Revisiting the in-the-middle algorithm and heuristic for integer programming and the max-sum problem

Dag Wedelin, dag@chalmers.se
Computer Science, Chalmers University of Technology

March 2013, revised November 2013

Abstract

We consider a very simple algorithm to solve large scale 0-1 integer linear programs - and a simple heuristic to encourage convergence to an integer solution - which have been very successful in commercial applications. We find it useful to revisit its main ideas and establish its relation to several known algorithms, including the generalized iterative scaling algorithm. We discuss ways to relate this linear programming based approach to the max-sum problem, and show that the algorithm has a close relation to convergent message passing algorithms for MAP-inference in graphical models such as max-sum diffusion. We further discuss non-conflicting and conflicting max-sum constraint updates, show that the two algorithms match with these concepts, and that the heuristic has a relation to the max-sum algorithm. We finally give a brief overview of known applications, including a commercial system for airline crew scheduling.

1 Introduction

We consider a method for solving large 0-1 integer programming problems, originally developed from the author's experiments with probabilistic inference and loopy belief propagation (Wedelin, 1989), and first described in Wedelin (1995). The method has been shown to be successful in applications, including a crew pairing system used by many large airlines. In this paper we structure the method in two algorithms, the in-the-middle algorithm and the in-the-middle heuristic.

The in-the-middle algorithm is a very simple algorithm, and we introduce it with the assignment problem (weighted bipartite matching) as an example. Consider the

first cost matrix of Figure 1. The assignment problem can be described as matching n people to n tasks, so that the sum of the costs for the chosen combinations is maximal. With binary variables the problem can be written as

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ s.t. \quad & \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1 \dots n \\ & \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1 \dots n \\ & x_{ij} \in \{0, 1\} \end{aligned} \tag{1}$$

The algorithm can be described as follows. Look at the first row, find the largest and second largest numbers (8 and 6), compute their average and subtract from the row. This yields the next updated cost matrix. This is an invariant transformation in that the solution does not change. Do the same for the other rows, do the same for each column, and iteratively repeat for rows and columns until no numbers change sign. The final result is shown in the last table, where the positive numbers indicate a feasible and also optimal solution.

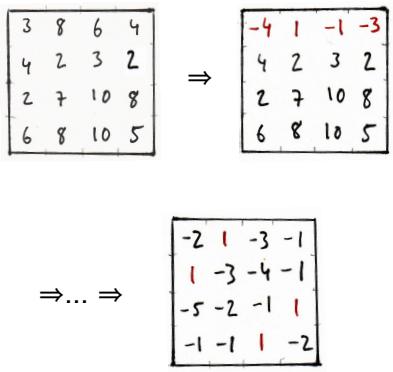

Figure 1: In-the-middle algorithm for the assignment problem.

Before we leave the example we will remark on some details. Just subtracting the smallest number - attempting what in linear programming is known as complementary slackness, does not work since the iteration gets stuck almost immediately. By subtracting the average, we choose a value *in-the-middle* of the interval that would give the same resulting signs, hence the name of the algorithm. The treatment of 0 and 1 is symmetric which is often not the case in linear programming.

In addition to the review of the existing method and its applications, specific new contributions of this paper are: A clear separation between the in-the-middle algorithm and the in-the-middle heuristic, the relationship to generalized iterative scaling including intermediate concepts, the careful presentation of the relationship between the linear programming (LP) dual and the height of a max-sum problem, the relationship to max-sum diffusion, the discussion on non-conflicting and conflicting max-sum constraint updates, the relationship to these updates, and the relationship to the max-sum algorithm.

2 The in-the-middle algorithm for 0-1 integer programming

The in-the-middle algorithm naturally generalizes to the binary integer linear program (ILP)

$$\max cx$$
$$Ax = b$$
$$x_j \in \{0, 1\}$$

where $a_{ij} \in \{-1, 0, 1\}$, and b is integer. While a subset of integer linear programming, the stated problem is NP-hard and includes well known and important problems such as set partitioning. Inequalities are handled by slack variables, which can be implicit in an implementation.

The in-the-middle algorithm is generalized to this problem as in the following way. Consider the reduced cost vector $\bar{c} = c - yA$, this is standard linear programming notation. For the assignment problem example, the vector c corresponds to a vector with the original 16 costs, the vector y corresponds to the 8 values subtracted for the constraints and and $\bar{c}$ corresponds to the updated costs.

Definition 1 *We associate a* **current state** x *to any* $\bar{c}$*, by assigning* $x_j = 1$ *when* $\bar{c}_j > 0$*, and* $x_j = 0$ *when* $\bar{c}_j < 0$*. If* $\bar{c}_j = 0$ *then* x_j *is undefined.*

The algorithm proceeds by iterative updates of $\bar{c}$, where each update attempts to satisfy a single constraint, given this association.

In-the-middle algorithm

For each constraint i in turn, update y_i to make $\bar{c} = c - yA$ feasible for this constraint. Select y_i in-the-middle of the possible interval. Iteratively repeat for every constraint until a feasible solution is found.

In every step, update $\bar{c}$ by updating $y_i := y_i + \Delta y_i$ where

$$\Delta y_i = \frac{r^+ + r^-}{2} \tag{2}$$

Here, r^+ is the $(b_i + N)$'th largest ratio $\bar{c}_j / a_{ij}$ and r^- the $(b_i + N + 1)$'th largest ratio. N is the sum of all negative coefficients a_{ij} of the constraint, so with no negative coefficients $N = 0$. We denote r^+ and r^- as the **critical values** of the constraint.

For this problem, the in-the-middle algorithm will sometimes succeed, and sometimes fail and not find a feasible solution. Note that selecting y_i in the middle of the interval $[r^-, r^+]$ is a design choice - any interior point of the interval would make the constraint feasible. For the special case $r^- = r^+$, see Section 3.

Normally, we are not seeking numerical convergence, but merely convergence to a feasible solution, in terms of the sign of the costs, which thereafter do not change. From this point of view we can see the algorithm as a form of local search, where the current state x is iteratively changed until a feasible solution is found. If iteration is continued, the experience is that the algorithm numerically converges to a fixpoint.

The restrictions on the linear constraints are chosen to ensure that the two **critical variables** (corresponding to the critical values), do not collapse into a single variable, with the consequence that the reduced cost of that variable is forced to 0, and the solution is undefined. However, one can always try to run the algorithm on a wider set of constraints.

The reader who wishes to quickly get an overview of the algorithms can continue to Section 3.

2.1 A dual interpretation

Choosing the state x as suggested is the same as solving the Lagrangian relaxation

$$\max_{0 \le x \le 1} cx + y(b - Ax) = yb + \max_{0 \le x \le 1} \bar{c}x \tag{3}$$

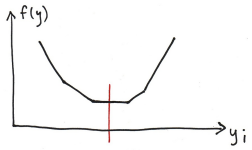

Figure 2: Minimizing $f(y)$ along one coordinate.

for a given y, where we have dualized the constraints $Ax = b$. Let the dual $f(y)$ be the value of this relaxation

$$f(y) = yb + \max_{0 \leq x \leq 1} \bar{c}x \tag{4}$$

It is well known that finding an y so that the solution of (3) satisfies $Ax = b$, corresponds to the unconstrained minimization of $f(y)$, which is a convex and piecewise linear function. More details about this function (in a minimization setting) are given in Wedelin (1995). We here make two observations. First, satisfying a single constraint is the same as minimizing $f(y)$ along a single coordinate, since the two main terms then balance each other, see Figure 2. The in-the-middle algorithm can therefore be interpreted as a *dual coordinate descent algorithm*. Secondly, since $f(y)$ is convex it follows that if all constraints are satisfied, we have found a point within a minimum plateau of $f(y)$.

We remark that this coordinate descent algorithm can be interpreted in a plain LP context, and be adapted for arbitrary $Ax = b$, but it would get stuck very quickly for general problems.

The dual interpretation is useful to establish necessary conditions for when a feasible solution can be found with the in-the-middle algorithm:

Theorem 1 (Wedelin, 1995) *A dual y such that $\bar{c} = c - yA$ gives a unique feasible solution exists if and only if the LP-relaxation of (ILP) has a unique solution that is also integer.*

See Wedelin (1995) for the proof. There must be no more than a single optimal solution, this is natural and trivial. That the LP-relaxation is integer is well known to characterize easy cases of ILP, since they can then be solved by a plain LP-algorithm. The LP-relaxation of the assignment problem is known to be tight in this sense. So while the in-the-middle algorithm is useful for some problems, there are also difficult problems that we cannot expect to solve in this way.

Another possible reason for failure of the in-the-middle algorithm has to do with the fact that generally coordinate descent on a piecewise linear function may get stuck and not reach the true minimum, a property that is well known and shared with similar algorithms. On the other hand, the coordinate descent step is very cheap compared to an iteration step in an arbitrary direction.

2.2 The relationship to the generalized iterative scaling algorithm

Iterative Proportional Fitting (Fienberg, 1970) (also called IPF, IPFP, IPS or Deming-Stephan algorithm), is a well known iterative algorithm which in its simplest form adjusts values in a two way contingency table to given marginals. The algorithm proceeds by scaling each row to fit the marginals, then scaling each column to fit its marginals, repeating until the marginals are sufficiently close. We assume this algorithm to be known by the reader.

In Darroch & Ratcliff (1972) a generalization is described (generalized iterative scaling, GIS), which solves a **scaling problem** where we from a given initial distribution q, wish to find a distribution p satisfying given linear constraints $Ap = b$ and the normalization constraint $\sum_i p_i = 1$. The relationship between p and q is

$$p_j = q_j \mu_0 \prod_i \mu_i^{a_{ij}} \tag{5}$$

where μ_i is the **scaling factor** for constraint i of $Ap = b$, and where μ_0 is the scaling factor for the normalization constraint. Just as the IPF, the GIS iteratively considers one constraint at a time, by updating the corresponding scaling factor so that p satisfies this constraint. If a solution exists, it is known to minimize the Kullback-Leibler divergence

$$H(p, q) = \sum_j p_j \log p_j - \sum_j p_j \log q_j \tag{6}$$

subject to the constraints, and the GIS is known to converge if q is positive. In (Darroch & Ratcliff, 1972) it is assumed that $\sum_j p_j = 1$ and $\sum_j q_j \leq 1$, however if the rhs of these expressions are modified to any constants, $H(p, q)$ scales linearly without changing fundamental results.

We model (ILP) as a scaling problem in the following way:

- We associate the variables p with the variables of (ILP), and let the constraints $Ap = b$ be the same as $Ax = b$ of (ILP). We initialize q from c as $q_j = d^{c_j}$, where $d > 1$ is a parameter.

- We create a **binary scaling problem** by adding **complementary variables** $\tilde{p}$, and **complementary constraints** $p_j + \tilde{p}_j = 1$. This ensures that all $p_j \leq 1$, independently of other constraints. Further, the sum of all variables $\sum_j p_j + \sum_j \tilde{p}_j$ will be constant for any solution, making the separate normalization constraint redundant. We initialize $\tilde{p}$ with $\tilde{q}$, where $\tilde{q}_j = 1$.

We observe that with this model, the sum of the second term of (6) corresponds to the linear objective of (ILP), which for large d will dominate over the first term, creating a close relationship between the solution to the binary scaling problem and that of (ILP) (its LP-relaxation). However, details are non-trivial, and the rest of this section does not depend on this observation.

We now turn to the GIS constraint update, using the constraint $p_1+p_2+p_3+p_4 = 1$ as an example. The constraint is satisfied by updating (5) with $\mu_i := v_i \, \mu_i$, where v_i is determined from the equation

$$v_i(p_1 + p_2 + p_3 + p_4) = 1 \tag{7}$$

We note that this standard GIS will converge very slowly for a typical (ILP) due to an expected huge variation in the magnitude of the q_j.

In order to establish a relationship with the in-the-middle-algorithm we need to take two important steps. The first step is to consider a new **GIS-C algorithm** (C for Critical), which only takes into account *two critical values* p^+ and p^- of the constraint, defined analogously with the critical values of the in-the-middle algorithm. For the constraint of (7), the critical values are the *two largest* p-values of $p_1, \ldots, p_4$, the other p-values are taken as 0, and (7) is modified to

$$v_i(p^+ + p^-) = 1 \tag{8}$$

As long as the p_i are of very different order of magnitude, the so calculated v_i will be close to that of GIS.

The other important step is to *simultaneously update* a constraint and the complementary constraints for all variables of the constraint. So we let one accelerated update perform the equivalent of a repeated iterative update of these single constraints. If we do this for the GIS-C algorithm, we receive an **accelerated GIS-C algorithm** for the binary scaling problem. By considering the GIS-C and not the GIS, the accelerated update becomes easy to compute in one step. All we need to determine v_i is to consider a subproblem with three constraints

$$\begin{array}{rcl} v_i(v^+ \, p^+ + v^- \, p^-) & = & 1 \\ v^+(v_i \, p^+ + \tilde{p}^+) & = & 1 \\ v^-(v_i \, p^- + \tilde{p}^-) & = & 1 \end{array} \tag{9}$$

The first equation corresponds to (8). The two other equations are for the complementary constraints of the critical variables, with v^+ and v^- as their scaling factors. The equations for the other complementary constraints are not needed to determine v_i. We may also w.l.o.g. assume that $\tilde{p}^+ = \tilde{p}^- = 1$, i.e. that any previous scaling of the complementary constraints is removed before the simultaneous update. In fact, in an implementation the complementary variables and their constraints do not need to be explicitly represented, so there is no need to actually update them.

Solving (9) for v_i gives

$$v_i = \frac{1}{\sqrt{p^+ p^-}} \tag{10}$$

The relationship to the in-the-middle algorithm can now be established by matching $\bar{c} = c - yA$ with the d-logarithm of (5) (w.l.o.g let $\mu_0 = 1$). This gives $\bar{c}_j = \log_d p_j$ and $y_i = -\log_d \mu_i$. Further, $r^+ = \log_d p^+, r^- = \log_d p^-$ and $\Delta y_i = -\log_d v_i$. The d-logarithm of (10) then becomes

$$\Delta y_i = \frac{r^+ + r^-}{2}, \tag{11}$$

which is the expression for the in-the-middle algorithm. Generalization to all constraint types in (ILP) is straightforward. In summary, we have shown the following: With these definitions, we have the following theorem:

Theorem 2 *The in-the-middle algorithm is equivalent to the accelerated GIS-C algorithm for the binary scaling problem.*

3 The in-the-middle heuristic

We first illustrate how the in-the-middle algorithm fails when we extend the assignment problem to a weighted 8-queens problem (placing 8 queens on a chessboard so that they do not threaten each other). The ILP model is based on the assignment problem (1), adding diagonal inequality constraints of the type $x_{12}+x_{23}+x_{34} \leq 1$. In (ILP) these are represented as equality constraints with binary slack variables. Repeated iteration then converges to the situation of Figure 3 (the slack variables are not shown). Here many constraints include two or more zeros. This is a fixpoint for the in-the-middle algorithm where x is undefined, so no feasible solution is found. This behavior of the in-the-middle algorithm is characteristic also for other difficult problems.

We will now show how we can help the algorithm to converge to a feasible solution by adding a simple feature, see Figure 4. Relating to the assignment problem

$$\begin{bmatrix} -0.1 & 0 & -3.1 & -0.2 & -0.4 & 0 & -0.3 & -1.1 \\ -2.2 & 0 & -0.2 & -0.3 & 0 & -0.1 & -0.1 & -0.1 \\ 0 & -0.1 & -0.4 & -2.3 & -1.9 & -0.2 & -0.1 & 0 \\ -0.2 & -0.3 & 0 & -0.2 & -0.1 & 0 & -0.5 & -1.0 \\ 0 & -1.1 & -0.1 & 0 & 0 & -0.4 & -0.2 & -0.3 \\ -0.1 & -0.3 & 0 & -0.1 & -1.5 & -0.2 & 0 & 0 \\ 0 & -0.7 & -0.2 & -0.8 & -0.3 & -0.3 & -0.1 & 0 \\ -0.2 & -0.5 & -0.1 & 0 & -0.2 & -1.2 & 0 & -0.4 \end{bmatrix}$$

Figure 3: Convergence for 8-queens problem.

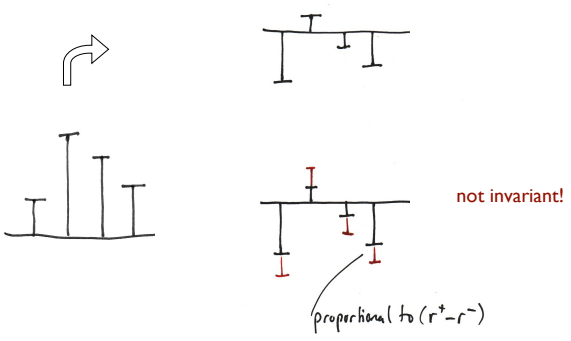

Figure 4: The in-the-middle heuristic.

example, the first figure depicts the costs in the first row of the assignment problem using bars. The next figure shows the in-the-middle update as described. Now, the last figure shows how *we heuristically move the costs away from 0* by adding additional offsets (the red sections in the figure). This counters the observed effect of convergence towards 0, and helps to keep the costs strictly positive or negative.

Unfortunately, the offsets have an undesirable effect in that the problem is *not invariant* to this change. The offsets should therefore be as small as possible to allow the iteration to converge to a feasible and close to optimal solution. We can interpret this as running the in-the-middle algorithm on a problem where the costs are modified in a non-invariant way. The heuristic can therefore be seen as a way to disturb the problem as little as possible so that it becomes easy to solve. An important positive side-effect is that *the heuristic makes the algorithm significantly faster*, by taking larger steps in the typical zig-zag behavior of the coordinate descent (see Wedelin (1995)).

In-the-middle heuristic (or "Wedelin heuristic")

Iteratively repeat for every constraint until a feasible solution is found:

1. Subtract the non-invariant offsets of the previous iteration of this constraint.

2. Update $\bar{c}$ by updating $y_i := y_i + \Delta y_i$ where

$$\Delta y_i = \frac{r^+ + r^-}{2} \tag{12}$$

3. Add the non-invariant offsets

$$\bar{c}_j := \bar{c}_j \ \pm \ \alpha \frac{r^+ - r^-}{2} \tag{13}$$

for all $\bar{c}_j$ of the constraint, so that they are moved away from 0. Remember the change to each variable so it can be undone in the next iteration.

The exact form of the heuristic requires further explanation. Here, $\alpha \geq 0$ is a parameter controlling the magnitude of the offsets. Using the difference between r^+ and r^- is simple, and is successful empirically. It also gives the heuristic a close relationship to the max-sum algorithm, as shown in section 5.1. An alternative **multiplicative heuristic** is to multiply the costs with a suitable factor, i.e. $\bar{c}_j := (1 + \alpha)\, \bar{c}_j$. This can give similar results, but is slightly slower in implementation.

The heuristic is completed by adding two important secondary features:

- **Sweep strategy** It is desirable to keep α as low as possible to avoid problem

distortion, but it needs to be sufficiently high to enable convergence. Since it is difficult to know how to set α for a given problem instance, we can employ a sweep strategy, where we begin to iterate with $\alpha = 0$ and then slowly increase during iteration. This means that we begin by running the plain in-the-middle algorithm, and then gradually introduce the heuristic.

- **Randomization** To resolve ties, we look for reduced costs close to 0, and add small random perturbations to create asymmetry.

With these features added, the heuristic is able to find solutions to problems where the plain in-the-middle algorithm fails. However, it shares the common property of most heuristics, in that they do generally not provide any guarantees with regard to their output. In this case, this is somewhat alleviated by the fact that the dual $f(y)$ provides an LP-based bound for the optimal solution.

In Wedelin (1995), the parameter in the heuristic is described as interpolating between different algorithmic design principles: linear programming ($\alpha = 0$), dynamic programming ($\alpha = 1$, also see Section 5.1), and a greedy algorithm ($\alpha = \infty$), effectively fixing the variables to best satisfy the constraint.

4 The max-sum problem as an ILP

Working with linear costs and constraints enables the theoretical support of optimization theory. However, for non-linear costs and constraints represented as tables, e.g. for statistical models, we commonly consider the max-sum problem (in Bertele & Brioschi (1972) called a *nonserial unconstrained problem*, in constraint programming known as a Valued CSP (Schiex et al., 1995)). We define the max-sum problem as

$$\max_{w} f(w) = \sum_{k} g_k(w^k) + C \tag{14}$$

where $g_k(w^k) \in \mathbb{R}$ are distinct arbitrary functions over subsets w^k of w, and where C is a constant. We call the terms **components**, distinguishing between **variable components** with one variable, and **constraint components** with two or more variables.

Modeling the max-sum problem as an ILP is straightforward and well known, although there can be some variations. Let the resulting ILP be the **max-sum ILP**. For every table entry in every component we introduce a binary variable in the

.

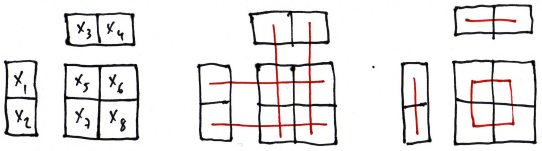

Figure 5: modeling the max-sum problem: variables, marginalization constraints and normalization constraints.

ILP, see Figure 5 left. To model the constant C, we introduce an extra variable x_c, together with the constraint $x_c = 1$.

We need **marginalization constraints** to link constraint components and variable components, see Figure 5 middle. They are written as $x_1 = x_5 + x_6, x_2 = x_7 + x_8$ and so on.

We also need **normalization constraints** to ensure that exactly one value is selected in each component, see Figure 5 right. We have $x_1 + x_2 = x_c, x_5 + x_6 + x_7 + x_8 = x_c$ and so on. Since $x_c = 1$ we can write x_c in the rhs instead of 1, simplifying for the next section.

4.1 The height and its relation to $f(y)$

An upper bound for the max-sum problem can be obtained as the sum of the largest value of each component (plus the constant C). Let this be the **height** of the problem (Werner, 2007). We can write the height as a function of $\bar{c}$ as

$$h(\bar{c}) = \max_{0 \le x \le 1, \text{norm constr}, x_c = 1} \bar{c}x \tag{15}$$

This actually corresponds to the Lagrangian relaxation of the max-sum ILP where only the marginalization constraints have been dualized (the term yb disappears in the relaxation since all dualized constraints have $b_i = 0$). Likewise, $f(y)$ for the max-sum ILP simplifies from (4) to

$$\min_y f(y) = \max_{0 \le x \le 1, x_c = 1} \bar{c}x \tag{16}$$

We then receive

Proposition 3 *$h(\bar{c}) = f(y)$ when the normalization constraints are satisfied, and $h(\bar{c}) \le f(y)$ otherwise.*

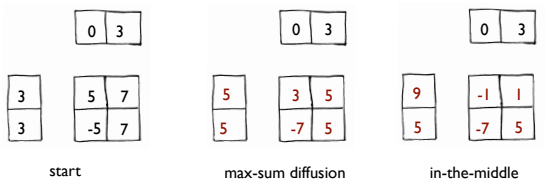

Figure 6: Max-sum diffusion.

We see that the height is equal to $f(y)$ for the normalized version of this problem (i.e. after the normalization constrains are updated according to the in-the-middle algorithm). We can then in max-sum algorithms *minimize the height instead of* $f(y)$, and not care about this trivial normalization until we so desire.

4.2 The relationship to max-sum diffusion

So one way to solve a max-sum problem is simply to run the in-the-middle algorithm on the max-sum ILP. This approach is similar to an algorithm known as max-sum diffusion, and we will now compare the two.

Max-sum diffusion is an algorithm for the max-sum problem developed by Schlesinger et al., see Werner (2007) for a review. In comparison to the in-the middle algorithm which operates on any set of linear constraints, max-sum diffusion, implicitly assumes the max-sum problem and the normalization constraints, and updates only the marginalization constraints.

The algorithm decreases the height by iteratively leveling out the costs between constraint and variable components, with the help of invariant transformations (also known as equivalence preserving transformations (Werner, 2007; Cooper et al., 2000; Cooper, 2008), this is the same as changing y in the max-sum ILP). In one update, it updates the marginalization constraints between one variable component and one constraint component so that the max-marginals of these components become equal. For an example, see Figure 6 where from the initial costs (left), the row-wise marginalization constraints are updated with max-sum diffusion (middle), and the in-the-middle algorithm (right).

We see that the update of the in-the-middle algorithm is different to that of max-sum diffusion for the upper row, but identical for the lower row. The two algorithms both consider two critical values for each constraint, and updates with the help of their average. But in the constraint of the upper row the in-the-middle algorithm

identifies the two values in the constraint component as critical - this can never happen with max-sum diffusion. However, we can eliminate this possibility by an invariant transformation of the normalization constraints so that all costs of the max-sum problem (except C) become negative. Max-sum diffusion is fundamentally unaffected by this change, but for the in-the-middle algorithm we have have the following theorem:

Theorem 4 *If all costs are negative, running the in-the-middle algorithm on the marginalization constraints is equivalent to max-sum diffusion.*

Proof sketch For the marginalization constraint $-x_1 + x_5 + x_6 = 0$, we shall according to the definition of the in-the-middle algorithm select the the critical values as the two largest values of $-c_1, c_5$ and c_6. If all costs are negative, $-c_1$ and the largest of c_5 and c_6 will always be selected. These values are also used by max-sum diffusion, and both algorithms will apply the same invariant transformation to make these values equal. The costs will remain negative also after the update, so the in-the-middle algorithm will continue to behave exactly as max-sum diffusion. □

This observation also establishes a link between max-sum diffusion and the GIS algorithm, via the in-the-middle algorithm as described in section 2.2.

5 Non-conflicting and conflicting constraint component updates

Rather than solving the max-sum problem by just seeing it as an ILP, another natural approach is to design a constraint component update, and in one step solve a subproblem consisting of a single constraint component and all its associated variable components. This can be interpreted as a simultaneous update of all involved linear constraints in the max-sum ILP with invariant transformations.

This is similar to what is common in constraint programming and belief propagation. Like in constraint programming, this is also a path towards finding *very efficient specialized update algorithms for constraints with special structure*, including more general linear constraints than what can be handled directly with the in-the-middle algorithm.

The purpose of this section is to clarify the nature of such updates, and establish when max-sum constraint updates lead to exact or to heuristic algorithms. This is

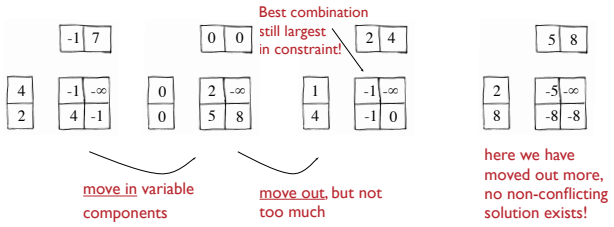

Figure 7: A basic non-conflicting update.

useful in itself, and for further understanding the in-the-middle approach in Section 5.1.

We first consider properties of a solution to the max-sum problem.

Definition 2 *A* **variable component solution** *selects a maximum value from every variable component.*

If all variable components have unique maxima, it is trivial to see the variable component solution. A variable component solution is not guaranteed to be optimal.

Definition 3 (adapted from Grohe & Wedelin (2007)) *A* **non-conflicting solution** *selects a maximum value from every component (both variable and constraint components). Otherwise the solution is* **conflicting**.

Proposition 5 *If a non-conflicting solution exists, it is optimal. Its value is equal to the height of the max-sum problem, which is then minimal.*

The last definition is related to the notion of arc consistency as described in e.g. (Cooper, 2008) for valued and weighted CSP's. It helps us to think directly about a desirable property of a solution, and is related to the notion of complementary slackness in linear programming. In contrast, arc consistency more abstractly refers to a property of the problem. Generally these concepts are descriptions of what the max-sum problem and its solutions look like when the height - or a similar quantity - is minimized.

Now suppose that we would like to find a variable component solution with invariant transformations. A natural goal for a general constraint component update is then to *find an optimal solution to the subproblem and unambiguously reveal that*

solution in the variable components. This can be straightforwardly done in two steps (see Figure 7):

1. **Move in** the costs of the variable components to the constraint component. Any locally optimal solution is now apparent by inspecting the constraint component.

2. **Move out** costs to the variable components so that they uniquely indicate the optimal solution.

After step 1, there is always a locally optimal and non-conflicting solution (the 8 in the example, variable components are uniform), and the height is locally minimal.

As long as just a little is moved out in step 2, the best combination is still largest in the constraint component, the variable component solution is non-conflicting, and the height is still minimal. We call this a **non-conflicting update** (adapted from Grohe & Wedelin (2007)). With non-conflicting updates for all constraint components, it follows that if a solution is found it will be non-conflicting and therefore globally optimal (this holds e.g. for max-sum diffusion).

If a lot is moved out, the non-conflicting solution disappears and the height is no longer minimal. The reason is that when we move out to more than one variable, the best value in the constraint component decreases more than any other value. This is illustrated in the last table of Figure 7, which corresponds to the max-sum algorithm, moving out the max-marginals. The variable component solution is optimal for the subproblem, but it is conflicting. With such **conflicting updates** for all constraint components we obtain a heuristic, in the sense that *the variable components* may converge to a solution which is not globally optimal (as is well known e.g. for the loopy max-sum algorithm). Intuitively, this can happen because the conflicting updates can hide much of the problem in the constraint components.

A natural possibility is to *move out as much as possible while still keeping the non-conflicting solution*. Moving out more can be expected to improve the speed of convergence. The limit is reached when the max-marginals of the constraint component are uniform, we call this a **uniform non-conflicting update**.

The main purpose of the paper is not to detail how to best design general constraint updates, but we mention a few points. Even for the uniform non-conflicting update there is a design choice in how to move out, since if we move out a lot to one variable component, we can normally move out less to another. We can do this in a fair way like for the in-the-middle algorithm, where we maximize the distance to 0 for all costs. Then, if a non-conflicting solution does not exist (existence can be established with theorem 1), conflicting updates are necessary to find a

variable component solution heuristically. Here, the ideas of the in-the-middle heuristic can be directly applied. Non-uniqueness can be handled with noise. We note that a different non-heuristic way to solve the max-sum problem, is to attempt to minimize the height with non-conflicting updates, and then apply branching strategies rather than heuristics (Cooper et al., 2000; Cooper, 2008), an approach that is also common for ILP.

5.1 In-the-middle as constraint component updates, and the relationship to the max-sum algorithm

We are now in a position to ask what kind of update the in-the-middle algorithm corresponds to, if we see it as a constraint component update. To enable the comparison, we model (ILP) as a max-sum problem in the following way. We model the variables of (ILP) with binary variable components, and set the two variable component values to 0 and c_j. We further match each linear constraint of (ILP) with a constraint component by suitably setting the values of the constraint component to 0 or $-\infty$. To avoid confusion, we note that we model (ILP) as a max-sum problem, for which invariant transformations are enabled by a different and larger max-sum ILP.

Theorem 6 *The update of the in-the-middle algorithm is a uniform non-conflicting update.*

Proof idea In the constraint component update, we can move out values to $\bar{c}_j$ in the variable components so that it corresponds to the in-the-middle algorithm. Because of the equality of the linear constraint, and that the same value y_i is used for all variables, all feasible values of the constraint component will have the same value $y_i b_i$ after the update. These values are distributed in the constraint component so that all marginals become equal. □

So, if we should begin to investigate max-sum constraint updates along the lines of Section 5, and then consider specialized updates for simple linear constraints, we would arrive at the same in-the-middle algorithm that is now the starting point of this paper. Within this framework we can also seamlessly mix fast in-the-middle updates for linear constraints, and constraint component updates for other constraints, providing flexibility in modeling different kinds of problems.

In the same way, we can see what the in-the-middle heuristic corresponds to if seen as a constraint component update:

Theorem 7 *The update of in-the-middle heuristic with $\alpha > 0$, is a conflicting*

update. For $\alpha = 1$ the heuristic is equivalent to the loopy max-sum algorithm.

Proof sketch The first part is trivial and follows from the fact that we move out more than in theorem 6. For $\alpha = 1$, we consider the constraint $x_1 + x_2 + x_3 = 1$. If the costs for the variables are $c_1 \geq c_2 \geq c_3$ then $r^+ = c_1$ and $r^- = c_2$. For the heuristic, the effect of the update for $\alpha = 1$ is to subtract r^- to the costs that shall be positive, and subtract r^+ to the costs that shall be negative. The net result is that the new costs after the update are $\bar{c}_1 = c_1 - c_2, \bar{c}_2 = c_2 - c_1, \bar{c}_3 = c_3 - c_1$. For max-sum, if we let a constraint component over binary variables describe the linear constraint and move in the costs c_1, c_2 and c_3 the max-marginal differences for these components give exactly the same numbers (since we from each cost shall subtract the largest of the other costs). The argument generalizes straightforwardly to other linear constraints. □

6 Applications of the in-the-middle heuristic

The in-the-middle heuristic was first developed to solve optimization problems in airline crew scheduling (Gopalakrishnan & Johnson, 2005; Andersson et al., 1998; Barnhart et al., 2003; Ernst et al., 2004). This problem concerns the the creation of a schedule with pairings (crew routes), where each pairing is a round trip of flight legs (non-stop flights) from a crew base and back again. The objective is to minimize total crew cost, subject to the restriction that every flight leg receives a crew. This is a major problem for any airline, where a 1 percent savings may amount to many millions of dollars yearly, and a successful crew system may save many times that, compared to previous practice.

The actual optimization problem is to select a subset of pairings from a large number of possible pairings. This can be modeled as an extended set covering problem (Caprara et al., 2000), where variables correspond to pairings, and constraints of the form $Ax \geq 1$ express that every flight leg requires at least one crew. Numerous additional constraints are required. some of which fall outside the form given by (ILP), and are of a more general knapsack form. Problem sizes can range up to millions of variables and tens of thousands of constraints. For this application, (Wedelin, 1995; Andersson et al., 1998; Alefragis et al., 2000) describe an optimized implementation of the in-the-middle heuristic with extensions for different kinds of constraints, known as the *paqs* optimizer. The optimizer is integrated in the Jeppesen (formerly Carmen) Crew Pairing System which is used by many of the world's large airlines. It includes an active set strategy, where iteration is more frequent within the most critical parts of the problem, and parallel computation.

The system and its optimizer has been running for more than 20 years, and the in-the-middle optimizer itself is regularly benchmarked against other state of the art integer optimizers such as CPLEX and XPRESS-MP.

The in-the-middle heuristic - in the literature sometimes referred to as the Wedelin heuristic - has also been shown to be efficient in several other contexts. In Bastert et al. (2010) several generalizations are suggested, and many implementation details are given. The paper also presents experiments on a large number of integer programming problems from mixed sources. These show that the heuristic finds comparable or better solutions than other state-of-the-art heuristics, typically needing only a fraction of their running time, and often finds integer solutions faster than what LP-algorithms require to find the LP-solution.

For a vehicle scheduling application, Ernst. et al. (2011) reports that the heuristic not only creates good schedules but also provides very good lower bounds. Other references include Atamturk et al. (1995); Mason (2002); Miura et al. (2009), who all confirm the effectiveness of the heuristic. The in-the-middle heuristic has also been implemented in commercial optimization software such as XPRESS-MP and SAS.

7 Conclusions and discussion

We have presented the in-the-middle algorithm and the in-the-middle heuristic. The former is LP-based, the latter can find non-trivial integer solutions and also significantly speed up convergence. We have provided a new analysis showing the relationship to other algorithms (GIS, max-sum diffusion, max-sum algorithm), and outlined relevant concepts for relating to the max-sum problem and generalizing to constraint component updates. Beyond specific results, we like to think that this opens up for exploring new related algorithms and relationships. (For a specific list of contributions of this paper, see the introduction.)

With respect to the overall modeling approach, we have found it useful to have linear constraints as a starting point (ILP, in-the-middle, GIS), rather than the max-sum problem, since analysis of the latter anyway seems to lead to the former. Our analysis also shows that in-the-middle and max-sum updates can be freely mixed, dissolving any sharp distinction between ILP and the max-sum problem. In Grohe & Wedelin (2007), one of the cited references contains an example of solving a crossword puzzle equivalent to a simple substitution cipher, using bigram statistics. The problem is straightforwardly modelled as a hybrid ILP/max-sum problem, and is solved with an adaptation of the in-the-middle algorithm (no heuristic

was required in this case).

We also comment on the general relationship between belief propagation and linear programming, which has been the topic of much research in recent years see e.g. Wainwright et al. (2005); Kolmogorov (2006); Wainwright & Jordan (2007). For the in-the-middle approach this relationship is obvious, with the clear separation between the non-heuristic in-the-middle algorithm (a dual LP algorithm), and the parameter-controlled in-the-middle heuristic. We have also shown that the two algorithms match with the more general concepts of non-conflicting and conflicting updates, which for the max-sum problem draws the line between exact LP algorithms and heuristics.

The power of the approach comes not only from the fact that it is LP-based, but also from the fact that the heuristic is able to find integer solutions when the LP-relaxation is fractional. This matches well with the established experience in optimization, where LP-based heuristics are common. In comparison, the loopy max-sum algorithm is already a heuristic since its updates are conflicting. This means that its LP-related aspects are not clearly visible, and we may have to handle convergence by making the updates less conflicting e.g. by some form of attenuation. However, also here, it is not just some relationship to LP that explains its well-known performance in solving difficult combinatorial problems, but also that its conflicting updates help to find integer solutions when plain LP is not sufficient.

We have finally highlighted the in-the-middle heuristic as an application of coordinate ascent/message passing in a major application of optimization and operations research, which has been in commercial use for many years. The effectiveness of the approach has been reported also in other benchmarks and applications. Depending on the application, it may be of interest to combine with other approaches, so the full usefulness cannot easily be assessed in isolation. For example, if an exact algorithm is desired, an algorithm such as the in-the-middle heuristic can be used to quickly provide both dual bounds and feasible integer solutions, improving the efficiency of a branch and bound search.

References

Alefragis, P., Sanders, P., Takkula, T., and Wedelin, D. Parallel integer optimization for crew scheduling. *Annals of Operations Research*, 99, 2000.

Andersson, E., Housos, E., Kohl, N., and Wedelin, D. Crew pairing optimization. In Yu, Gang (ed.), *Operations Research in the Airline Industry*. Lawrence Erlbaum Associates, Inc., 1998.

Atamturk, A., Nemhauser, G., and Savelsbergh, M. A combined lagrangian, linear programming, and implication heuristic for large-scale set partitioning problems. *Journal of Heuristics*, 1, 1995.

Barnhart, C., Cohn, A., Johnson, E., Klabjan, D., Nemhauser, G., and Vance, G. P. Airline crew scheduling. In *HANDBOOK OF TRANSPORTATION SCIENCE, International Series in Operations Research & Management Science*. Lawrence Erlbaum Associates, Inc., 2003.

Bastert, O., Hummel, B., and deVries, S. A generalized wedelin heuristic for integer programming. *INFORMS Journal on Computing*, 22, 2010.

Bertele, U. and Brioschi, F. *Nonserial Dynamic Programming*. Academic Press, 1972.

Caprara, A., Toth, P., and Fischetti, M. Algorithms for the set covering problem. *Annals of Operations Research*, 98, 2000.

Cooper, M. Virtual arc consistency for weighted csp. In *Proc. 23rd AAAI Conference on Artificial Intelligence*, 2008.

Cooper, M., de Givry, S., and Schiex, T. Optimal soft arc consistency. In *Proc. IJCAI*, Hyderabad, India, 2000.

Darroch, J. N. and Ratcliff, D. Generalized iterative scaling. *The Annals of Mathematical Statistics*, 53(5), 1972.

Ernst, A., Jiang, H., Krishnamoorthy, M., Owens, B., and Sier., D. An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 2004.

Ernst., A., Gavriliouk, E., and Marquez, L. An efficient lagrangean heuristic for rental vehicle scheduling. *Computers and Operations Research*, 38, 2011.

Fienberg, S. E. An iterative procedure for estimation in contingency tables. *The Annals of Mathematical Statistics*, 41(3), 1970.

Gopalakrishnan and Johnson, E. Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, 140, 2005.

Grohe, B. and Wedelin, D. Cost propagation: numerical propagation for optimization problems. *LNCS*, 5015, 2007.

Kolmogorov, V. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28, 2006.

Mason, A. Elastic constraint branching, the wedelin/carmen lagrangian heuristic and integer programming for personnel scheduling. *Annals of Operations Research*, 108, 2002.

Miura, R., Imaizumi, J., Fukumura, N., and Morito, S. An application of wedelin's method to railway crew scheduling problem. *IEEJ Transactions on Electronics, Information and Systems*, 129, 2009.

Schiex, T., Fargier, H., and Verfaillie, G. Valued constraint satisfaction problems: Hard and easy problems. In *Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, Montreal, 1995.

Wainwright, M. and Jordan, M. Graphical models, exponential families, and variational inference. In *Foundations and Trends in Machine Learning*, volume 1. Lawrence Erlbaum Associates, Inc., 2007.

Wainwright, M., Jaakkola, T., and Willsky, A. Map estimation via agreement on hypertrees: message passing and linear programming approaches. *IEEE Trans. Information Theory*, 51, 2005.

Wedelin, D. Probabilistic inference and combinatorial optimization. Technical report, Department of Computer Sceince, Chalmers University of Technology, 1989.

Wedelin, D. An algorithm for large scale 0-1 integer programming with application to airline crew scheduling. *Annals of Operations Research*, 57, 1995.

Werner, T. A linear programming approach to max-sum problem: A review. *IEEE Trans. PAMI*, 29, 2007.