsets k in definition 1.

2. The weighted constraints f_V of definition 3 are replaced by new constraints f'_V defined as
$$f'_V(t) = \begin{cases} -\infty & \text{if } f_V(t) < k \\ -f_V(t) & \text{otherwise} \end{cases}, \quad \forall t.$$

3. The cost functions g_k of the max-sum formulation are constructed from the new constraints, *i.e.* $g_k = f'_V$, as appropriate.

2 THE IN-THE-MIDDLE ALGORITHM

This section will present the in-the-middle algorithm for max-sum problems, and the framework of *variable components*, *constraint components* and *constraint updates* employed by the algorithm. It will provide theoretical results on the feasibility and optimality guarantees provided by the algorithm, and will present two variations of the update procedure used in the algorithm. Previous improvements to the LP formulation will also be applied to the max-sum version. First, however, the original LP algorithm will be reviewed.

2.1 Original LP formulation

To provide context for the max-sum formulation of the in-the-middle algorithm and heuristic, the original algorithm as applied to LP problems will be briefly described, and explicit pseudo-code for the algorithm will be presented. Full descriptions of the LP variant of the in-the-middle algorithm are available from Wedelin (1995) and Bastert, Hummel, and de Vries (2010).

The algorithm solves an integer linear programming (ILP) problem

$$\begin{aligned} &\min c^\top x \\ \text{s.t. } &Ax = b, \\ &x \in \{0,1\}^n, \end{aligned} \tag{1}$$

where $A \in \{-1,0,1\}^{m\times n}$ and $b \in \mathbb{N}^m$ (Wedelin (1995) considers the case $A \in \{0,1\}^{m\times n}$). The idea is to exploit the Lagrangian relaxation of the problem,

$$\begin{aligned} &\min c^\top x - y^\top(Ax - b) \\ \text{s.t. } &x \in \{0,1\}^n, \end{aligned} \tag{2}$$

where y are the Lagrangian multipliers — once a value of y has been fixed to $\hat{y}$ it is easy to find the minimum of this relaxation. Assuming all reduced costs $\bar{c} = (c^\top - \hat{y}^\top A)$ are non-zero one may then express an optimal solution $\hat{x}$ to the original problem, with $\hat{x}_i = 1$ for $\bar{c}_i < 0$ and $\hat{x}_i = 0$ for $\bar{c}_i > 0$.

The goal of the algorithm is to manipulate y to minimize (2) while maintaining feasibility for (1). The basic idea is to, for every single constraint j of the problem, find the elements in $\bar{c}$ that correspond to the variables of that constraint (denoted by $\bar{c}^j$) and subtract the average of two *critical values* r^+ and r^-. The critical values are chosen so that at most b_j of the values in $\bar{c}^j$ are strictly positive. If no sign changes occur after visiting each constraint once, the algorithm has converged and a feasible (and possibly optimal, if $\bar{c} \neq 0$) solution has been found. Algorithm 1 provides complete pseudo-code for this algorithm, and Wedelin (1995, 2013) provides further theoretical results.

The attentive reader will note that the method described by algorithm 1 may produce situations in which $\bar{c}_i = 0$ for some i, terminating without an integer solution to the original problem. To address this issue, the algorithm is transformed from being exact to being approximate by introducing a heuristic. The purpose of the heuristic is to "nudge" the reduced costs and move them away from 0, while still distorting them as little as possible. This will force an integer solution through what may be interpreted as coordinate search. A parameter κ is introduced to govern the heuristic: for $\kappa = 0$ there is no approximation and for $\kappa = 1$ there is maximal approximation.

The idea of the heuristic is then to add a small positive value to positive elements $\bar{c}_i$, and a small negative value to negative ones. To this end, the empty line 11 of

Algorithm 1: The in-the-middle algorithm without its heuristic. Counting the sign changes may be done efficiently in the final assignment to $\bar{c}$.

```
Input: $A \in \{-1, 0, 1\}^{m \times n}$, $b \in \mathbb{N}^m$, $c \in \mathbb{R}^n$
Output: Optimal solution $\hat{x} \in \{0, 1\}^n$ to eq. (1) or $\emptyset$
$\bar{c} \leftarrow c$
$s^j \leftarrow 0$
repeat
    for $j = 1, \dots, m$ do
        $r \leftarrow \bar{c}^j + s^j$
        $d \leftarrow r \cdot a_{\cdot,j}^{-1}$
        $r^+ \leftarrow b_j$th largest element of $d$
        $r^- \leftarrow (b_j + 1)$th largest element of $d$
        $\Delta y \leftarrow (r_j^+ + r_j^-)/2$
        $s^j \leftarrow \Delta y \cdot a_{\cdot,j}$
                                                  ▶ Intentionally left blank
        $\bar{c} \leftarrow r - s^j$
    end
until no sign changes in $\bar{c}$
if $\bar{c} \neq 0$ then
    for $i = 1, \dots, n$ do
        if $\bar{c}_i \leq 0$ then $\hat{x}_i \leftarrow 1$ else $\hat{x}_i \leftarrow 0$
    end
else
    return $\emptyset$
end
```

algorithm 1 is replaced, introducing an assignment

$$s_i^j \leftarrow s_i^j \pm \left(\frac{\kappa}{1-\kappa}(r_j^+ - r_j^-) + \delta\right)$$

where addition is used when $r_i - s_i^j \geq 0$ and subtraction otherwise. This assignment is (as implied) applied to all elements of s^j. The (small) parameter $\delta > 0$ ensures that elements of $\bar{c}$ are kept non-zero at all times.

Note that Bastert, Hummel, and de Vries (2010, p. 97) provide a different but equivalent formulation of algorithm 1 with the approximate heuristic.

2.2 MAX-SUM FORMULATION

Working from the LP formulation of the in-the-middle algorithm, Grohe and Wedelin (2008) introduce a similar algorithm for max-sum problems (which they call *cost propagation*). The formulation is examined further by Wedelin (2013, p. 11 sqq.). This section will describe the max-sum formulation of the in-the-middle algorithm, reference some theoretical results, and provide explicit pseudo-code for the algorithm.

The algorithm considers a max-sum problem

$$\max_x \sum_{g_k \in C} g_k(x^k), \tag{3}$$

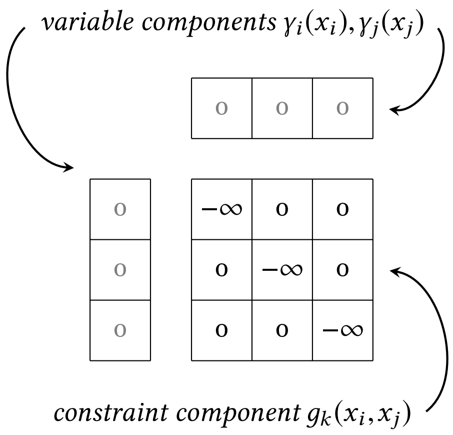
in accordance with definition 1. Recalling the *component model* (page 6), the functions g_k may be divided into *variable* and *constraint* components — for instance, the problem

$$\max_x g_1(x_1) + g_2(x_2) + g_3(x_3) + g_4(x_1, x_2) + g_5(x_2, x_3)$$

has variable components g_1, g_2, g_3 and constraint components g_4, g_5. While it is possible to implement the algorithm so that all functions g_k are translated into constraint components, this division has computational advantages in that the variable components may be represented implicitly in a cost variable. Therefore, to keep the variable and constraint components apart, the former will be denoted by $\gamma_1, \ldots, \gamma_n$ and the latter by $g_1, \ldots, g_m$. The components are then $\gamma_i : D_i \mapsto \mathbb{R}_{\leq 0} \cup \{-\infty\}$ and $g_k : D^k \mapsto \mathbb{R}_{\leq 0} \cup \{-\infty\}$, where each g_k is associated with a subset of all variable components as well, as shown by fig. 1 (this subset will be called γ^j).

With this in mind, the basic framework of the algorithm may be introduced (algorithm 2), which intentionally is very similar to that of the LP formulation. Note especially the *modified* variable components $\hat{\gamma}_i$, which roughly correspond to the reduced costs $\bar{c}$ of the LP formulation (in the same way that the actual variable components γ_i roughly correspond to the actual cost vector c).

The constraint updates used in the algorithm will now be described and theoretically related to the updates used in the LP formulation. In addition to this, some

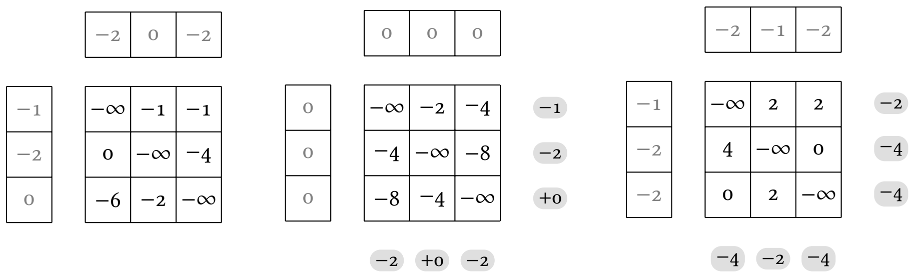


variable components $\gamma_i(x_i), \gamma_j(x_j)$
0 0 0
0 0 0
$-\infty$ 0 0
0 $-\infty$ 0
0 0 $-\infty$
constraint component $g_k(x_i, x_j)$

Figure 1: *A single constraint of two variables x_i, x_j with $|D| = 3$, and consequently a cost table $g_k(x_i, x_j)$ with nine values. This particular constraint is a hard* all-different *constraint $x_i \neq x_j$.*

Algorithm 2: The basic framework of the max-sum in-the-middle algorithm. If the constraint updates are non-conflicting, the output is an optimal solution to eq. (3).

```
   Input: Variable components $\gamma_1, \ldots, \gamma_n$, constraint components $g_1, \ldots, g_m$
   Output: Feasible solution $\hat{x} \in X$ to eq. (3) or $\emptyset$
 1 for $i = 1, \ldots, n$ do $\bar{\gamma}_i \leftarrow \gamma_i$
 2 for $j = 1, \ldots, m$ do $s_i^j \leftarrow 0$
 3 repeat
 4     for $j = 1, \ldots, m$ do
 5         $\overline{\gamma^j}, s^j \leftarrow$ UpdateConstraint($g_j$, $\bar{\gamma}^j$, $s^j$)
 6     end
 7 until $\arg\max_{x_i} \bar{\gamma}_i(x_i)$ did not change for any $i$
 8 if $\arg\max_{x_i} \bar{\gamma}_i(x_i)$ is unique for all $i$ then
 9     foreach $i = 1, \ldots, n$ do $\hat{x}_i \leftarrow \arg\max_{x_i} \bar{\gamma}_i(x_i)$
10 else
11     return $\emptyset$
12 end
```



***(a)** Initial constraint* ***(b)** The "move in" operation* ***(c)** The "move out" operation*

Figure 2: *An illustration of a constraint update (in this case a dynamic programming (DP) update). Note that this particular update is* conflicting, *which results in a lost optimality guarantee.*

important theoretical results will be mentioned. Finally, two specific updates will be described.

The basic idea of the constraint update, as with the LP formulation, is to modify the variable components in an invariant manner to uniquely identify the (optimal) solution given the constraint component. This is done by considering only the subproblem consisting of the constraint component g_j along with its associated variable components γ^j and performing a local optimization. While Grohe and Wedelin (2008, p. 100 sq.) present a framework for updates defining a few sought-after properties, only what they call *consistent* and *locally optimal* updates are considered here. These guarantee that any solution found by algorithm 2 will be feasible. Some updates (in particular the fractional DP update described later, for some parameter values) are additionally *non-conflicting*, which guarantees that any feasible solution found is also optimal.

Each update, as illustrated by fig. 2, first *moves in* the costs of the variable components into the constraint component (fig. 2b, *i.e.* a temporary constraint component $h_k(x^k) = g_k(x^k) + \sum_{\gamma_j \in \gamma^k} \gamma_j(x_j)$ is constructed). Then, costs from the temporary constraint component h_k are moved out again (fig. 2c). It is mainly this last step that varies between different updates — it is desirable to move out as much as possible (this improves the convergence rate of the algorithm), but if one moves out too much the *non-conflicting* property is lost and optimality is no longer guaranteed (Grohe and Wedelin 2008, p. 105; Wedelin 2013, p. 15).

The DP constraint update

The first update considered is the DP update (algorithm 3 and fig. 2). It is a fairly unsophisticated update, which extracts the max-marginals along each dimension of

Algorithm 3: The DP constraint update.

```
1 Function UpdateConstraint($g_k, \bar{\gamma}^k, s^k$)
      Data: A constraint component $g_k$, variable components $\bar{\gamma}^k$ and offsets $s^k$
      Result: Updated variable components $\bar{\gamma}'^k$ and offsets $s'^k$
2     forall the $\bar{\gamma}_j \in \bar{\gamma}^k$ do $r_j^k \leftarrow \bar{\gamma}_j - s_j^k$
3     $h_k(x^k) \leftarrow g_k(x^k) + \sum_{r_j \in r^k} r_j(x_j)$
4     forall the $\bar{\gamma}_j \in \bar{\gamma}^k$ do $\bar{\gamma}_j'(x_j) \leftarrow \max_{x \in x^k \setminus x_j} h(x)$
5     forall the $\bar{\gamma}_j' \in \bar{\gamma}'^k$ do $s_j'^k \leftarrow \bar{\gamma}_j' - r_j^k$
6     return $\bar{\gamma}'^k, s'^k$
7 end
```

Algorithm 4: The fractional DP constraint update. Note that the only difference between this and algorithm 3 is on line 4.

```
1 Function UpdateConstraint($g_k, \bar{\gamma}^k, s^k$)
      Data: A constraint component $g_k$, variable components $\bar{\gamma}^k$ and offsets $s^k$
      Result: Updated variable components $\bar{\gamma}'^k$ and offsets $s'^k$
2     forall the $\bar{\gamma}_j \in \bar{\gamma}^k$ do $r_j^k \leftarrow \bar{\gamma}_j - s_j^k$
3     $h_k(x^k) \leftarrow g_k(x^k) + \sum_{r_j \in r^k} r_j(x_j)$
4     forall the $\bar{\gamma}_j \in \bar{\gamma}^k$ do $\bar{\gamma}_j'(x_j) \leftarrow \alpha \max_{x \in x^k \setminus x_j} h(x)$
5     forall the $\bar{\gamma}_j' \in \bar{\gamma}'^k$ do $s_j'^k \leftarrow \bar{\gamma}_j' - r_j^k$
6     return $\bar{\gamma}'^k, s'^k$
7 end
```

the constraint component g_k and moves this amount into the corresponding variable component $\gamma_i(x_i)$ (*i.e.* the new variable component value becomes $\bar{\gamma}_i'(x_i) = \max_{x \in x^k \setminus x_i} h(x)$ for every variable component γ_i and value $x_i \in D_i$). While this should ensure quick convergence since it in effect moves out as much as possible, the update isn't non-conflicting (Grohe and Wedelin 2008, p. 105), which means optimality cannot be guaranteed.

The fractional DP constraint update

Since the regular DP update cannot guarantee optimality, it is interesting to see if it may be modified to provide such a guarantee. Grohe and Wedelin (2008, p. 105 sqq.) provide some theory useful in the construction of non-conflicting updates, in particular a weak upper bound on the amount possible to move out. They also introduce the *fractional* DP update (algorithm 4), which is very similar to the regular DP update

— instead of moving out the max-marginals of the constraint component, a fraction α of the max-marginal is moved out (*i.e.* the new variable component value becomes $\bar{\gamma}_i'(x_i) = \alpha \max_{x \in x^k \setminus x_i} h(x)$).

It is immediately obvious that $\alpha = 1$ corresponds to the regular DP update, and as such this may be taken as an upper limit of the parameter. Additionally, Grohe and Wedelin (2008) provide an upper bound for α that guarantee non-conflicting updates (p. 107). This upper limit is $\alpha = 1/n$, where n is the number of variables associated with the constraint component (*e.g.* $n = 2$ for the constraint component in fig. 2). Interesting values of α for parameter sweep schemes are thus $\alpha \in [n^{-1}, 1]$.

2.3 EXTENSIONS AND IMPROVEMENTS

Previous work on the LP formulation of the in-the-middle algorithm has yielded useful improvements in performance, and it is therefore interesting to apply these variants to the max-sum variant as well. Two such improvements are presented.

First, a modification of the LP algorithm introduced by Bastert, Hummel, and de Vries (2010) is considered. Then, an important addition to the algorithm designed to break ties in variables is introduced.

The "push" operation

In their paper examining the in-the-middle algorithm, Bastert, Hummel, and de Vries (2010) introduce a "push" operation intended to improve the quality of any non-optimal solutions found by the approximate algorithm (p. 99 sq.). The operation is aimed at improving the solution by modifying the reduced costs $\bar{c}$ to make them "more similar" to c, contending that the algorithm in fact optimizes with respect to the reduced costs (to which the approximate variant applies non-invariant changes). This is done by setting $\bar{c} \leftarrow \bar{c} + \rho c$ (with $\rho > 0$) and reducing the parameter κ of the approximate LP formulation (algorithm 1) by a set factor when a feasible solution is found (and $\kappa > 0$). Additionally, a flag is set that makes the algorithm treat all constraints as violated (p. 100).

Given the favourable results of Bastert, Hummel, and de Vries (2010), it is interesting to benchmark the operation for the max-sum algorithm as well. Luckily, the "push" operation is easily translated to apply to the max-sum formulation of the in-the-middle algorithm. After encountering a feasible solution (and assuming the optimality guarantee does not apply), the modified variable components $\bar{\gamma}_i$ are updated as in the LP formulation (*i.e.* $\bar{\gamma}_i \leftarrow \bar{\gamma}_i + \rho\gamma_i$), and the parameter α is reduced in a similar way.

Tie-breaking using noise

In some problems — especially those with few hard constraints — there may be several solutions in a neighbourhood which have the same objective value. Such situations may result in the modified variables components $\bar{\gamma}_j$ being tied for some variables j, which means the algorithm cannot determine a feasible solution (since the corresponding variable value $\hat{x}_j$ is ambiguous). To address this issue, a stochastic tie-breaking mechanism is introduced.

The tie-breaking mechanism considers the modified variable components $\bar{\gamma}_j$, in which $\arg\max_{x_j} \overline{\gamma_j}(x_j)$ is non-unique if variable x_j is ambiguous. Moving through all the variable components $\bar{\gamma}_j$, any element whose value "too close" (determined by a threshold value ϵ) to the maximum value $M = \max_{x_j} \overline{\gamma_j}(x_j)$ is modified by adding uniformly distributed random noise of magnitude ζ, *i.e.*

$$\bar{\gamma}'_j(x_j) = \bar{\gamma}_j(x_j) + \begin{cases} u \operatorname{sgn}\left(\bar{\gamma}_j(x_j) - M\right) & \text{if } |\bar{\gamma}_j(x_j) - M| \leq \epsilon \\ 0 & \text{otherwise} \end{cases}, \quad \forall x_j \in D_j,$$

where $u \in \mathcal{U}(-\zeta, \zeta)$ is a random variable taken from a uniform distribution. If noise has been added in a previous iteration, that noise is removed from the variable component first.

2.4 INTERPRETATION IN A WCSP CONTEXT

The framework commonly employed in WCSP optimization focuses on providing equivalence-preserving transformations of the problem graph, in order to obtain good upper and lower bounds which are then used in branch-and-bound strategies. A fairly thorough presentation of the *arc consistency* notions used in this framework is given by Cooper et al. (2010).

The notion of *generalized arc consistency* (p. 7) is in fact used by the algorithm, corresponding to the *consistent* updates introduced by (Grohe and Wedelin 2008, p. 101). This type of arc consistency is normally not utilized in WCSP theory, since it is rather weak. Stronger notions, such as *(full) arc consistency* and *existential arc consistency* (de Givry et al. 2005), are centred around providing very good bound on the problem by projecting costs away from constraints and moving them into the variable components or the nullary constraint, which then immediately provides an easily accessible lower bound on the optimal solution.

Algorithms enforcing full arc consistency are in fact very similar to the *non-conflicting* updates proposed by Grohe and Wedelin (2008). De Givry et al. (2005, p. 85 sq.) present an algorithm enforcing this property, and the general structure of this algorithm is very similar to the DP update described above. First, the costs in one of the variable components are moved into the constraint component. Then (as in the DP update), the max-marginals are calculated and "moved out" (*projected*) onto the other variable component. The result is one variable component with mostly zero-value costs, and one variable component with (if possible) no zero-value costs. This fact is then used when enforcing another property, *node consistency* (Cooper et al.

2010, p. 7), which in effect moves redundant costs from the variable components to the nullary constraint.

Stronger constraint component updates
Although the theory behind the in-the-middle algorithm isn't aimed at providing good upper and lower bounds, instead being concerned with providing an optimality guarantee, there is no reason to discard existing theory in the WCSP field. It appears feasible to implement some of the consistency-enforcing algorithms of de Givry et al. (2005) as constraint component updates, and enforcing node consistency could be done once every few iterations independently of the updates (or in a preprocessing stage). This could provide lower bounds which could be employed as a secondary optimality guarantee, which could be used to measure the optimality gap in situations where potentially conflicting updates are used.

It may therefore be possible or even desirable to extend notions of stronger arc consistency to the in-the-middle algorithm.

III

Method

In this chapter, specific implementation details relating to the algorithm as used in the benchmark will be presented. This includes considerations which are of great importance when implementing efficient variants of the algorithm, including memory considerations as well as computational ones. The implementation details are intended to facilitate independent implementations of the algorithm.

Additionally, the method used in the benchmark will be explained (including hardware used). The problem sets used in the benchmark will be introduced, including a short background and some characteristics of each set. Finally, the other solvers used in the benchmark will be briefly introduced including a short review of their underlying algorithms.

1 IMPLEMENTATION

The algorithm — including reading routines for the WCSP file format (Otten 2008), parameter sweep strategy and timing facilities — was implemented in C++11. The implementation is partly based on an existing framework for the original LP in-the-middle algorithm. The code was compiled using version 5.1 of the LLVM compiler, with all safe optimizations enabled (*i.e.* `-O3`).

There are several implementation details which are highly relevant to the performance of the algorithm, and this section will explore such details in depth. In particular, the choice of data structure for constraint component data as well as implementation of constraint updates makes significant impact on the runtime of the algorithm.

Algorithm 5: Fast implementation of the fractional DP update described in algorithm 3.

```
Function UpdateConstraint($g_k$, $\bar{\gamma}^k$, $s^k$)
    Data: A constraint component $g_k$, (pointers to) variable components $\bar{\gamma}^k$ and offsets $s^k$
    Result: Updated variable components $\bar{\gamma}^k$ and offsets $s^k$, number of sign changes $c$
    $c \leftarrow 0$ forall the $\bar{\gamma}_j \in \bar{\gamma}^k$ do
        $r_j^k \leftarrow \bar{\gamma}_j - s_j^k$
        $s_j^k \leftarrow -\infty$                                  ▶ Allows omitting $g_k(x^k) = -\infty$
        $\gamma_j^+, \gamma_j^- \leftarrow -\infty$                 ▶ Used in transforming $\bar{\gamma}_j$
    end
    forall the pairs $\langle x^k, g_k(x^k)\rangle$ do
        $v \leftarrow \sum_{x_i^k \in x^k} \gamma_i(x_i^k)$
        $v \leftarrow \alpha(v + g_k(x^k))$
        forall the $x_j \in x^k$ do
            $s_j^k \leftarrow \max\left\{s_j^k, v\right\}$
            if $v > \gamma_j^+$ then
                $\gamma_j^- \leftarrow \gamma_j^+$
                $\gamma_j^+ \leftarrow v$
            else if $v > \gamma_j^-$ then
                $\gamma_j^- \leftarrow v$
            end
        end
    end
    forall the $\bar{\gamma}_j \in \bar{\gamma}^k$ do $\bar{\gamma}_j \leftarrow \bar{\gamma}_j - (\gamma_j^+ + \gamma_j^-)/2$      ▶ Transforms $\bar{\gamma}_j$

    forall the $\bar{\gamma}_j \in \bar{\gamma}^k$ do
        Increment $c$ by $\#\{\bar{\gamma}_j : \bar{\gamma}_j \cdot s_j^k \le 0\}$   ▶ Counts number of variable components changed by this update
        $\bar{\gamma}_j \leftarrow s_j^k$
        $s_j^k \leftarrow s_j^k - r_j^k$
    end
    return $\bar{\gamma}^k, s^k, c$
end
```

A parameter sweep strategy used in conjunction with the algorithm, which controls the α parameter of the fractional DP update, will also be introduced and explained in further detail.

1.1 CONSTRAINT COMPONENT DESIGN DECISIONS

Several design decisions in the implementation of the constraint components have significant impact on the performance of the constraint component updates. These design decisions mostly relate to the data structure representing costs inside the constraint component, and the main concern in selecting this data structure is quick access in the update loop. Since the constraint component is kept constant through all iterations, and the temporary cost table h can be made implicit, this is the only major concern.

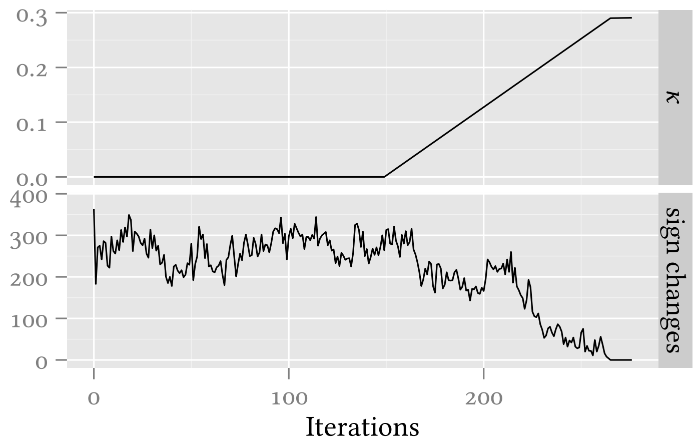
Another implementation detail to note is the storage of the (modified) variable components $\bar{\gamma}_i$. Storing these sequentially in memory is a good choice, but care needs to be taken when ordering the variable components in memory — due to CPU cache characteristics, storing variable components used in the same constraint component next to each other is highly beneficial. However, the implementation used here does not reorder variable components in this manner, instead storing them in the order they have been defined in the problem input.

A good data structure for the constraint components g_k is a vector with a sparse representation of the table costs (omitting infeasible values, *i.e.* $g_k(x^k) = -\infty$). These values may be implicitly represented by proper initialization of variables in the constraint update. The vector representation may be described as a list of pairs $\langle x^k, g_k(x^k)\rangle$, where the (fixed) variable values x^k additionally may be used to access corresponding variable component values $\bar{\gamma}_i(x_i^k)$. In the actual implementation, x_i^k are represented as pointers to the values $\bar{\gamma}_i(x_i^k)$.

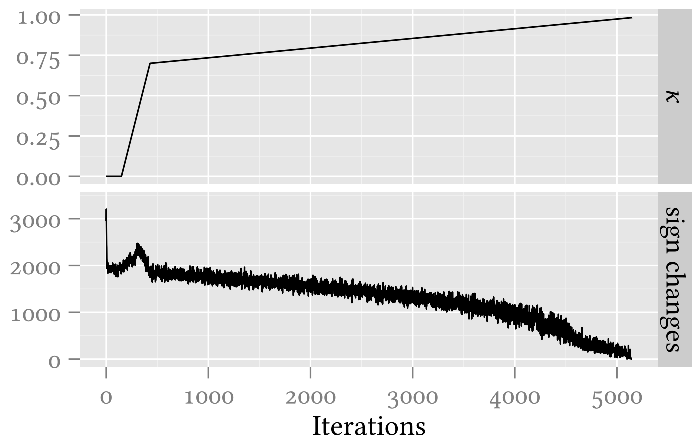
Algorithm 5 highlights these implementation details, and also shows the use of an invariant transformation of the variable components $\bar{\gamma}_i(x_i^k)$ that allows the use of existing code to detect solution changes by counting sign changes.

1.2 PARAMETER SWEEP STRATEGY

As explained earlier, the fractional DP update of constraint components depends on a parameter α, which dictates the amount "moved out" of the constraint component. Choosing this parameter is difficult, but some of the results discussed earlier may be used to create a strategy for a parameter sweep. Knowing that values $\alpha \leq n^{-1}$ (where n is the arity of a constraint) guarantee that any solution found is optimal, n^{-1} may be chosen as a lower limit for the parameter. A reasonable upper limit for the parameter is $\alpha = 1$, which in essence corresponds to the regular, non-fractional DP update.



(a) *A max-CSP problem from the "Composed" set.*



(b) *A Markov random field (MRF) problem from the "Object Detection" set.*

Figure 3: *Influence of the κ parameter for two different problems using the fractional update, with noise applied to resolve ties. Only the first trial is shown.*

To vary the parameter between these two values, a sweep strategy is employed. Figure 3 shows the value of a parameter κ (defined so that $\alpha = n^{-1}(1 + \kappa(n-1))$, *i.e.* mapping $[0,1]$ to $[n^{-1},1]$ *individually* for each constraint component) over the first *trial* for two different problems, along with the number of sign changes which is used as a termination criterion.

In particular, fig. 3b shows a full run in which κ is varied throughout the entire range $[0,1]$. As can be seen, κ is initially kept at 0 for a number of iterations — this is in effect an attempt to find guaranteed optimal solutions if possible, only attempting to solve the problem heuristically if this fails. Then, the parameter is increased fairly quickly until reaching an upper limit (70 % of the final value of κ), after which it is increased more slowly.

In fig. 3b the final value of κ is 1, but this is not always the case. For instance, when a solution has been found in a previous trial, the κ for which that solution was found is used as a final value instead. This means that in the next trial of the problem shown in fig. 3a the final value of κ will be around 0,3.

The purpose of increasing κ slowly near this value is to increase accuracy by avoiding overshoot, as a lower α will result in a better approximate solution.

Trials

As briefly mentioned above, the optimization involves several *trials*. Before each trial all constraints, variables and costs are reset to their original state. Then, the parameter sweep is performed and until the final κ value is reached or a feasible solution is found. If $\kappa = 0$ (*i.e.* the solution is optimal), no more trials are run. Otherwise, the program moves on to the next trial.

The number of trials is configurable, but the current implementation moves on to a new trial unless the best solution hasn't been improved in the last 4 trials.

2 BENCHMARKING

In order to determine the efficiency of the algorithm, and determine what problem paradigm the algorithm is most usefully applied to, extensive benchmark testing will be performed. The algorithm was tested against (part of) the large problem set used by de Givry (2014), which includes problems from the MRF, WPMS, CFN, Max-CSP, constraint programming (CP) and Computer Vision and Pattern Recognition (CVPR) domains. All problems in the set are available in the WCSP file format. Exact data (elapsed time and obtained solution for every solver, as well as proven optima and upper bounds for every problem) for these data sets have been obtained directly from de Givry.

This section will describe the method used to benchmark the algorithm, including the calculation of presented data. It will also briefly present the problems used in the benchmark, to provide background that may explain the performance characteristics of the algorithm, as well as short introductions to other solvers with which the in-the-middle algorithm will be compared.

2.1 BENCHMARKING METHOD

All problem instances were limited in runtime by the upper time limit $t_{\max} = 1200\,\mathrm{s}$, and the benchmarks were run on a single core of an Intel Core i5 processor at 2,3 GHz, with 8 GiB RAM. This is comparable to the conditions of the benchmark performed by de Givry (2014), and should ensure that the comparisons are valid.[1] Due to time constraints, data for the comparison solver used in the benchmark was not regenerated on this hardware.

Since the algorithm when used with the corresponding heuristic is an inexact algorithm (while the solvers benchmarked by de Givry (2014) are all exact solvers) the quality of the solution must be compared in addition to the elapsed time per problem. While the time may be compared as-is (given the comparable hardware conditions), the found solution will have to be compared relative to the proven optimum for each problem.

The relative deviation of the obtained solution (an *optimality gap*) may be calculated as $(f - \overline{f})/(\mathrm{UB} - \mathrm{LB})$, where f is the solution found by the algorithm, $\overline{f}$ is the proven optimum and UB, LB is the trivial upper and lower bound of the problem (the upper bound is the sum of the largest cost of each constraint component, and the lower bound is trivially the lowest cost among all constraints). When $\overline{f}$ is unknown, the lower bound is used instead.

For the "push" operation, the constant ρ was set to 5, the reduction of κ was set to 0,8 and the algorithm was run for at most 100 additional iterations. This matches the parameters used by Bastert, Hummel, and de Vries (2010).

The stochastic tie-breaking mechanism was included in all benchmark runs, with threshold $\epsilon = 0{,}01$ and noise amplitude $\zeta = 1$. Since all constraint clauses of the problems used in the benchmark have integer costs, having $\zeta \leq 1$ ensures that the tie-breaking noise will never promote tied variables above better, non-tied solutions.

2.2 PROBLEM SETS

The problem sets obtained from de Givry (2014) belong to a number of different domains and represent different types of problems from industry, academia and random generation. This section will briefly review each problem set used in the benchmark, and review both problem source, interpretation and size. All of these problems are directly representable as WCSPs, and hence as max-sum optimization problems, and

1. In fact, brief testing with the Toulbar2 solver on the same hardware supports this — runtimes are of the same order of magnitude as those found by de Givry (2014).

de Givry (2014) provide details on the translation from each field to the WCSP formulation used in this benchmark.

Several sets from the benchmark performed by de Givry (2014) have been omitted from this benchmark. This is because the in-the-middle algorithm wasn't able to solve any instances in the set (due to time or memory constraints), making them uninteresting in the sense that the algorithm isn't a feasible alternative for those problems. The omitted problem sets are mostly from the CP and WPMS categories (where only two resp. one problem(s) were kept), but the *Chinese Characters*, *Colour Segmentation*, *Matching Stereo* and *Photo Montage* sets from CVPR are also omitted from the benchmark.

Constraint Function Network (CFN)
This category contains six problem sets, all from the CFLib collection mentioned by de Givry (2014, p. 3). Most of them are real-world problems or generated to approximate such problems, and all of them are readily available in the WCSP file format mentioned earlier.

Auction The combinatorial auction problem has been previously used by Larrosa, Heras, and de Givry (2008) and Sandholm (1999). In summary, the problem allows bidders to bid for indivisible subsets of goods, and the optimization problem is to maximize the revenue of the bid-taker. The problems are generated, but inspired by real-world scenarios. All variables are binary (the original problem is a binary max-SAT problem), and the problems contain up to 246 variables and 12 000 constraints.

The problem set includes *scheduling* and *path* distribution problems, but omits the *regions* distribution mentioned by Larrosa, Heras, and de Givry (2008, p. 228).

CELAR As detailed by Cabon et al. (1999) (and to some extent Meseguer, Rossi, and Schiex 2006, p. 315 sq.), this problem set concerns radio frequency assignment, *i.e.* the problem of providing communication channels from limited resources while minimizing interference in the network. The problems where initially introduced in 1993 by *Centre d'Electronique de l'Armement*, and are based on real-world data.

The CELAR problems are fairly large, with variable domains ranging up to 44, with up to 458 variables and 2400 constraints. Problems included in the benchmark are mainly the CELAR sub-instances (Cabon et al. 1999, p. 85) and some GRAPH instances.

Pedigree This category contains problems relating to the Mendelian error correction on complex pedigree (Sánchez, Givry, and Schiex 2008; Meseguer, Rossi, and Schiex 2006, p. 317 sq.), which is a real-world WCSP. The problem may be described as surveying a pedigree (similar to a family tree), detecting individuals that are erroneous in the sense that they do not conform to the Mendelian laws of inheritance. Specifically, the problem formulation is to find the minimum number of errors needed to explain erroneous data.

The problems are very large, with the number of variables reaching 10 000 and almost 20 000 constraints, with variable domains around 25.

Protein Design Computational protein design (CPD) problems concern the identification of proteins performing given tasks. The actual problem statement is the optimization of a complex energy function over amino acid sequences, and it is described in length by Allouche et al. (2012).

The problems may be expressed by CFN or ILP models — only the CFN formulations are used in this benchmark. The problems contain few (roughly 20) variables with very large domains (up to 200), and around 170 constraints.

SPOT5 The SPOT5 problems are in essence planning problems, taken from real-world planning of earth optical observation satellites. Given a number of images to be taken during one day using one of three instruments, an associated importance and a set of imperative constraints (transition times, data flow limitations, on-board recording capacity *etc.*), the problem is to find a feasible subset of images that maximize the sum of the associated weights (Bensana, Lemaître, and Verfaillie 1999).

The problems are large, with roughly 1000 variables and 22 000 constraints, but the variables are all 4-ary.

Warehouse Originally presented by Kratica et al. (2001), the uncapacitated warehouse location instances are randomly generated instances of the facility location problem. In essence, the problem concerns the optimal placement of facilities (in this case warehouses) while minimizing transport costs. These instances were previously used by de Givry et al. (2005) in their evaluation of existential arc consistency for CSPs.

These problem instances are very large. The variable domain reaches 300 for some problems, with 1100 variables and 101 100 constraint functions.

Constraint Programming (CP)

Two problems (one real-world problem and one academic) from this category have been included in this benchmark. All problems in these sets come from the MiniZinc Challenge[1] (de Givry 2014, p. 5), and represent specific problems (representable as

1. `http://www.minizinc.org/`

WCSP instances) defined through constraint programming languages. Note that CP problems are not generally representable as WCSPs, which makes this category less interesting in the context of WCSP or max-sum algorithms.

On-call Rostering This problem is a planning problem in which staff members are assigned to days in a rostering period. Requirements on staff (both forced rostering and staff being unavailable) affect the schedule, and work load should be even over the rostering period. Additionally, staff members are not allowed to be on call more than two days in a row, and prefer not to be on call for two consecutive days. Other, similar constraints may also be present.

Problems in this set are generally fairly large in terms of domain size (up to 90), but only contain up to 2200 variables and 4500 constraints, which is small compared to other sets.

Parity Learning The parity learning set contains instances of an optimization variant of the minimal disagreement parity (MDP) problem (Crawford, Kearns, and Schapire 1994). A set of input/output samples of a Boolean function is given, where the function outputs the parity of an unknown subset of the input variables. A stated number of the input/output samples are incorrect with respect to the given function, and the goal of the parity learning problem is to find a subset of the input variables that minimizes the number of errors.

In terms of problem size, instances of this set are fairly small. The number of variables is below 760, and the number of constraints at most 1440. The variable domain sizes are below 20.

Computer Vision and Pattern Recognition (CVPR)
In this category there are nine problem sets containing MRF instances from the OpenGM2 benchmark (Kappes et al. 2013). The problems have been collected from various sources (p. 1330), but all concern various computer vision tasks performed on real-world images.

The size of these problems vary, with 20 to 500 000 variables, 210 to 2 000 000 constraints and variable domains reaching 20 for some sets.

Max-CSP
The seven max-CSP sets are restated binary CSP instances such that the optimal solution of each instance is the minimal number of unsatisfiable constraints in the original CSP problem. The instances are from the 2008 max-CSP competition.[1,2] The

1. http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html
2. http://www.cril.univ-artois.fr/CPAI08/

problems are mostly academic and random, with a relatively small number of variables and constraints (below 450 and 6500, respectively).

Black Hole "Black Hole" is a solitaire card game in which card from 17 piles are moved to a centre pile according to certain rules. The instances in this set correspond to a simplification of this solitaire game, and were used in the 2005 CSP Solver Competition. Gent et al. (2007) provide a theoretical background to the CSP formulation of this problem.

Colouring In the well-known graph collaring (decision) problem, the objective is to decide whether a given graph is k-colourable, *i.e.* whether each edge can be assigned one of k distinct colours such that no adjacent (connected) nodes have the same colour. As such, the instances are crafted academic problems. The instances in this set originate from the the *Center for Discrete Mathematics and Theoretical Computer Science* (DIMACS), and have been previously studied by Benhamou and Saïdi (2007).

Composed These are random instances composed of an unconstrained CSP core combined using binary constraints with auxiliary fragments. Such problems have been previously used by Lecoutre, Boussemart, and Hemery (2004) and Jussien, Debruyne, and Boizumault (2000).

EHI The EHI problem instances are random 3-SAT (SAT problems where every clause consists of exactly three literals) problems converted into CSP instances with binary constraints, then further restated as max-CSP problems. They have previously been considered by Lecoutre, Boussemart, and Hemery (2004).

Geometric This set contains problems generated from random points in the unit square. For every pair of points, a hard constraint forbidding the pair is added if they are within a certain distance from each other. These instances were used in the 2005 max-CSP competition (Boussemart, Hemery, and Lecoutre 2005).

Langford The Langford instances are academic instances solving the problem of arranging k sets of numbers ranging from 1 to n so that appearances of the number m are exactly m places apart. A general formulation of the problem has been presented by Linek (2003).

QCP The quasi-group completion problems (QCPs) are concerned with deciding if a partial Latin square can be filled in order to obtain a full Latin square. The set consists of 60 instances previously used in the 2005 CSP Solver Competition, and the general problem has previously been studied by Gomes and Shmoys (2002).

Markov Random Fields (MRF)
This category consists of seven sets, where the task is to estimate MAP probabilities on MRF. The sets represent different underlying problems such as image alignment, genetic linkage analysis, protein folding and other probabilistic problems.

All of these problems, except those in the *Linkage* set (which was used in the UAI'08 probabilistic inference evaluation and later by Favier et al. (2011) and Kishimoto and Marinescu (2013)), are from the 2011 Probabilistic Inference Challenge.[1] The problems were translated into their WCSP equivalent using a $-\log$ transformation (de Givry 2014, p. 4).

Most problems are modest in size, with 60 to 2000 variables and 1000 to 10 000 constraints and variable domains below 30. Some problems have variable domains approaching 500, and some have up to 6400 variables and 20 000 constraints. Notably, the *Segmentation* set contains both binary and 21-ary formulations of each problem.

Weighted Partial Max-SAT (WPMS)
From the field of WPMS, only one problem set was kept. Problems in this field contain a very large number of (binary, for obvious reasons) variables and cost functions with very large arity — in fact, this is the only field in which cost function arity exceeds 5 (most other sets have cost functions of two variables only). The only kept problem set, *Max-Clique*, has a cost function arity of 2. Discarded sets were omitted due to memory concerns, likely caused by inefficient representation of the cost functions. The instances were used in the 8th Max-SAT Evaluation.[2]

Max-Clique The Max-Clique problem, which may be restated as a max-SAT problem (Heras and Larrosa 2008), is the well-known problem of finding the largest *clique* (complete subgraph) of a graph. It may be regarded as an academic problem, but has many applications to real-world problems and has seen extensive study when it comes to tailored algorithms for the original formulation. The original instances are from the second DIMACS challenge (Johnson and Trick 1996), and have been previously used by *e.g.* Östergård (2002) for benchmarking max-clique algorithms.

When translated into their max-SAT equivalent, max-clique instances become very large. While the number of variables is fairly low (below 3400), the number of constraints is very large (approaching 380 000).

2.3 SOLVERS

Three solvers used by de Givry (2014) are included in this benchmark. This section will briefly describe their original field of application, give brief pointers to the

1. `http://www.cs.huji.ac.il/project/PASCAL/` 2. `http://maxsat.ia.udl.cat/13/benchmarks/`

method they employ, and relate them to the in-the-middle algorithm.

Toulbar2
The Toulbar2 solver is an exact anytime solver for WCSPs based on a depth-first branch-and-bound algorithm (Allouche, de Givry, and Schiex 2010). In addition to using a strong arc consistency property,[1] the solver uses a sizeable bag of tricks including variable elimination, dead-end elimination (de Givry, Prestwich, and O'Sullivan 2013) and pairwise decomposition (Favier et al. 2011).

The Toulbar2 solver is by far the most advanced solver in the WCSP field, with significant amounts of theory behind it especially with respect to strong arc consistency properties (the one implicitly employed by the in-the-middle algorithm, generalized arc consistency, is quite weak in comparison). It is therefore an interesting benchmark "opponent", and matching it in a benchmark (especially the MRF, CFN and max-CSP categories) could indicate potential for the in-the-middle algorithm in those categories.

CPLEX
The well-known CPLEX solver, which is an exact mixed integer linear programming (MILP) solver rather than a WCSP solver, is also included in the benchmark. Using the *direct* encoding described by de Givry (2014, p. 3) of WCSPs into 0–1 ILP problems, CPLEX was found to have very good performance for some problem categories an therefore it is also included in this benchmark. CPLEX is highly optimized proprietary software, but uses the well-known simplex and barrier interior point methods to solve LP problems. Even so, previous results (Mason 2001; Ernst, Jiang, and Krishnamoorthy 2006) have shown that the LP formulation of the in-the-middle algorithm is competitive with CPLEX, and it is therefore interesting to see how the two relate in the WCSP field.

MaxHS
The MaxHS solver is a max-SAT and WPMS solver based on decomposing max-SAT problems into several smaller SAT instances, solving these using cooperation between an underlying SAT solver and a MILP solver (Davies and Bacchus 2011). It has been found to perform well in solving non-random max-SAT instances (Davies and Bacchus 2013), and is therefore a prime candidate for comparison in the WPMS (and to some extent max-CSP) categories.

1. Existential directed arc consistency, as presented by de Givry et al. (2005).

IV

Results

This chapter will review the results of the benchmark, and compare these to previous results due to de Givry (2014). Results from two modified variants, using the "push" operation and using the greedy DP update, will also be reviewed and compared to the benchmark of the standard algorithm. Problem sets which yield good performance will be identified and examined further.

The chapter is divided into three parts. First, benchmarking results of the standard in-the-middle algorithm will be presented. Then, results from both modified variants will be presented along with the chosen subset of benchmarking problems applied to these specific variants. Finally, the results are discussed and interpreted in further depth.

1 STANDARD ALGORITHM

Benchmarking the standard algorithm on the large set of problems provided earlier produced mixed results. The number of problems (out of the total number available from each set) that were solved by the in-the-middle algorithm indicate that the algorithm is able to solve the same types of problem as the *Toulbar* and *CPLEX* solvers, with a few exceptions. This implies that the algorithm may be useful in most of the fields from which problems were drawn, but does not indicate whether it is useful in its current state, performance-wise.

However, as table 1 shows, the algorithm performed very well when comparing runtime to other solvers and in fact it was the fastest for almost half of the sets after removing incomplete data (runtimes based on data where less than 70 % of the problems were solved). In most problem sets the optimality gap was small as well (for several sets optimal solutions were found) with the notable exception of the *Scene Decomposition* set.[1]

It should be noted that Toulbar2, CPLEX and MaxHS may be at a slight disadvantage in terms of runtime presented in table 1. Brief testing of the Toulbar2 solver indicated that the difference in hardware between the in-the-middle runtimes and those provided by de Givry (2014) may skew the results in favour of the in-the-middle solver, with actual runtimes on the same hardware being 0 % to 50 % lower for Toulbar2. Fortunately, this difference only makes comparisons in a small number of the sets (*Pedigree*, *In-Painting* and *Max-Clique*) potentially invalid, since the difference in runtime between solvers is large in most cases.

The algorithm shows promise especially in the Max-CSP and MRF categories, where overall solution quality is good and the algorithm had the best performance for several sets. Compared to both Toulbar2 and CPLEX, the in-the-middle algorithm appears to be a useful complement providing (good) approximate solutions to problems the other solvers have great difficulty in solving. Results from the CP category are also promising, but these problem sets are small and not all CP problems have max-sum formulations.

Figure 4 shows accumulated runtimes for two of the sets in which the algorithm performed well. The algorithm has a consistent advantage in the *Segmentation* set (fig. 4a), which is reflected by the mean runtime in table 1. Figure 4b highlights a more interesting situation. It shows performance in the *DBN* set, in which the algorithm has an advantage in total runtime across the whole set, almost entirely due to good performance on the more difficult problems. In fact, the runtimes are fairly evenly distributed whereas the runtimes for CPLEX and Toulbar vary significantly between the difficult and easy problems of the set.

The largest problem solved optimally by the in-the-middle algorithm was an instance of in the *In-Painting* set with search space $d^n = 4^{14\,400}$. The smallest unsolved instances are in the *Auction* set, with search spaces $d^n \approx 2^{80}$.

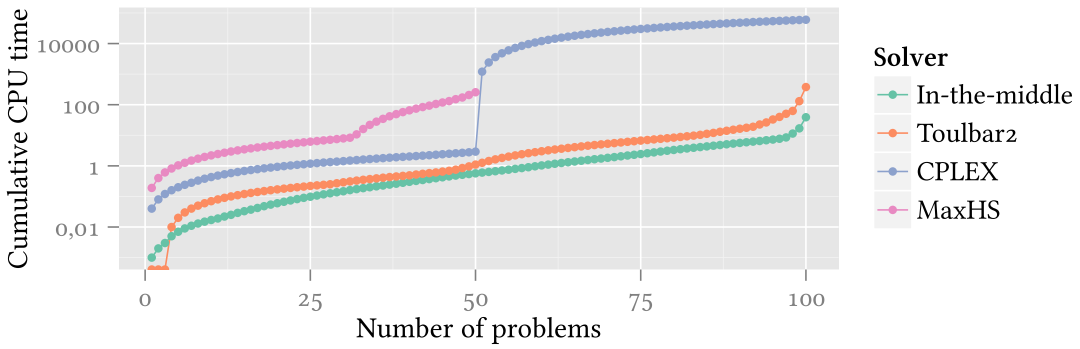
2 Extensions and Improvements

Two variants of the algorithm have been discussed earlier in the thesis, with different purpose and functionality. Benchmark data was produced for these variants as well, but on a limited subset of the problems chosen specifically to test the variants and their intended purpose. Results for both variants will be reviewed in in order to determine if they have the desired effect, and if their function is satisfactory with respect to the drawbacks they introduce.

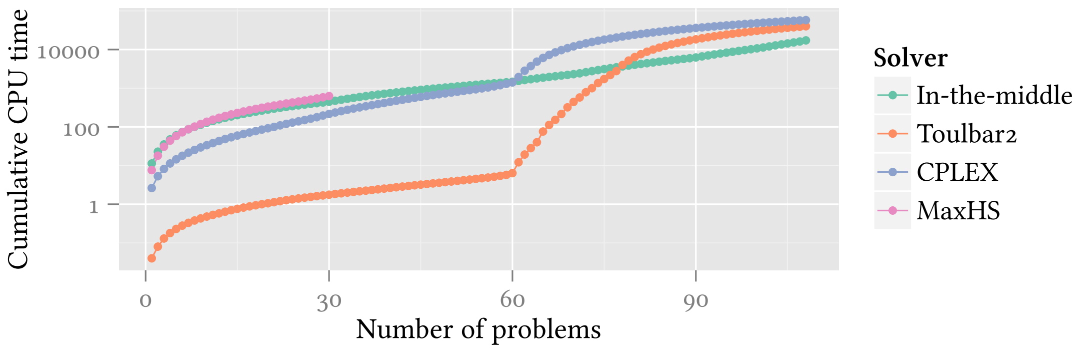
1. Note however that the Toulbar2 solver finds the same non-optimal solutions in this problem set.

Table 1: *Optimality gap and runtime. For each problem instance used in the benchmark, the in-the-middle solver runtime is compared the other solvers included in the benchmark, and the objective value is compared to the best known optimum from de Givry (2014). Problem sets marked with † include unsolved problems (no feasible solution found by the in-the-middle solver), and n/a values indicate that none of the problems in the set were solved. Runtimes based on less than 70 % of the problems are faded, while the best runtime of those remaining is emphasized.*

			Mean solution time (s)			
Category	Set	Gap (%)	ITM	Toulbar2	CPLEX	MaxHS
CFN	Auction†	0,000	82,86	8,20	*0,03*	0,04
	CELAR†	0,000	193,34	*22,38*	1200,00	n/a
	Pedigree	1,805	*2,38*	4,13	0,71	0,03
	ProteinDesign	0,000	43,40	*2,33*	1200,00	n/a
	SPOT5†	0,005	6,44	1200,00	*0,47*	0,82
	Warehouse†	0,000	55,86	0,16	*0,05*	0,56
CP	OnCallRostering†	0,000	*10,35*	71,04	1200,00	18,95
	ParityLearning	0,000	*34,53*	368,08	1200,00	222,69
CVPR	GeomSurf	2,091	*0,05*	0,07	6,62	27,11
	InPainting	0,018	*1009,51*	1200,00	1200,00	n/a
	Matching	0,000	17,93	*4,12*	1200,00	n/a
	ObjectSeg	0,000	*1200,00*	*1200,00*	*1200,00*	n/a
	SceneDecomp	75,455	0,02	*0,02*	1200,00	521,16
Max-CSP	BlackHole	0,901	*58,89*	1200,00	315,05	0,64
	Coloring	0,000	1,69	*0,41*	1,28	0,03
	Composed	0,134	20,34	*0,12*	5,76	32,69
	EHI	0,900	*191,22*	1200,00	1200,00	n/a
	Geometric	1,082	98,98	0,62	1200,00	*0,15*
	Langford	1,311	*70,78*	600,26	851,61	0,27
	QCP	1,292	43,26	1200,00	1200,00	*0,13*
MRF	DBN	0,000	37,90	*0,18*	48,28	20,02
	Grid†	n/a	n/a	1200,00	*160,64*	542,85
	ImageAlignment	0,000	*0,58*	1,80	1200,00	n/a
	Linkage†	0,000	41,07	32,05	327,63	*16,52*
	ObjectDetection	6,466	*279,86*	1200,00	1200,00	n/a
	ProteinFolding†	0,000	1200,00	*23,14*	116,74	n/a
	Segmentation	0,000	*0,03*	0,15	600,07	0,30
WPMS	MaxClique†	2,583	*257,09*	389,75	481,55	8,80



(a) *The* Segmentation *set of the MRF category.*



(b) *The* DBN *set of the MRF category.*

Figure 4: *Accumulated runtime of the algorithms in three different sets, sorted by runtime individually for every solver. Note the logarithmic scale of the y axis.*

Table 2: *Optimality gap and runtime using the "push" operation. For several chosen problem sets, the "push" variant runtime is compared to the results obtained by the standard algorithm (see table 1).*

		# solved		Gap (%)		Mean time (s)	
Category	Set	Std.	"Push"	Std.	"Push"	Std.	"Push"
CFN	Pedigree	10	10	1,805	1,512	2,38	5,61
CVPR	GeomSurf	600	600	2,091	2,091	0,05	0,04
	SceneDecomp	715	715	75,455	75,455	0,02	0,02
Max-CSP	BlackHole	37	37	0,901	1,081	58,89	31,33
	Langford	4	4	1,311	1,554	70,78	56,29
	QCP	60	60	1,292	1,304	43,26	38,07
MRF	ObjectDetection	37	37	6,466	6,466	279,86	167,02

2.1 THE "PUSH" OPERATION

The purpose of the "push" operation is to decrease the optimality gap of the approximate algorithm once a feasible solution has been found. Six problem sets from table 1 were therefore chosen for this benchmark, all with comparatively bad solutions (optimality gaps close to or above 1 %) but competitive runtimes. The expectation was to obtain better solutions while maintaining good runtimes.

Table 2 shows the results of benchmarking the "push" operation on the selected problems. Surprisingly, the optimality gap did not improve for a majority of the problems. In fact, for some sets the optimality gap was increased, and the runtime of the algorithm improved instead (which given the already competitive runtime of the standard algorithm is an unwanted result). In fact, the only problem set for which the expected result was obtained is the CFN *Pedigree* set.

2.2 THE GREEDY DP UPDATE

The greedy DP update, obtained by fixing $\alpha = 1$ of the fractional DP update, should theoretically improve convergence at the expense of solution quality. To test this a large number of problems from table 1 were selected, all exhibiting low optimality gaps and a reasonable but noncompetitive runtime. The expectation was to decrease runtime at the expense of solution quality. Additionally, some sets with zero optimality gap and competitive runtime were included to observe the effects of this variant on already well-performing problem sets.

Table 3 shows the results of benchmarking the greedy DP algorithm against the selected problems. As expected, the runtime of all sets (except the *Auction* set) was improved significantly — between 4 and 350 times — but with little or no increase in

Table 3: *Optimality gap and runtime using the greedy DP update (setting $\alpha = 1$). For several chosen problem sets, the greedy DP runtime is compared to the results obtained by the standard algorithm (see table 1). Problem sets marked with † include unsolved problems (no feasible solution found by the greedy DP update), and n/a values indicate that none of the problems in the set were solved. Runtimes based on less than 70 % of the problems are faded.*

		# solved		Gap (%)		Mean time (s)	
Category	Set	Std.	$\alpha = 1$	Std.	$\alpha = 1$	Std.	$\alpha = 1$
CFN	Auction†	102	0	0,000	n/a	82,86	n/a
	ProteinDesign†	10	9	0,000	0,000	43,40	0,72
	Warehouse†	38	53	0,000	0,000	55,86	0,68
CP	ParityLearning	7	7	0,000	0,000	34,53	3,13
CVPR	Matching	4	4	0,000	0,000	17,93	4,55
Max-CSP	BlackHole†	37	36	0,901	1,081	58,89	13,17
	Coloring	22	22	0,000	0,000	1,69	0,21
	Composed	80	80	0,134	0,000	20,34	1,00
	Geometric	100	100	1,082	0,941	98,98	13,37
	Langford	4	4	1,311	0,967	70,78	7,39
	QCP	60	60	1,292	2,123	43,26	4,42
MRF	DBN	108	108	0,000	0,000	37,90	0,45
	ObjectDetection	37	37	6,466	6,466	279,86	0,84
	Segmentation	100	100	0,000	0,000	0,03	0,13

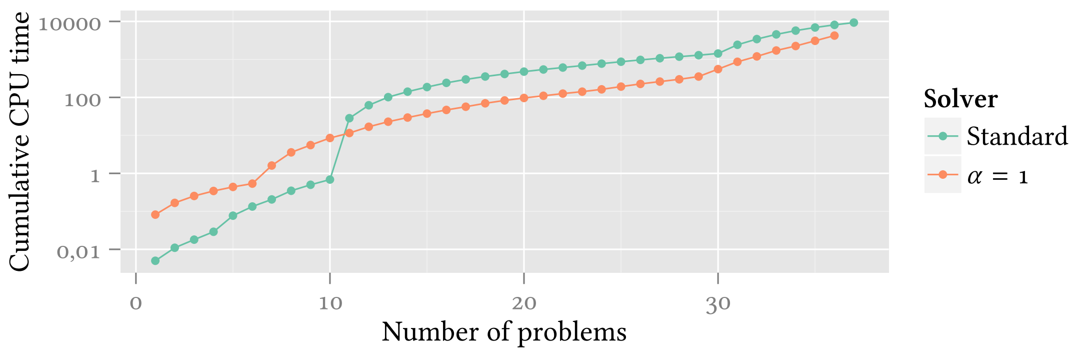


Figure 5: *Accumulated runtime of the standard and greedy algorithm in the* Black Hole *set, sorted by runtime individually for each variant. Note the logarithmic scale of the y axis.*

optimality gap. The runtime improvements are most significant for the CFN and MRF problems, where the greedy DP variant is competitive with all three other solvers. Three problem sets from the max-CSP category additionally show improved optimality gaps as well as better runtimes. The only set in which the greedy update performs significantly worse is the *Auction* set, in which it fails to solve any problems at all.

Figure 5 illustrates the utility of the greedy DP update. While it is slower for very small problems in the *Black Hole* set, it is significantly faster in solving the more difficult problems.

3 DISCUSSION

Analysing the shortcomings of the algorithm it is clear from table 1 that it had significant difficulties in solving the problems from some categories, while it was very successful in others.

The algorithm shows some promise in the WPMS category. While most problem sets in this category were omitted due to memory concerns, the algorithm has good runtime performance in the remaining set. The current implementation represents binary variables of the problem using two distinct variables internally, which could be improved by representing binary variables using only one variable. This could improve both memory use and runtime, making the algorithm more interesting for approximate applications in the WPMS field. The same variable representation issue applies to problems in the *Auction* and *DBN* sets.

One would expect the algorithm to perform well on problems with many hard clauses (due to the sparse representation of constraint components including these implicitly, resulting in fewer clauses to operate on in the constraint update), and this is the case for those sets included in the benchmark. The algorithm performs well in all sets where a majority of the clauses are hard (*On Call Rostering*, *Parity Learning* and *Pedigree*), but performs worse in sets where only 25 % to 50 % of the clauses are hard (*Max-Clique*, *Protein Design* and *Linkage*).

The number of zero-value clauses does not seem to have any effect on the performance of the algorithm, suggesting that the tie-breaking method employed efficiently resolves any resulting ties.

Compared to the results of CPLEX, the LP-based competitor in the benchmark, the in-the-middle algorithm performs better in most sets. This indicates that the translation of the algorithm to a max-sum based approach is a better approach than translating max-sum problems to LP instances. The only sets in which CPLEX performs decisively better (those from CFN) mainly contain problems of an operations research character (resource allocation, planning *etc.*).

Runtimes could potentially be improved by re-implementing the algorithm using single-precision or integer arithmetic. The implementation used in the benchmark uses double-precision arithmetic throughout in order to take advantage of the existing framework of the LP formulation, for instance when considering sign changes in the modified variable components and when applying noise for tie-breaking. Single-precision arithmetic could probably be used with little impact on either the implementation or the solution quality achieved by the algorithm. The problems used in the benchmark all have integer costs (in fact, the Toulbar2 solver appears to work exclusively with integer-valued WCSPs), so in theory it would be possible to use and integer arithmetic. This would however require the code detecting changed variable components to be reimplemented, potentially causing worse performance, and would also require a new tie-breaking mechanism.

3.1 EXTENSIONS AND IMPROVEMENTS

The performance of the two variants of the algorithms was fairly unexpected, especially with respect to the "push" operation. While the optimality gap was expected to decrease, with a corresponding increase in runtime, the opposite happened instead. The reason for this may be that the algorithm, in trials after the "push" operation has been applied, is more conservative than the original algorithm in that the maximum κ value will be lower. This means more trials fail to force an integer solution, reducing the number of trials and as a consequence reducing the runtime as well as increasing the optimality gap. A more correct implementation would take into consideration the reduction of κ caused by the push operation when calculating the maximum κ value of subsequent trials.

Due to the results presented above, the "push" operation is not as interesting when applied to the max-sum algorithm as it is in the original LP formulation.

The result of the greedy DP update was more in line with expectations. Solution quality was in fact largely maintained while improving runtimes significantly. The maintained solution quality may be due to the problem instances having very large ranges between trivial upper an lower bounds, with many solutions of similar cost — this would result in very small changes in the optimality gap, which may not show in the data. However, the optimality gap is still small in all problem sets, which may be an acceptable compromise given the very low runtimes.

For one problem set (*Auction*) the greedy DP update showed abysmal performance, failing to solve any instances. It seems that the reason for this is that the algorithm never reaches a situation where the solution doesn't change between two iterations, instead oscillating between a set of solutions. This could potentially be resolved by increasing the threshold ϵ and amplitude ζ of the tie-breaking noise.

With these results in mind, the greedy algorithm may be very useful when exact solutions aren't required. The greedy update variant may also be useful as part of a broader strategy, by for instance providing fast and good upper bounds or by providing fast unproven solutions while waiting for exact solvers. Another interesting direction for the greedy DP update could be to run it in parallel with the standard algorithm (sharing the immutable constraint components in memory), again providing quick approximate solutions as well as good solutions. This would also be useful in *e.g.* the *Black Hole* set, where the standard algorithm performed better on easy instances and the greedy DP update was better on difficult ones.

3.2 LIMITATIONS

The benchmark has a few limitations which should be mentioned. For instance, the implementation as it stands has the capability to combine regular WCSP constraints with linear (set-partitioning) constraints (Grohe and Wedelin 2008, p. 102), for which more efficient constraint updates may be constructed. For some problem sets, this could result in runtime improvements. However, identifying such constraints in a preprocessing stage increases the complexity of the implementation.

Another limitation of the benchmark is the omission of most WPMS instances. This is a category where conventional WCSP solvers have difficulties due to the large cost function arities, which causes issues with local consistency enforcing. In theory, in-the-middle algorithm has no such issues with large cost function arities, but the memory requirements for such problems are significant. Including these problems in the benchmark would likely require better hardware and/or a specialized implementation (specifically, simplifications can be made when solving WPMS problems due to their binary variable domains).

V

Conclusions

In this final chapter I will attempt to consolidate the results of the thesis work, conclusively addressing the objective and purpose of the thesis itself. I will do this by identifying already apparent applications of the in-the-middle algorithm within the field of graphical models while also identifying the limitations and weaknesses of the algorithm, and by pointing out some areas of further research which I believe are tractable and meaningful to pursue when it comes to applying the algorithm to optimization within that field.

1 APPLICATIONS

Since the performance of the algorithm varies widely between different problem sets, one cannot immediately identify any single field in which the algorithm would be decisively competitive. However, the good overall performance and low optimality gap of the algorithm — especially when applied to problem sets in which other solvers struggle — suggest that the algorithm may be a good candidate for a so-called portfolio approach, wherein several solvers are applied in parallel.

The same properties, especially the good optimality gap, suggest that the algorithm may be useful in applications where good upper bounds on the solution are required. Since the greedy DP variant of the algorithm provides such upper bounds quickly, that variant of the algorithm may have applications in branch-and-bound applications. In combination with an exact solver, the in-the-middle algorithm may serve as an approximate anytime solver providing good approximate solutions while waiting for the exact solver to supply truly optimal solutions.

2 FURTHER RESEARCH

The algorithm shows promise in several categories of those included in the benchmark. In these fields it would therefore be of interest to extend the algorithm and develop the underlying theory to further improve performance in these fields.

Previous work, mainly from the *Institut national de la recherche agronomique* (INRA), has provided strong theory on graphical model optimization especially with respect to arc consistency (Cooper et al. 2010, 2008; de Givry, Schiex, and Verfaillie 2006; de Givry et al. 2005). The in-the-middle algorithm, as previously detailed, uses a relatively weak arc consistency property (general arc consistency). If stronger notions such as existential (de Givry et al. 2005) or optimal soft (Cooper et al. 2010) arc consistency can be extended to the framework used by the in-the-middle algorithm, it may increase runtime performance and solution quality.

Another area requiring further research is the design of the constraint updates, especially their effect on the optimality guarantee of the algorithm. A common situation for the algorithm is that the heuristic finds optimal solutions but cannot guarantee the optimality, which in turn means the algorithm wastes time in subsequent trials. Constructing constraint updates with better guarantees or providing stronger theory on other parts of the algorithm may therefore improve the performance of the algorithm. Additionally, common constraint types such as the *all-different* constraint may benefit from specialized constraint component updates.

There are also many tricks, some of them used in LP implementations of the in-the-middle heuristic, that may be applied to the algorithm. In this benchmark, only tie-breaking using noise was introduced. Further modifications that would require further research include not updating constraint components for which the associated variable components are unchanged and using the existing LP or set-partitioning constraint update (Grohe and Wedelin 2008, p. 102) for constraints of that type. Both these modifications may improve runtime for some problems.

Finally, specializations of the implementation may prove interesting in some areas. For example, the WPMS category of problems would benefit from a formulation where the binary variables are stored in a single, shared variable component element instead of two different ones as in the current implementation. Similar simplifications based on various assumptions of the problem structure may be introduced in the constraint component updates.

Bibliography

Alefragis, P., P. Sanders, T. Takkula, and D. Wedelin. 2000. "Parallel Integer Optimization for Crew Scheduling." *Annals of Operations Research* 99 (1-4): 141–166. ISSN: 0254-5330. DOI: 10.1023/A:1019293017474.

Allouche, D., S. de Givry, B. Hurley, G. Katsirelos, B. O'Sullivan, and T. Schiex. 2014. "An Experimental Evaluation of CP/AI/OR Solvers for Optimization in Graphical Models." In *ROADEF - 15ème congrès annuel de la Société française de recherche opérationnelle et d'aide à la décision.* Bordeaux.

Allouche, D., S. de Givry, and T. Schiex. 2010. *ToulBar2, an open source exact cost function network solver.* Technical report. Institut National de Recherche en Informatique et en Automatique, July 8.

Allouche, D., S. Traoré, I. André, S. Givry, G. Katsirelos, S. Barbe, and T. Schiex. 2012. "Computational Protein Design as a Cost Function Network Optimization Problem." In *Principles and Practice of Constraint Programming,* edited by M. Milano, 7514:840–849. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. DOI: 10.1007/978-3-642-33558-7_60.

Ansótegui, C., M. L. Bonet, J. Gabàs, and J. Levy. 2013. "Improving WPM2 for (Weighted) Partial MaxSAT." In *Principles and Practice of Constraint Programming,* edited by C. Schulte, 8124:117–132. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-642-40626-3. DOI: 10.1007/978-3-642-40627-0_12.

Ansótegui, C., and J. Gabàs. 2013. "Solving (Weighted) Partial MaxSAT with ILP." In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems,* edited by C. Gomes and M. Sellmann, 7874:403–409. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-642-38171-3. DOI: 10.1007/978-3-642-38171-3.

Argelich, J., C. M. Li, F. Manyà, and J. Planes. 2011. "Analyzing the Instances of the MaxSAT Evaluation." In *Theory and Applications of Satisfiability Testing - SAT 2011,* edited by K. A. Sakallah and L. Simon, 6695:360–361. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-642-21580-3. DOI: 10.1007/978-3-642-21581-0_29.

Bastert, O., B. Hummel, and S. de Vries. 2010. "A Generalized Wedelin Heuristic for Integer Programming." *INFORMS Journal on Computing* 22 (1): 93–107. ISSN: 1526-5528. DOI: 10.1287/IJOC.1090.0328.

Benhamou, B., and M. R. Saïdi. 2007. "Local Symmetry Breaking During Search in CSPs." In *Principles and Practice of Constraint Programming – CP 2007,* edited by C. Bessière, 4741:195–209. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-540-74969-1. DOI: 10.1007/978-3-540-74970-7_16.

Bensana, E., M. Lemaître, and G. Verfaillie. 1999. "Earth Observation Satellite Management." *Constraints* 4 (3): 293–299. ISSN: 1383-7133. DOI: 10.1023/A:1026488509554.

Bistarelli, S., U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. 1999. "Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison." *Constraints* 4 (3): 199–240. ISSN: 1383-7133. DOI: 10.1023/A:1026441215081.

Boussemart, F., F. Hemery, and C. Lecoutre. 2005. "Description and representation of the problems selected for the first international constraint satisfaction solver competition." In *Proceedings of the 2nd International Workshop on Constraint Propagation and Implementation,* 7–26. CPAI'05.

Cabon, B., S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. 1999. "Radio Link Frequency Assignment." *Constraints* 4 (1): 79–89. ISSN: 1383-7133. DOI: 10.1023/A:1009812409930.

Cooper, M., S. de Givry, M. Sanchez, T. Schiex, and M. Zytnicki. 2008. "Virtual Arc Consistency for Weighted CSP." In *Proceedings of the 23rd National Conference on Artificial Intelligence,* 1:253–258. AAAI'08. Palo Alto, CA: AAAI Press. ISBN: 978-1-57735-368-3.

Cooper, M., S. de Givry, T. Schiex, M. Zytnicki, and T. Werner. 2010. "Soft arc consistency revisited." *Artificial Intelligence* 174 (7-8): 449–478. ISSN: 0004-3702. DOI: 10.1016/J.ARTINT.2010.02.001.

Crawford, J. M., M. J. Kearns, and R. E. Schapire. 1994. *The minimal disagreement parity problem as a hard satisfiability problem.* Technical report. Computational Intelligence Research Laboratory, University of Oregon and AT&T Bell Laboratories, February 1.

Davies, J., and F. Bacchus. 2011. "Solving MAXSAT by Solving a Sequence of Simpler SAT Instances." In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming,* 225–239. CP'11. Berlin, Heidelberg: Springer. ISBN: 978-3-642-23785-0.

Davies, J., and F. Bacchus. 2013. "Exploiting the Power of MIP Solvers in MAXSAT." In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing,* 166–181. SAT'13. Berlin, Heidelberg: Springer. ISBN: 978-3-642-39070-8.

De Givry, S. 2014. "Multi-Paradigm Evaluation of Exact Solvers in Graphical Model Discrete Optimization." Submitted.

De Givry, S., S. D. Prestwich, and B. O'Sullivan. 2013. "Dead-End Elimination for Weighted CSP." In *Principles and Practice of Constraint Programming,* edited by C. Schulte, 8124:263–272. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-642-40626-3. DOI: 10.1007/978-3-642-40627-0_22.

De Givry, S., T. Schiex, and G. Verfaillie. 2006. "Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP." In *Proceedings of the 21st National Conference on Artificial Intelligence,* 1:22–27. AAAI'06. Palo Alto, CA: AAAI Press. ISBN: 978-1-57735-281-5.

De Givry, S., M. Zytnicki, F. Heras, and J. Larrosa. 2005. "Existential Arc Consistency: Getting Closer to Full Arc Consistency in Weighted CSPs." In *Proceedings of the 19th International Joint Conference on Artificial Intelligence,* 84–89. IJCAI'05. San Francisco, CA: Morgan Kaufmann Publishers Inc.

Ernst, A., H. Jiang, and M. Krishnamoorthy. 2006. "A New Lagrangian Heuristic for the Task Allocation Problem." In *Industrial Mathematics,* edited by M. C. Joshi, A. K. Pani, and S. V. Sabnis, 137–158. Oxford: Alpha Science International Ltd. ISBN: 978-81-7319-577-8.

Favier, A., S. De Givry, A. Legarra, and T. Schiex. 2011. "Pairwise Decomposition for Combinatorial Optimization in Graphical Models." In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence,* 3:2126–2132. IJCAI'11. Palo Alto, CA: AAAI Press. ISBN: 978-1-57735-515-1.

Gent, I. P., C. Jefferson, T. Kelsey, I. Lynce, I. Miguel, P. Nightingale, B. M. Smith, and S. A. Tarim. 2007. "Search in the Patience Game 'Black Hole'." *AI Communications — Constraint Programming for Planning and Scheduling* 20 (3): 211–226. ISSN: 0921-7126.

Gomes, C. P., and D. Shmoys. 2002. "Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem." In *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations,* edited by D. S. Johnson, A. Mehrotra, and M. A. Trick. Ithica, NY.

Grohe, B., and D. Wedelin. 2008. "Cost Propagation — Numerical Propagation for Optimization Problems." In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems,* edited by L. Perron and M. A. Trick, 5015:97–111. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-540-68154-0. DOI: 10.1007/978-3-540-68155-7_10.

Heras, F., and J. Larrosa. 2008. "A Max-SAT Inference-Based Pre-processing for Max-Clique." In *Theory and Applications of Satisfiability Testing – SAT 2008,* edited by H. Kleine Büning and X. Zhao, 4996:139–152. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-540-79718-0. DOI: 10.1007/978-3-540-79719-7_13.

Johnson, D. J., and M. A. Trick, eds. 1996. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993.* Boston, MA: American Mathematical Society. ISBN: 987-0-821-86609-5.

Jussien, N., R. Debruyne, and P. Boizumault. 2000. "Maintaining Arc-Consistency within Dynamic Backtracking." In *Principles and Practice of Constraint Programming,* edited by R. Dechter, 1894:249–261. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-540-41053-9. DOI: 10.1007/3-540-45349-0_19.

Kappes, J. H., B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, K. Sungwoong, et al. 2013. "A Comparative Study of Modern Inference Techniques for Discrete Energy Minimization Problems." In *Proceedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition,* 1328–1335. DOI: 10.1109/CVPR.2013.175.

Kishimoto, A., and R. Marinescu. 2013. "Recursive best-first and/or search with overestimation for genetic linkage analysis." In *Proceedings of the Workshop on Constraint Based Methods for Bioinformatics,* 17–25. WCB'13.

Kolmogorov, V. 2013. “The Power of Linear Programming for Finite-Valued CSPs: A Constructive Characterization.” In *Automata, Languages, and Programming,* edited by F. V. Fomin, R. Freivalds, M. Kwiatkowska, and D. Peleg, 7965:625–636. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-642-39205-4. DOI: 10.1007/978-3-642-39206-1_53.

Kratica, J., D. Tošic, V. Filipović, and I. Ljubić. 2001. “Solving the simple plant location problem by genetic algorithm.” *RAIRO — Operations Research* 35 (1): 127–142. ISSN: 1290-3868. DOI: 10.1051/RO:2001107.

Larrosa, J., F. Heras, and S. de Givry. 2008. “A logical approach to efficient Max-SAT solving.” *Artificial Intelligence* 172 (2-3): 204–233. ISSN: 0004-3702. DOI: 10.1016/J.ARTINT.2007.05.006.

Lecoutre, C., F. Boussemart, and F. Hemery. 2004. “Backjump-based techniques versus conflict-directed heuristics.” In *16th IEEE International Conference on Tools with Artificial Intelligence,* 549–557. ICTAI’04. Washington, DC: IEEE Computater Society. DOI: 10.1109/ICTAI.2004.37.

Linek, V. 2003. “Extending Skolem sequences, how can you begin?” *Australasian Journal of Combinatorics* 27:129–137. ISSN: 1034-4942.

Mason, A. J. 2001. “Elastic Constraint Branching, the Wedelin/Carmen Lagrangian Heuristic and Integer Programming for Personnel Scheduling.” *Annals of Operations Research* 108 (1-4): 239–276. ISSN: 0254-5330. DOI: 10.1023/A:1016023415105.

Meseguer, P., F. Rossi, and T. Schiex. 2006. “Soft Constraints.” In *Handbook of Constraint Programming,* edited by F. Rossi, P. van Beek, and T. Walsh. Foundations of Artificial Intelligence. Amsterdam: Elsevier. ISBN: 978-0-080-46380-3.

Otten, L. 2008. “WCSP file format.” Graphical Model Algorithms at UC Irvine. March 18. Accessed May 16, 2014. `http://graphmod.ics.uci.edu/group/WCSP_file_format`.

Sánchez, M., S. Givry, and T. Schiex. 2008. “Mendelian Error Detection in Complex Pedigrees Using Weighted Constraint Satisfaction Techniques.” *Constraints* 13 (1-2): 130–154. ISSN: 1383-7133. DOI: 10.1007/S10601-007-9029-5.

Sandholm, T. 1999. “An Algorithm for Optimal Winner Determination in Combinatorial Auctions.” In *Proceedings of the 16th International Joint Conference on Artificial Intelligence,* 1:542–547. IJCAI’99. San Francisco, CA: Morgan Kaufmann Publishers Inc. ISBN: 978-1-55860-613-0.

Wedelin, D. 1995. “An algorithm for large scale 0–1 integer programming with application to airline crew scheduling.” *Annals of Operations Research* 57 (1): 283–301. ISSN: 0254-5330. DOI: 10.1007/BF02099703.

Wedelin, D. 2013. “Revisiting the in-the-middle algorithm and heuristic for integer programming and the max-sum problem.” In preparation.

Werner, T. 2007. “A Linear Programming Approach to Max-Sum Problem: A Review.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29 (7): 1165–1179. DOI: 10.1109/TPAMI.2007.1036.

Östergård, P. R. J. 2002. “A fast algorithm for the maximum clique problem.” *Discrete Applied Mathematics* 120 (1–3): 197–207. ISSN: 0166-218X. DOI: 10.1016/S0166-218X(01)00290-6.