The Alternating Directions Method of Multipliers as a Message-Passing Algorithm

Jonathan Yedidia
Disney Research Boston

This work is based on a collaboration about the difference-map and "Divide and Concur" algorithms with Veit Elser (Cornell)

It was inspired by the paper by Boyd, et.al. (2010) on the ADMM algorithm, which dates back to Gabay & Mercier (1976), Glowinski & Marrocco (1975).

Boyd et.al.:

"While we have emphasized applications that can be concisely explained, the algorithm would also be a natural fit for more complicated problems in areas like graphical models."

A general optimization problem

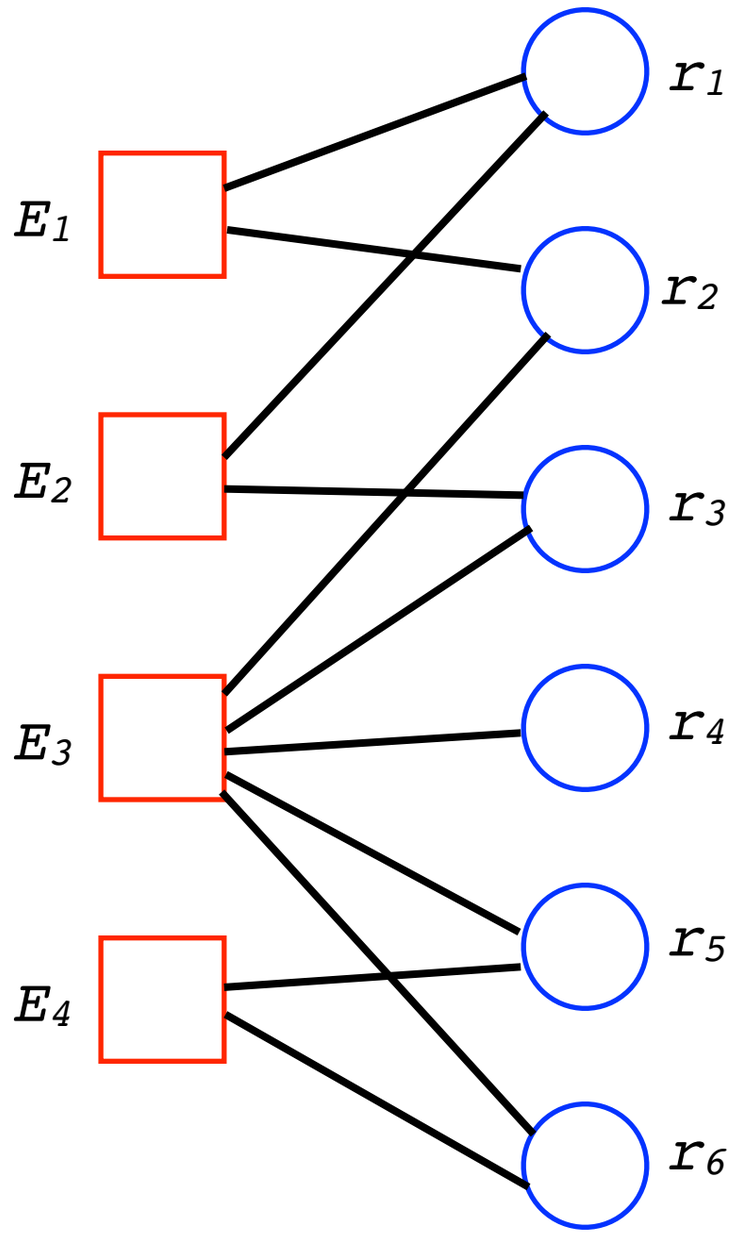
$$\boxed{\begin{array}{c}\text{minimize } E_0(r) \text{ with } r \in \mathbb{R}^n \\ \text{subject to } k \text{ constraints on } r\end{array}}$$

is equivalent to

$$\boxed{\text{minimize } \sum_{a=0}^{k} E_a(r) \text{ with } r \in \mathbb{R}^n}$$

using infinite-cost functions to enforce hard constraints.

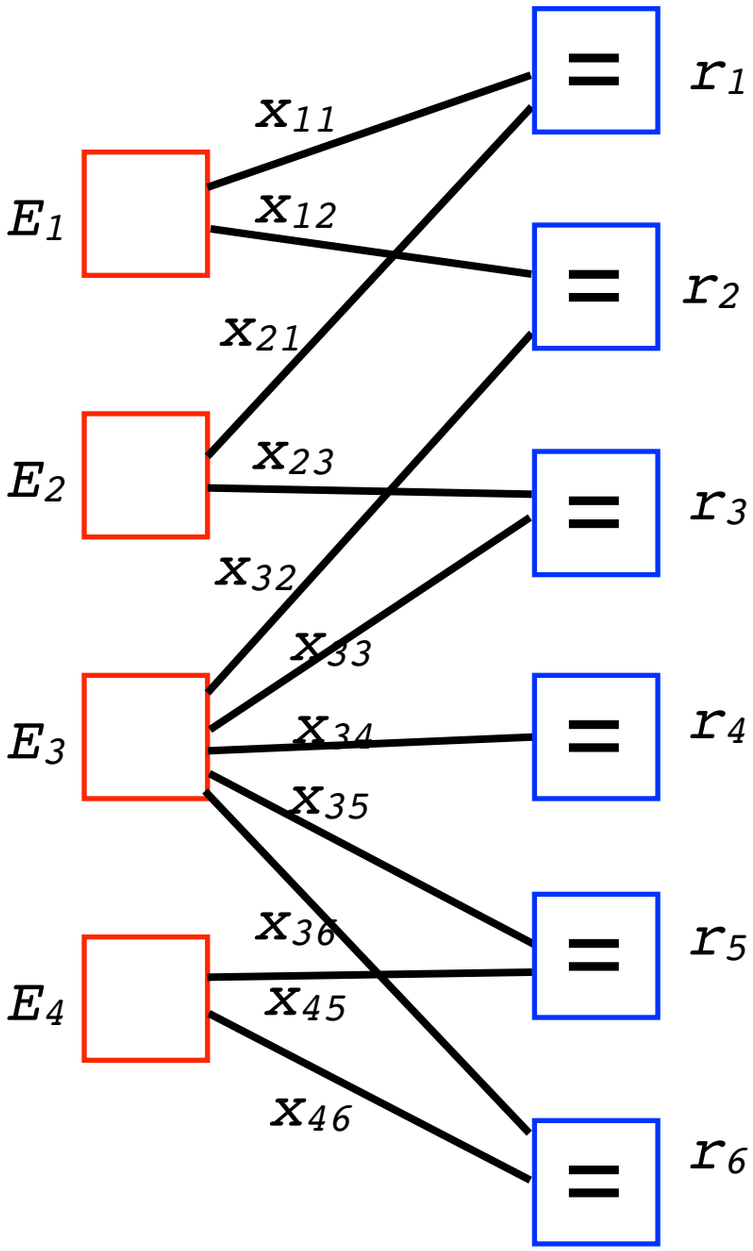
The "Factor Graph"
representation of $E(r) = \sum_{a=1}^{m} E_a(r)$ looks like



m factor nodes

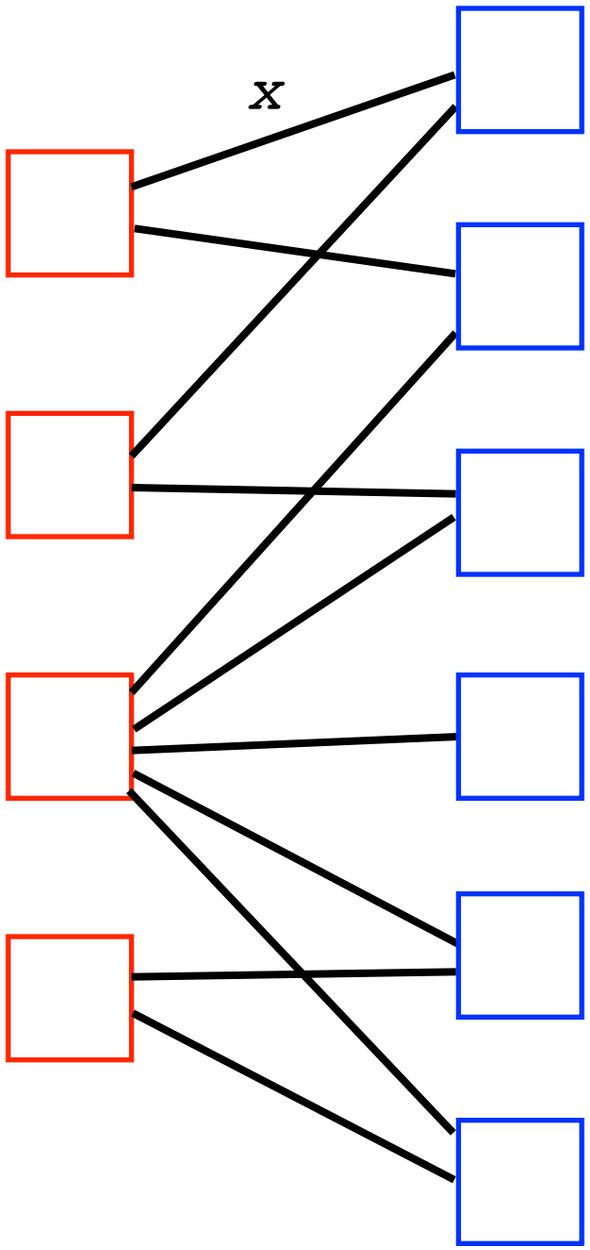
n variable nodes

In Forney's equivalent "normal" representation, variables live on edges.



Equality nodes enforce equality of neighboring edges.

So any optimization problem can be mapped onto an equivalent problem on a bipartite Forney factor graph.

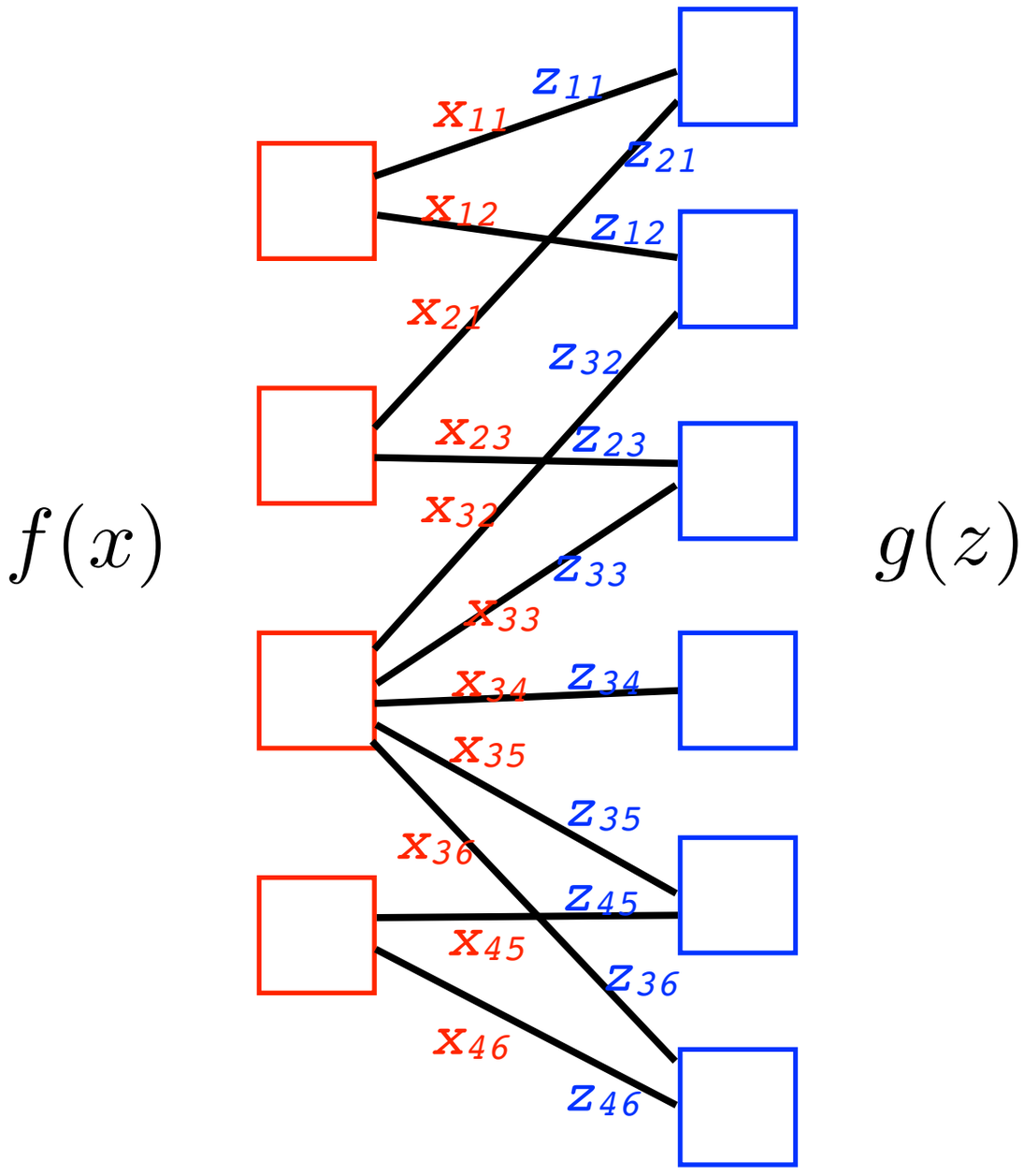


$$\text{minimize } E_0(r) \text{ with } r \in \mathbb{R}^n$$
$$\text{subject to } k \text{ constraints on } r$$

$$\text{minimize } E(x) = \sum_a E_a(x)$$

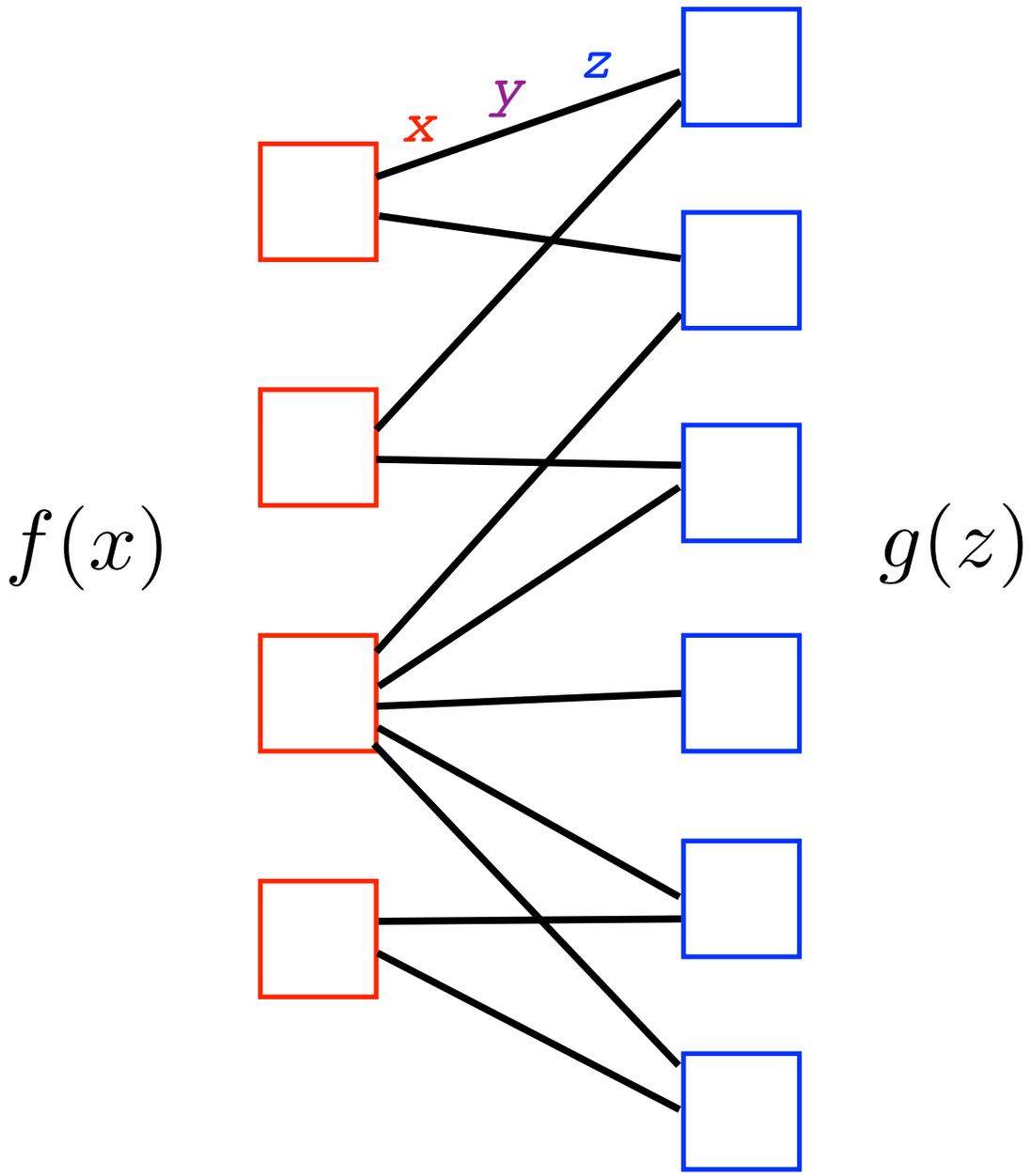
Let's make a copy z of each edge variable *x*.
So an equivalent problem is

$$\text{minimize } f(x) + g(z) \text{ subject to } x = z$$



To solve it, we introduce a Lagrangian.

$$L(x, y, z) = f(x) + g(x) + y \cdot (x - z)$$

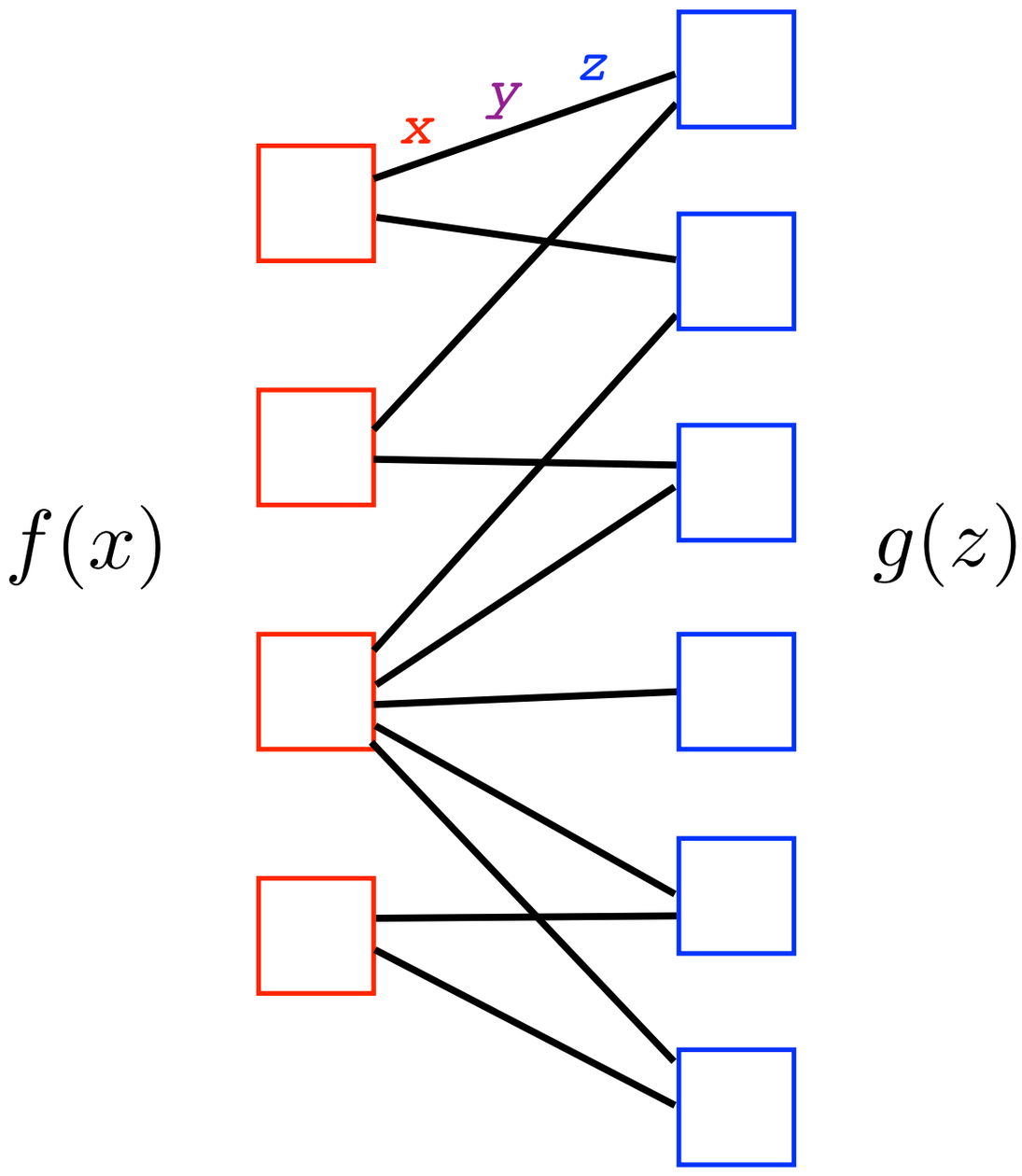


Assume convexity of $f(x)$ and $g(z)$, and add a "penalty term" to get strict convexity.

penalty term

$$L(x, y, z) = f(x) + g(x) + y \cdot (x - z) + (\rho/2)(x - z)^2$$

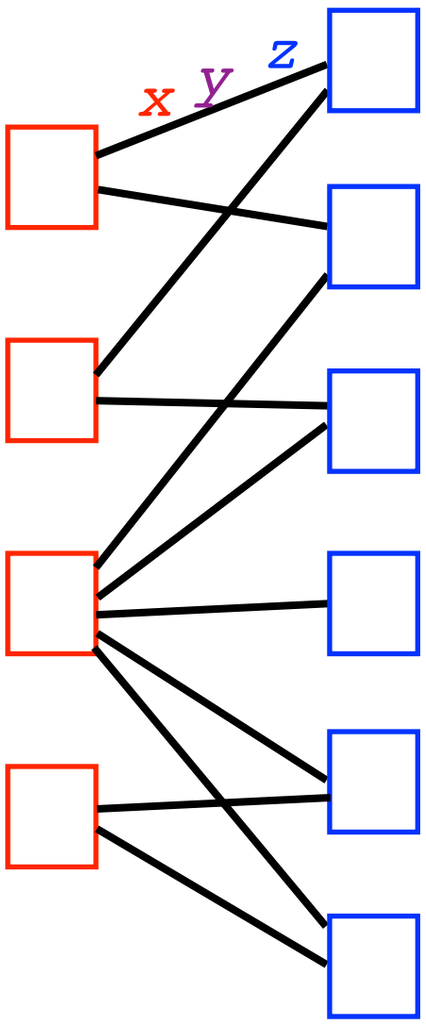
scalar parameter



The penalty term doesn't change the optimum.

Here's the dual problem:

$$\text{maximize } h(y) = \underset{x,z}{\text{argmin}}\, L(x, y, z)$$



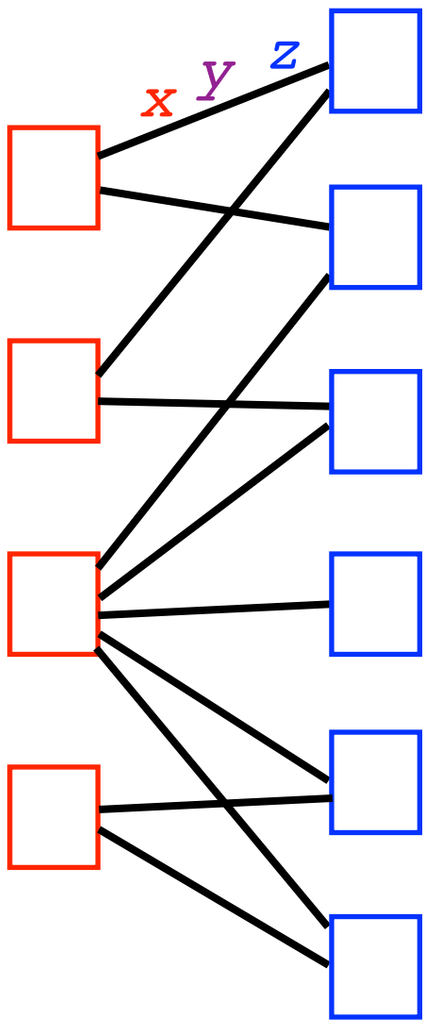
We can use gradient ascent to solve the dual problem :

Repeat:

1. $(x^{t+1}, z^{t+1}) = \underset{x,z}{\text{argmin}}\, L(x, y^t, z)$
2. $y^{t+1} = y^t + \alpha \frac{\partial h}{\partial y} = y^t + \alpha(x^{t+1} - z^{t+1})$

The choice $\alpha = \rho$ is normally made as it enforces dual feasibility of the iterates.

The "Alternating Direction Method of Multipliers" (ADMM) takes advantage of the $f(x) + g(z)$ structure to decouple the computations of *x* and *z*.

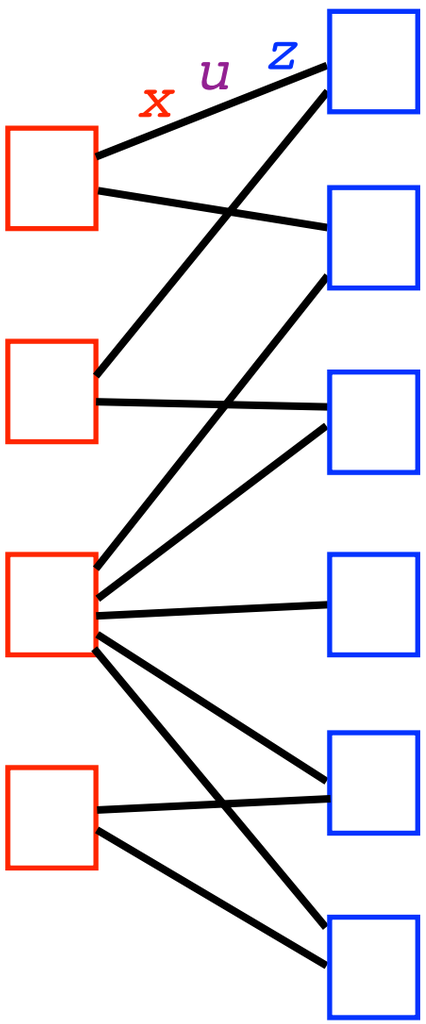


Repeat:

1. $x^{t+1} = \underset{x}{\operatorname{argmin}}\, L(x, y^t, z^t)$
2. $y^{t+1} = y^t + \rho(x^{t+1} - z^t)$
3. $z^{t+1} = \underset{z}{\operatorname{argmin}}\, L(x^{t+1}, y^{t+1}, z)$

If *f(x)* and *g(z)* are convex, ADMM is guaranteed to converge to the optimum.

We can rescale $u = y/\rho$
and obtain the explicit form:



Repeat:

1. $x^{t+1} = \underset{x}{\text{argmin}} \left[f(x) + (\rho/2)(x - z^t + u^t)^2 \right]$
2. $u^{t+1} = u^t + x^{t+1} - z^t$
3. $z^{t+1} = \underset{z}{\text{argmin}} \left[g(z) + (\rho/2)(z - x^{t+1} - u^{t+1})^2 \right]$

Notice that *u* is a running sum of the difference between *x* and *z*; it keeps adjusting to try to equalize them.

The ADMM algorithm was derived and proven to converge to the optimum for convex *f(x)* and *g(z)*, and is a state-of-the-art algorithm for many convex optimization problems.

Repeat:

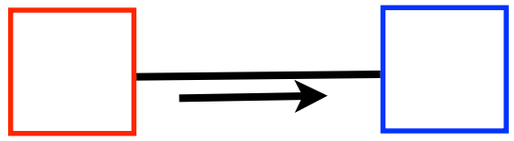
1. $x^{t+1} = \underset{x}{\operatorname{argmin}} \left[f(x) + (\rho/2)(x - z^t + u^t)^2 \right]$
2. $u^{t+1} = u^t + x^{t+1} - z^t$
3. $z^{t+1} = \underset{z}{\operatorname{argmin}} \left[g(z) + (\rho/2)(z - x^{t+1} - u^{t+1})^2 \right]$

But it is actually a well-defined algorithm for any *f(x)* and *g(z)* that are bounded below, and can serve as a powerful heuristic algorithm for non-convex optimization problems.

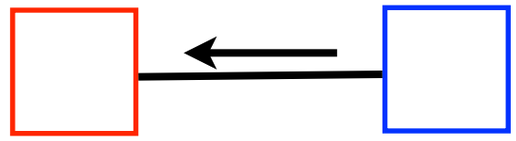
It will be useful to build some intuition.

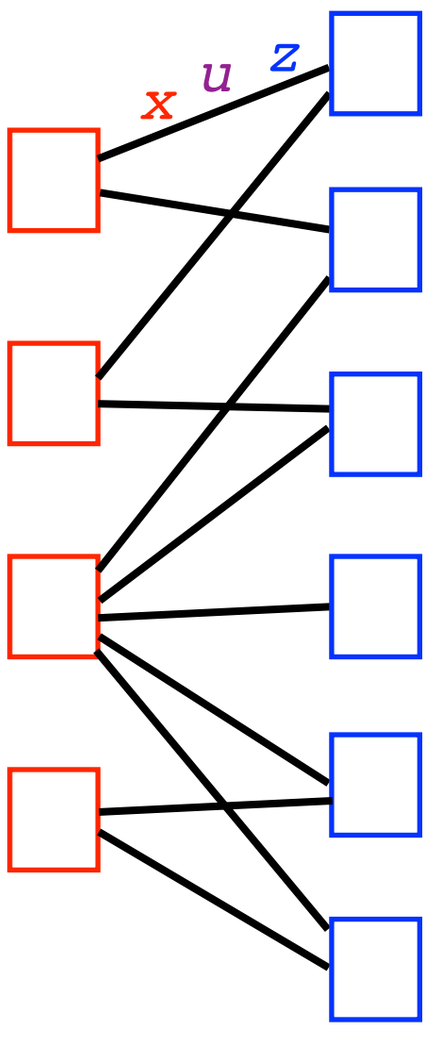
We can introduce "messages."

$$m^t = x^t + u^t$$



$$n^t = z^t - u^t$$





Repeat:

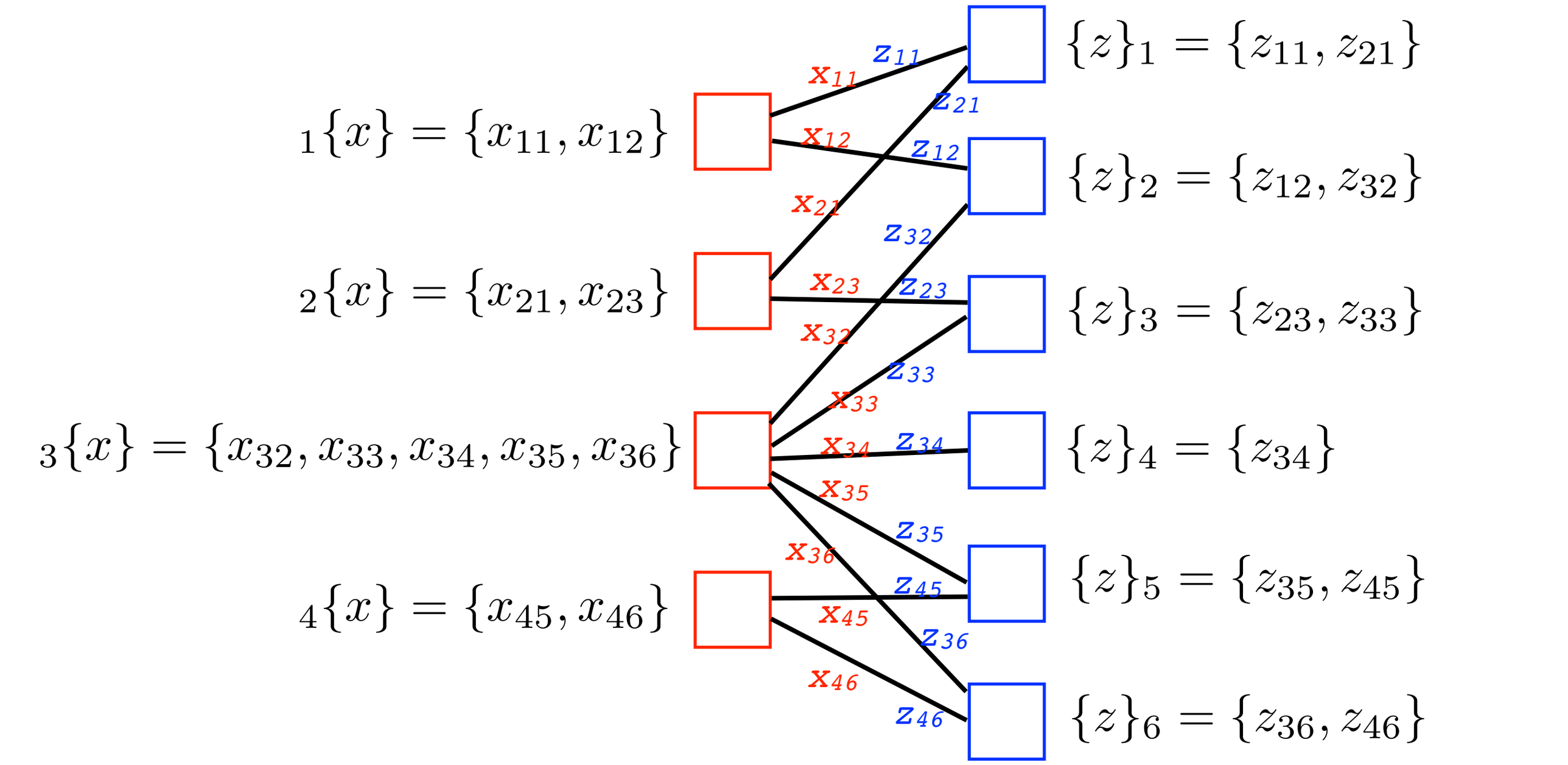
1. $x^{t+1} = \underset{x}{\operatorname{argmin}} \left[f(x) + (\rho/2)(x - n^t)^2 \right]$

2. $u^{t+1} = u^t + x^{t+1} - z^t$

3. $z^{t+1} = \underset{z}{\operatorname{argmin}} \left[g(z) + (\rho/2)(z - m^{t+1})^2 \right]$

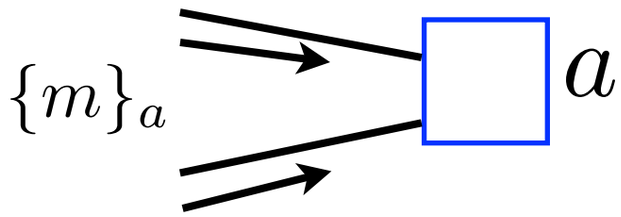
The computation of "beliefs" *x* and *z* is done locally at the factor nodes using "messages" *m* and *n*.

It is useful to introduce some (non-standard) notation.



$$x = {}_1\{x\} \oplus {}_2\{x\} \oplus {}_3\{x\} \oplus {}_4\{x\} \qquad z = \{z\}_1 \oplus \{z\}_2 \oplus \ldots \oplus \{z\}_6$$

The local computation



$$\{z\}_a^{t+1} = \underset{\{z\}_a}{\operatorname{argmin}} \left[g_a(\{z\}_a) + (\rho/2)(\{z\}_a - \{m\}_a^{t+1})^2 \right]$$

balances the desire to minimize the local part of the *g(z)* function with the desire to agree with the messages coming from other nodes.

The ρ parameter lets us vary the relative strength of these competing influences.

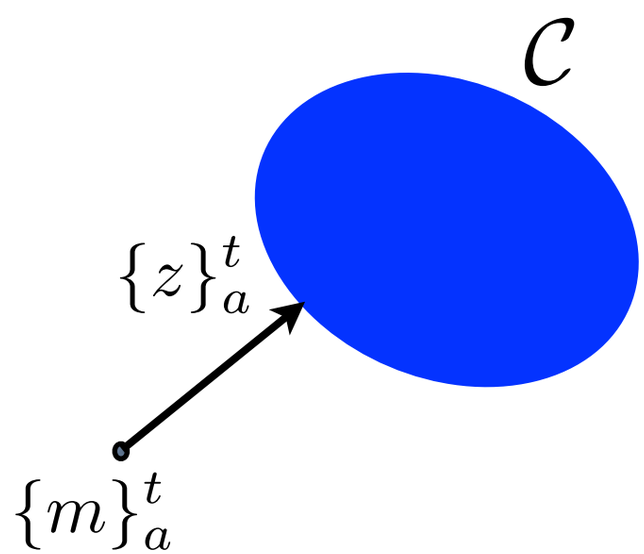
When a local function $f_a({}_a\{x\})$ or $g_a(\{z\}_a)$ is a hard constraint, e.g.:

$$g_a(\{z\}_a) = 0 \text{ for } \{z\}_a \in \mathcal{C}$$
$$g_a(\{z\}_a) = \infty \text{ for } \{z\}_a \notin \mathcal{C}$$

then the local computation turns into a projection onto the constraint set $\mathcal{C}$.

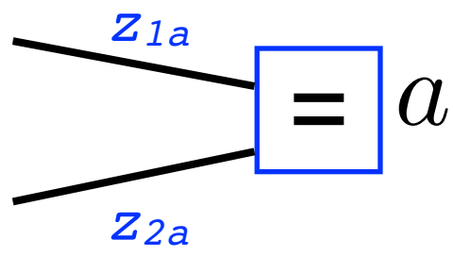
$$\{z\}_a^t = \underset{\{z\}_a}{\text{argmin}} \left[g_a(\{z\}_a) + (\rho/2)(\{z\}_a - \{m\}_a^t)^2 \right]$$

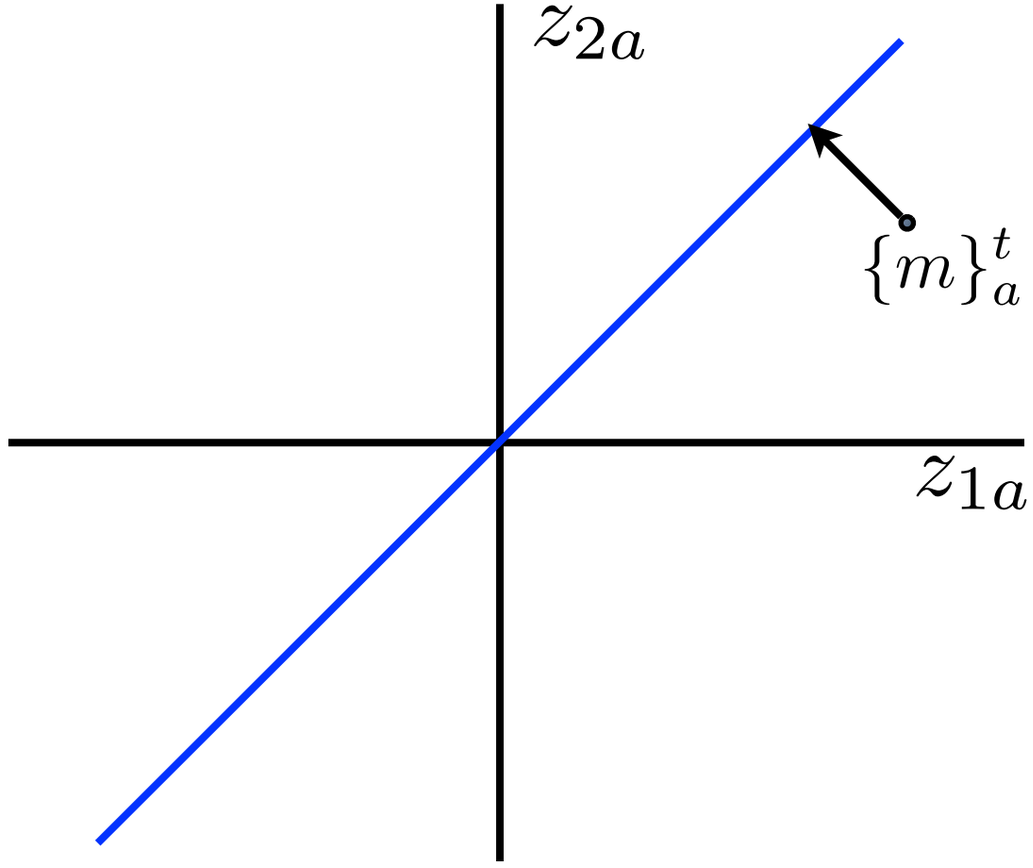
$$\{z\}_a^t = P_{\mathcal{C}}(\{m\}_a^t)$$



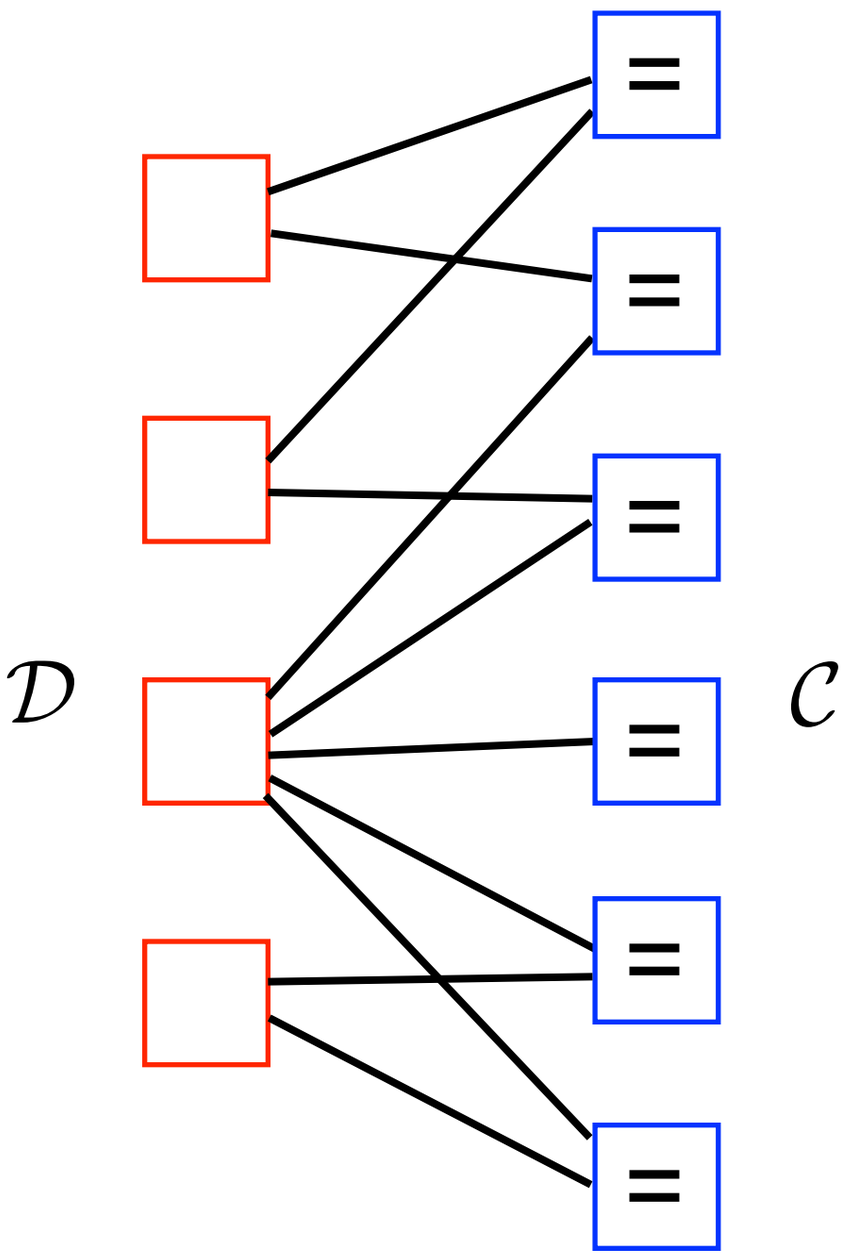
In particular, an equality node is a (convex) hard constraint and the projection will give the mean:

$$z_{ia}^t = \langle \{m\}_a^t \rangle \text{ for all } i$$





For a constraint satisfaction problem asking for a solution satisfying a number of hard constraints, ADMM on a factor graph reduces to the "Divide and Concur" (DC) algorithm.



Repeat:

1. $m^{t+1} = 2P_{\mathcal{D}}(n^t) - n^t$
2. $n^{t+1} = n^t + P_{\mathcal{C}}(m^{t+1}) - P_{\mathcal{D}}(n^t)$

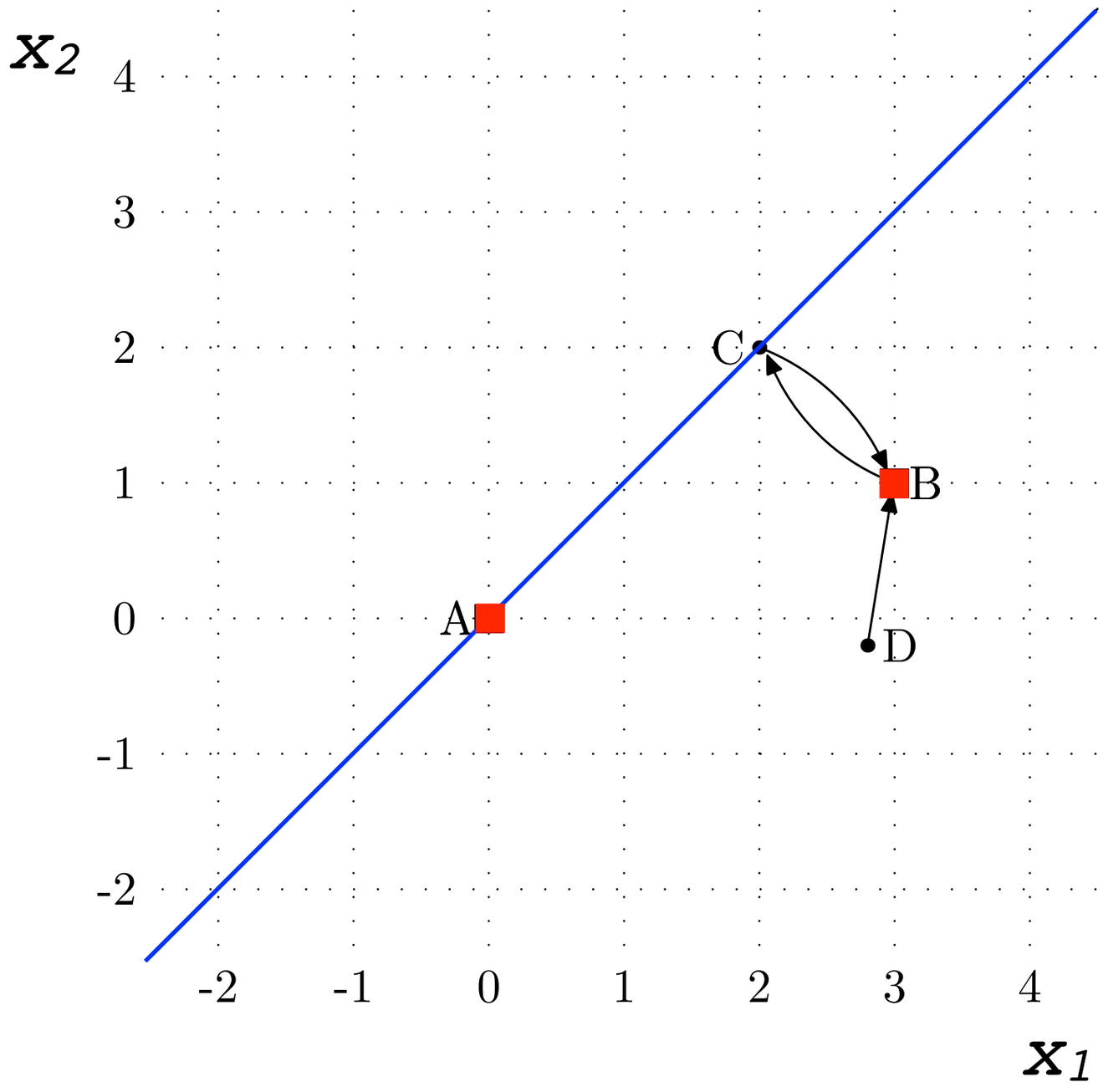
or equivalently

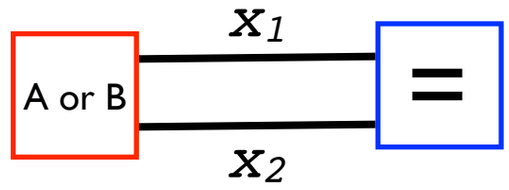
Repeat:

$$n^{t+1} = P_{\mathcal{C}}(2P_{\mathcal{D}}(n^t) - n^t) - (P_{\mathcal{D}}(n^t) - n^t)$$

DC is not guaranteed to converge for non-convex problems, but if it does converge, it is guaranteed to have found a solution.

DC avoids the simple traps that cause problems for the naive alternating-projections algorithm.



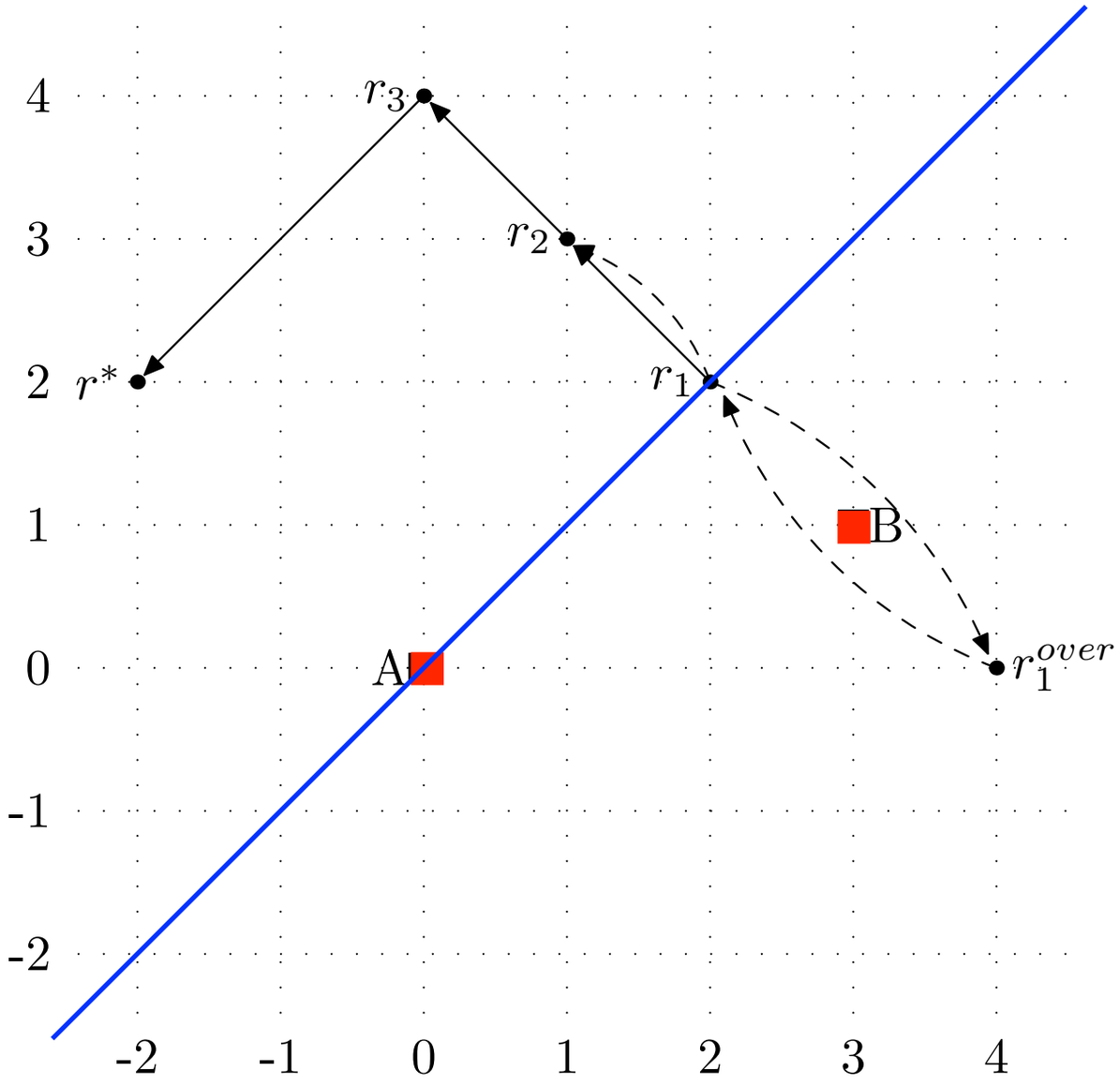


Alternating Projections:

Repeat:

$$x^{t+1} = P_{\mathcal{C}}(P_{\mathcal{D}}(x^t))$$

DC's "difference-map" dynamics turns simple traps into "repellers." It also converges much faster than alternating projections for many convex problems.



t	$\boldsymbol{r}_t$	$P_D(\boldsymbol{r}_t)$	$\boldsymbol{r}_t^{over}$	$\boldsymbol{r}_t^{conc}$
1	$(2,2)$	$(3,1)$	$(4,0)$	$(2,2)$
2	$(1,3)$	$(3,1)$	$(5,-1)$	$(2,2)$
3	$(0,4)$	$(0,0)$	$(0,-4)$	$(-2,-2)$
4	$(-2,2)$	$(0,0)$	$(2,-2)$	$(0,0)$
5	$(-2,2)$			

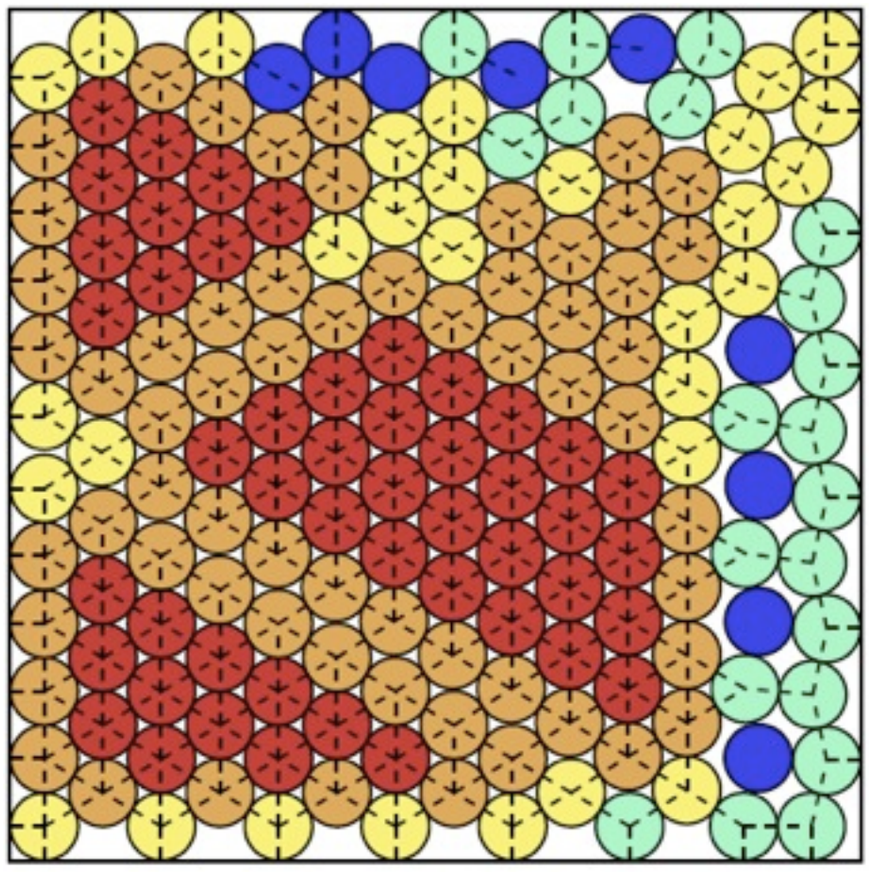
Difference-map

Repeat:

$$r_{t+1} = P_{\mathcal{C}}(r_t + 2(P_{\mathcal{D}}(r_t) - r_t)) - (P_{\mathcal{D}}(r_t) - r_t)$$

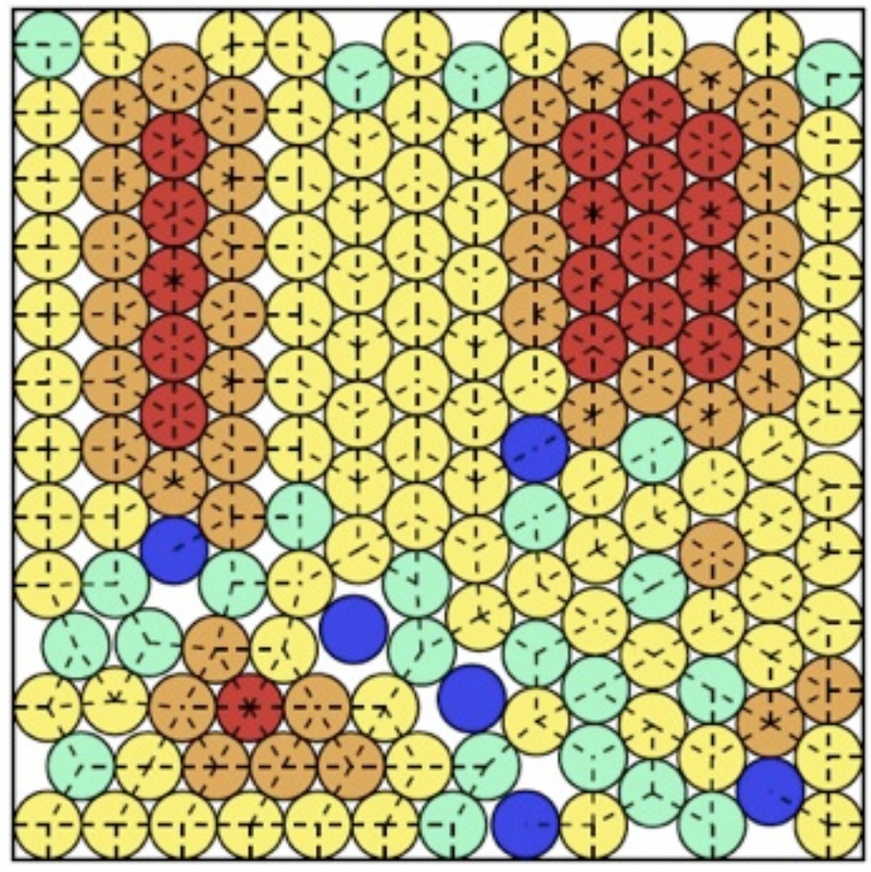
The DC algorithm provides state-of-the-art results for many non-convex constraint-satisfaction problems, including e.g., a variety of packing problems.

(from S. Gravel Ph.D. thesis, 2009)



density = 0.8393

previous best packing of 169 spheres



density = 0.8399

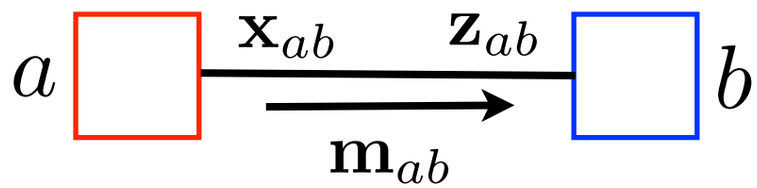
improved packing found using DC

The packing example illustrates several advantages of DC and ADMM algorithms compared with belief propagation (BP) algorithms:

1. They deal very efficiently with continuous variables.
2. They easily handle unusual hard constraints.
3. They do not need local evidence, and cannot converge to non-informative fixed-points or pseudo-solutions.

Like sum-product BP, these message-passing algorithms can be derived from a "variational" argument, but now we are directly optimizing the energy function instead of the Bethe approximation to the free energy.

It often makes sense to represent a q-ary discrete variable using q indicator variables. This turns all the variables and messages on an edge into q-ary vectors.



$$\mathbf{z}_{ab} = (z_{ab}^{(1)}, z_{ab}^{(2)}, \ldots, z_{ab}^{(q)})$$

The fact that exactly one indicator variable should be equal to one and the rest equal to zero per edge will be enforced as a hard constraint.

This representation in an ADMM algorithm for a discrete-variable problem will give nearly identical memory requirements for messages and beliefs as in a BP algorithm. Analogously to BP, one can loosely interpret the vectors as probability distributions.

Such a representation was used by Elser in a DC algorithm to solve 25x25 Sudoku problems that are beyond the reach of other (back-tracking) algorithms.

									M	W	B	P	E	U	K	Y	A	X	F	T	D	R	I	V
I		W	O	U	L	D				H	M	V	F	N	Q	T	J	B	E	A	C	S	G	Y
		D	E	S	I	G	N									P	L	M	U	X	F	K	B	W
								A		G	R	I	D							L	Q	E	M	N
											L	A	C	K	I	N	G							
															D	U	P	L	I	C	A	T	E	S
		I	N		E	A	C	H																
S						R	O	W						E	A	H	C	Q	V	N	G	B	D	X
H	P	A	D					B	L	O	C	K			E	G	F	Y	N	M	U	J	W	I
E	Q	C	X	B	G					A	N	D								F	L	Y	H	O
V	W	J	G	T	S	B	F					C	O	L	U	M	N			I	E	Q	K	H
R	X	B	S	F	D	P	K	N	J											O	M	W	A	C
Y	A	U	K	D	O	Q	I	L	C	M	W	R	N	B	F	J	E	H	S	V	P	G	X	T
Q	O	P	C	L	H	M	E	G	Y	T	K	F	A	V	W	B	I	D	X	U	J	N	S	R
N	H	M	I	E	V	W	A	T	U	Q	X	J	S	G	P	O	K	C	R	Y	B	F	L	D
C	S	L	M	H	J	N	D	I	P	K	G	U	T	R	X	F	O	V	B	E	W	A	Y	Q
W	T	G	R	N	X	V	U	M	O	E	P	B	J	D	H	A	Y	K	Q	S	I	C	F	L
P	D			X	W	E	T	F					L	C	G	I					N			
U											Q		W		T					R				
O	V				B					I										H		U		
G	L	V		P	F	T		U		R	H		X	W	Y	Q								
	M		Q				L	K	G	J	U			O		R	V							
	U			R	Q	I			S			N												
X		E	Y		M																			

Like other algorithms based on gradient descent, the ADMM and DC algorithms are not scale-invariant. This is reflected in the ρ parameter, which could be turned into a vector parameter, with a different value for each edge.

For DC algorithms, the effect of changing the ρ vector will be to change the metric by which projections are measured.

Like other gradient-descent algorithms, convergence rates will be improved by setting the scales such that all the variables have a similar variance in the dynamics. An algorithm that automatically adjusted its scales could be very useful.

There are many possible variants of these algorithms. Consider a DC algorithm described using messages:

Repeat:

1. $m^{t+1} = 2P_{\mathcal{D}}(n^t) - n^t$
2. $n^{t+1} = n^t + P_{\mathcal{C}}(m^{t+1}) - P_{\mathcal{D}}(n^t)$

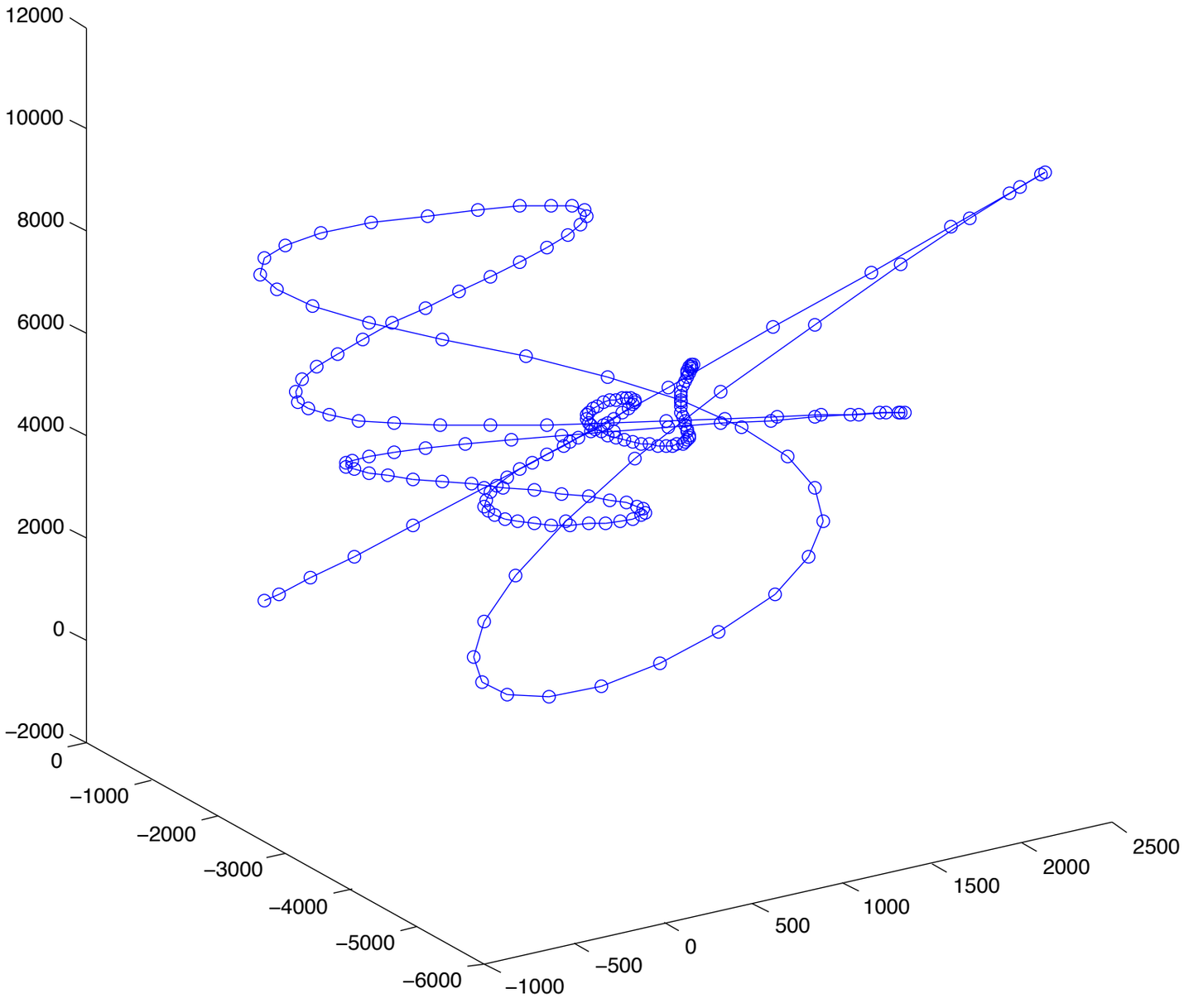
A straightforward generalization (suggested by Elser) is:

Repeat:

1. $m^{t+1} = P_{\mathcal{D}}(n^t) + \gamma(P_{\mathcal{D}}(n^t) - n^t)$
2. $n^{t+1} = n^t + \beta(P_{\mathcal{C}}(m^{t+1}) - P_{\mathcal{D}}(n^t))$

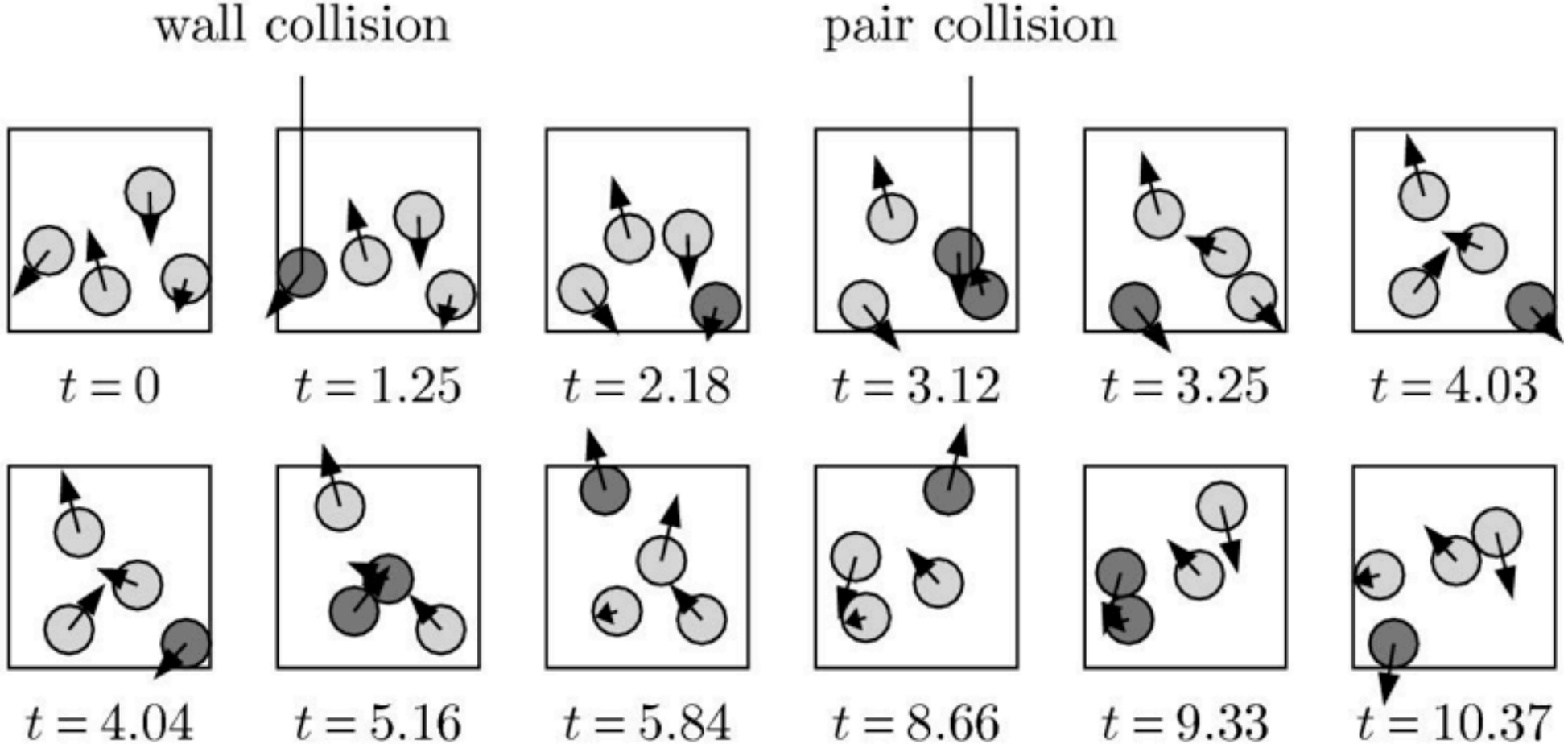
The limit of small β gives a flow (differential) limit. Such a limit also exists for ADMM algorithms.

For convex problems, DC and ADMM converge along a smooth, though possibly highly intricate path.



Three-dimensional slice of a 15-dimensional (β=1) DC message trajectory for an intersection of polyhedron problem

For non-convex problems with discrete variables, the flow limit often results in message trajectories that follow straight line paths punctuated by jumps. For some problems, these jump points can be calculated, leading to the possibility of highly efficient event-driven implementations, analogous to those used in hard-sphere simulations.



See W. Krauth, "Statistical Mechanics: Algorithms and Computations," 2006

ADMM-based message-passing algorithms have many promising potential applications, including for example machine learning, computer vision, control, and protein folding. In these areas, one typically needs to optimize complicated functions over many continuous variables. The naturally parallel and distributed nature of these algorithms means they fit well with modern multi-core and cloud-computing trends.

These algorithms also provably converge to the correct solution for convex problems, although the convergence rate is sometimes rather slow. Improved convergence rates might be attained by approaches which vary the scaling parameter(s) temporally or spatially, or by event-driven implementations.