Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions

David Eppstein

Dept. Information and Computer Science
Univ. of California, Irvine

http://www.ics.uci.edu/∼eppstein/

Outline

I. Applications

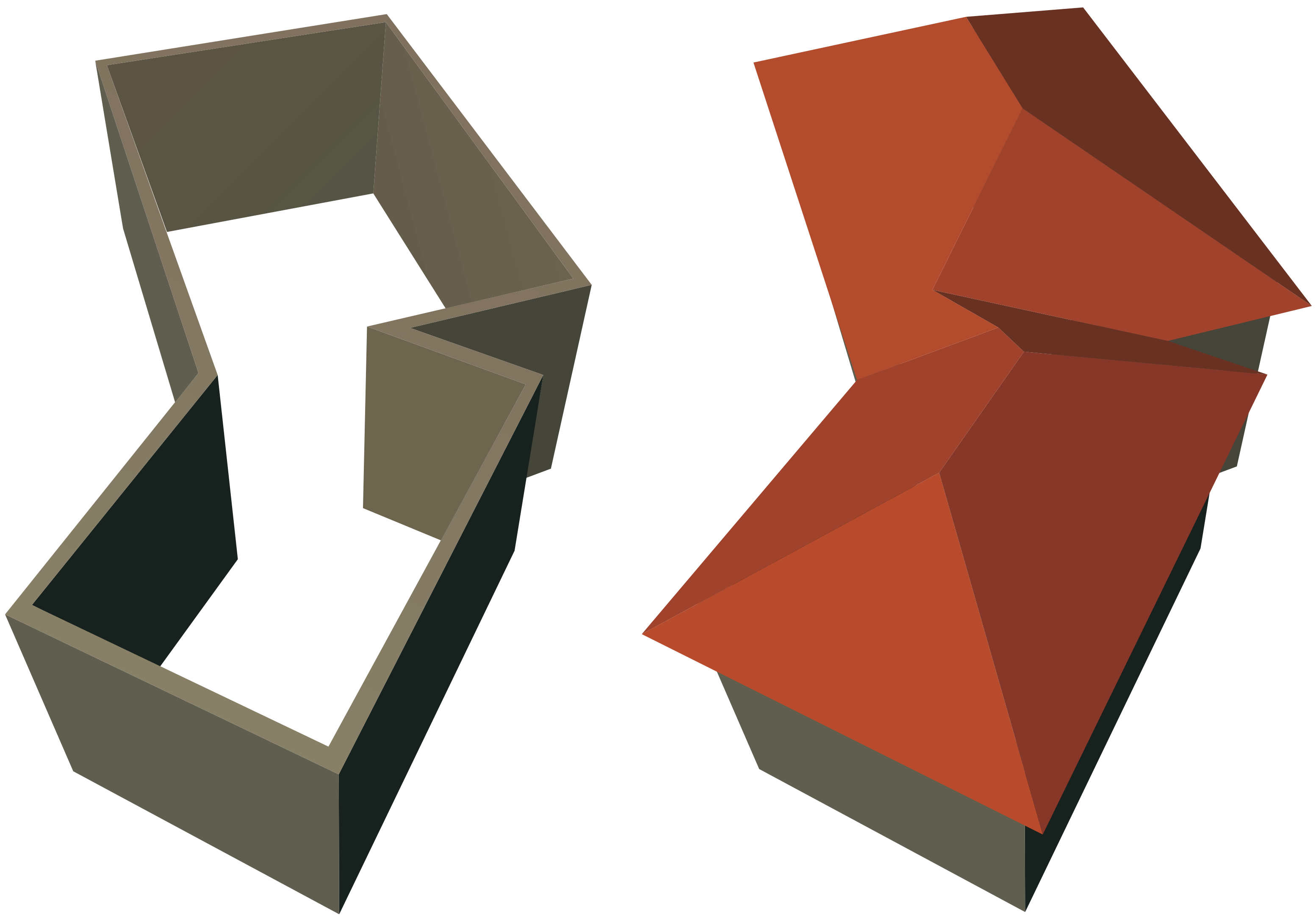
- i) Architectural design
- ii) Collision detection with moving objects
- iii) Parts layout optimization
- iv) Greedy matching (hiring decisions)

II. Formalization

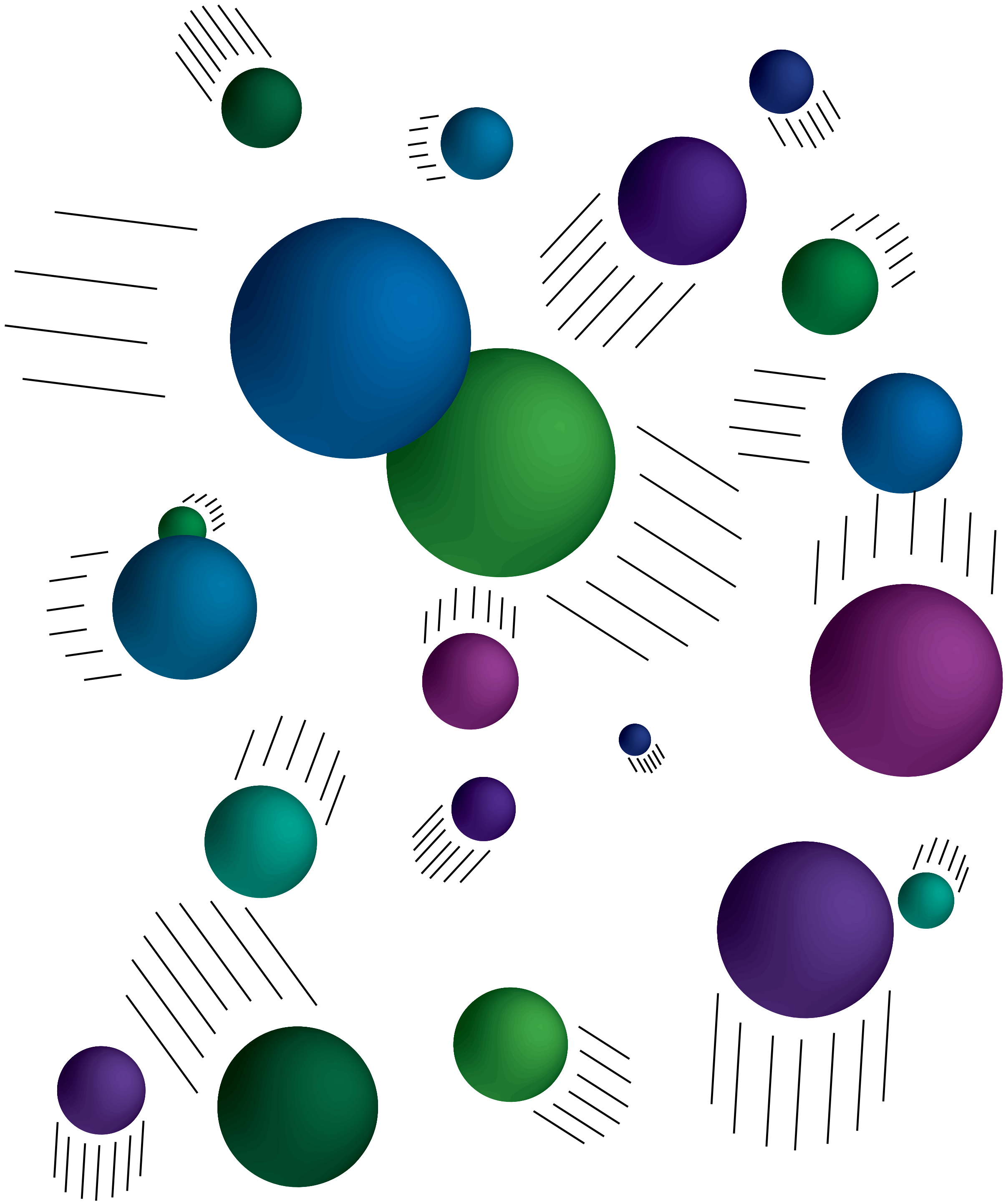
III. Results and comparison with other methods

IV. The data structure

How to fit a roof to these walls?

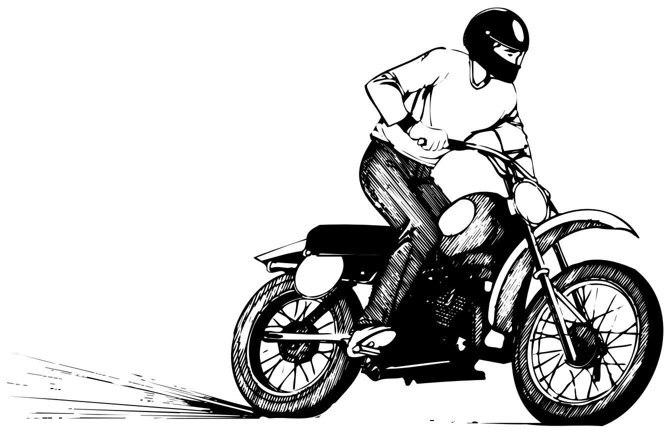


Which two particles collide next?

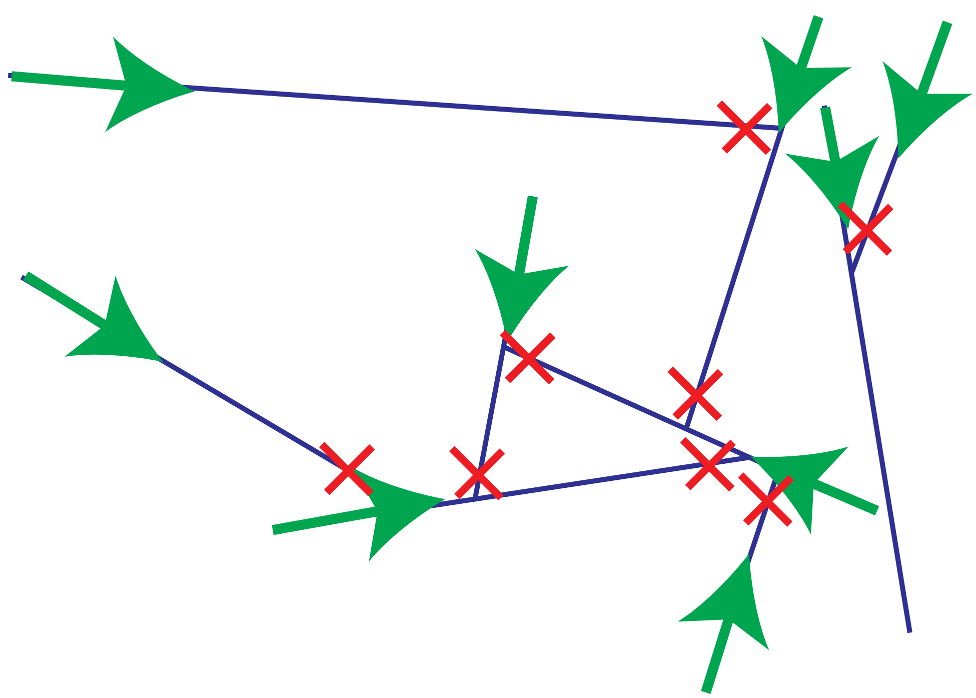


TRON Motorcycle Game

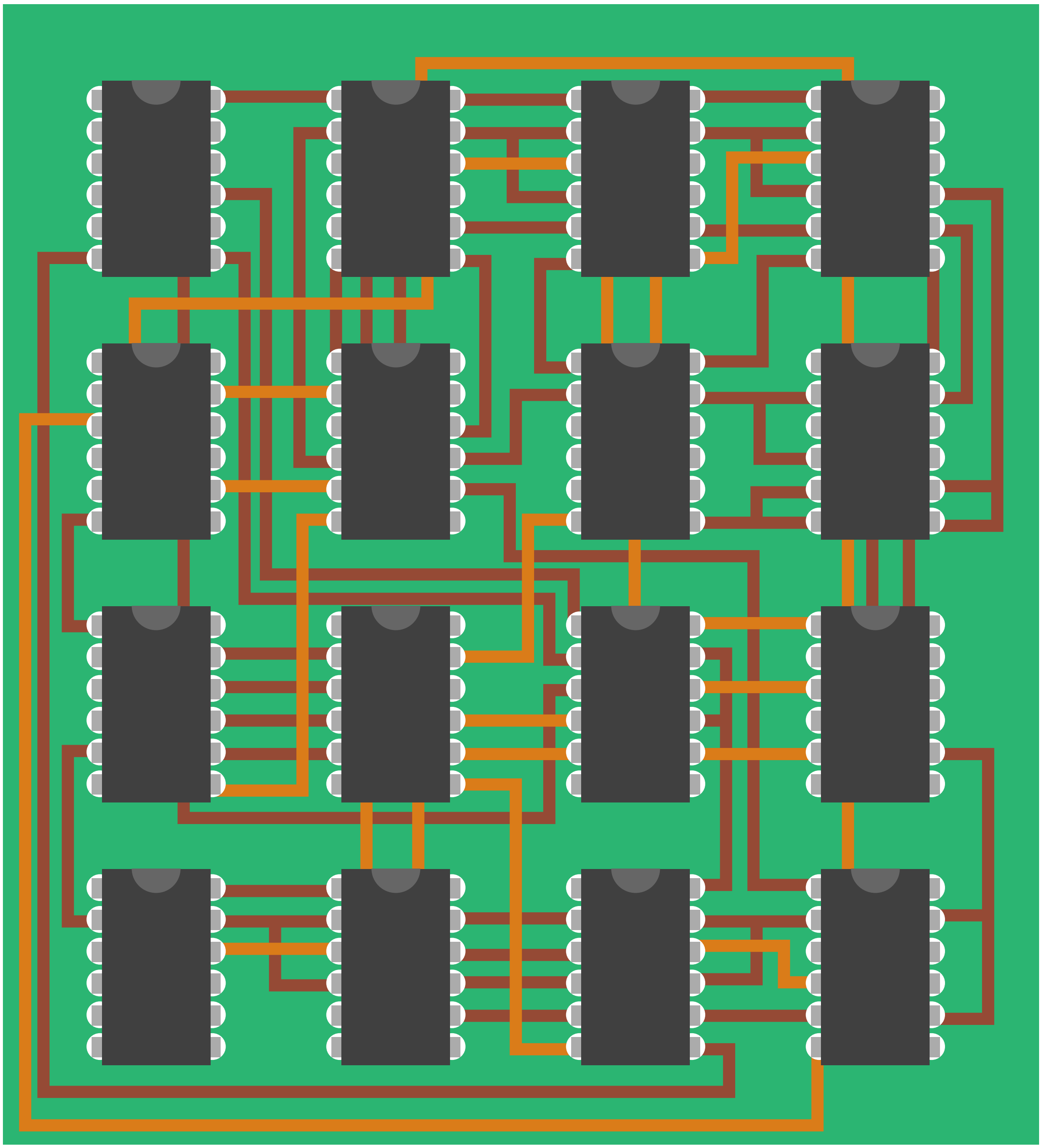
Cycles crash when they cross others' paths



Which cycle stays up longest?



Which two parts should trade places?



Which applicants to assign to which jobs?

Given multiple job applicants, multiple job openings, measure of how well applicant fits an opening

Greedy matching:

- repeatedly choose the best fitting pair of applicant and position
- remove that pair from the lists
- continue with remaining applicants and openings until all jobs filled or all applicants gone

(More complicated algorithms are possible but require more information than just ordering on pairs.)

Formalization

Given two sets S and T, undergoing additions, removals, and modifications of objects

and given a binary function $f(s, t)$

maintain the pair $(s \in S, t \in T)$
that minimizes the value $f(s, t)$

(Why two sets instead of just one?

Needed for roof design, hiring applications

Needed in definition of data structure)

Application of formalization

Collision detection:

$S = T =$ all objects
$f =$ time to collision of pair ($f(s, s) = +\infty$)
modification = new motion after collision

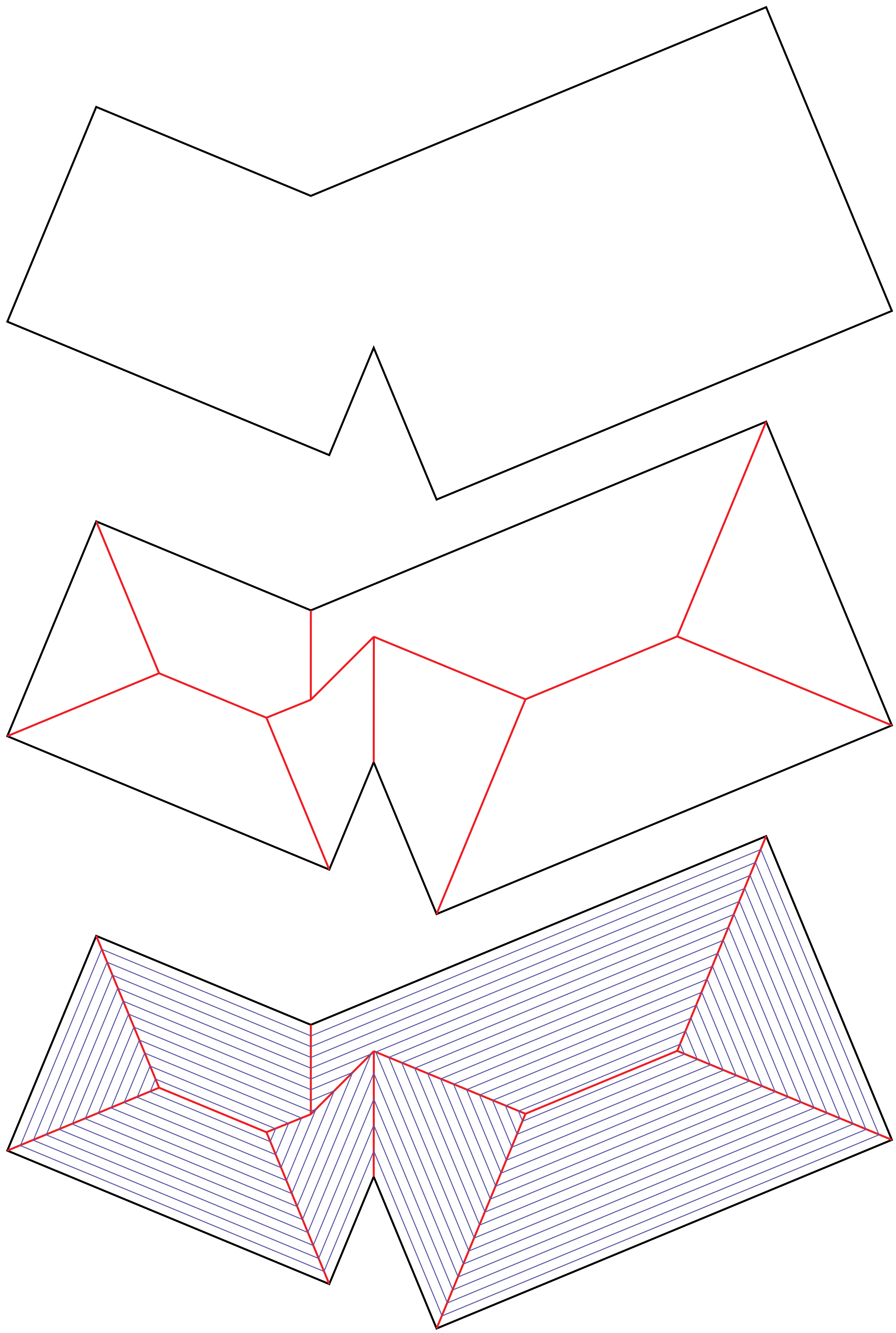
Layout optimization:

$S = T =$ all parts
$f =$ quality improvement from swapping pair
modification = change f for neighboring parts

Roof design:

$S =$ edges of offset polygon
$T =$ vertices of offset polygon
$f =$ offset depth at which vertex meets edge
modification = topology change in offset polygon

Straight Skeleton and Offset Curves



Previous approaches I: Brute force

After each update, compare all pairs (s,t) to find the pair minimizing $f(s,t)$

Advantage: no extra storage

Disadvantage: slow ($O(n^2)$ per update)

Previous approaches II: Discrete event simulation

Maintain priority queue of all pairs

After update, change n queue entries

Advantage: relatively fast ($O(n \log n)$ per update)

Disadvantage: uses too much memory ($O(n^2)$)

Previous approaches III: Physical modeling

Divide time and space up into discrete units

Test all pairs of objects within the same region for any given time step

Advantage: take advantage of geometric structure

Disadvantages:

- Only applies to objects in motion (not parts layout)
- Inefficient when objects move around a lot between collisions
- Hard to choose time/space subdivisions
- No good worst-case performance bounds

New results

Relatively simple data structure

$O(n)$ space

$O(n \log^2 n)$ time per update
(worst-case; average may be smaller)

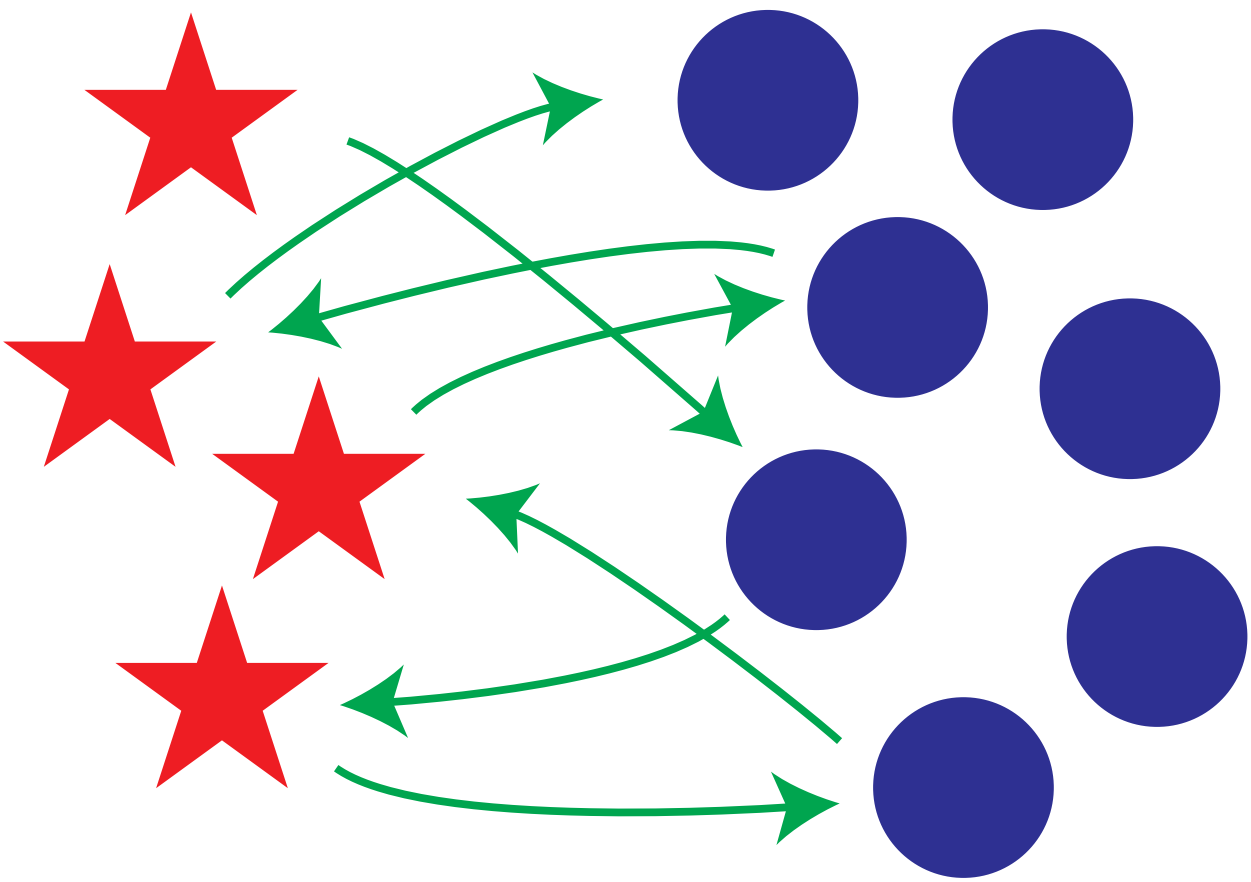
Can take advantage of geometric structure
to achieve sublinear update times bounds

> E.g. for roof design, $O(n^{6/11+\epsilon})$ per update with $O(n^{17/11+\epsilon})$ space, or $O(n^{3/4+\epsilon})$ per update with $O(n^{1+\epsilon})$ space. Since roof design performs $O(n)$ updates, can compute roof structure in $O(n^{17/11+\epsilon})$ time, or $O(n^{7/4+\epsilon})$ with nearly linear space.
>
> — *Joint work with Jeff Erickson*

Conga Lines

Choose any object to start the line

End of line chooses its favorite object among unchosen objects in other set



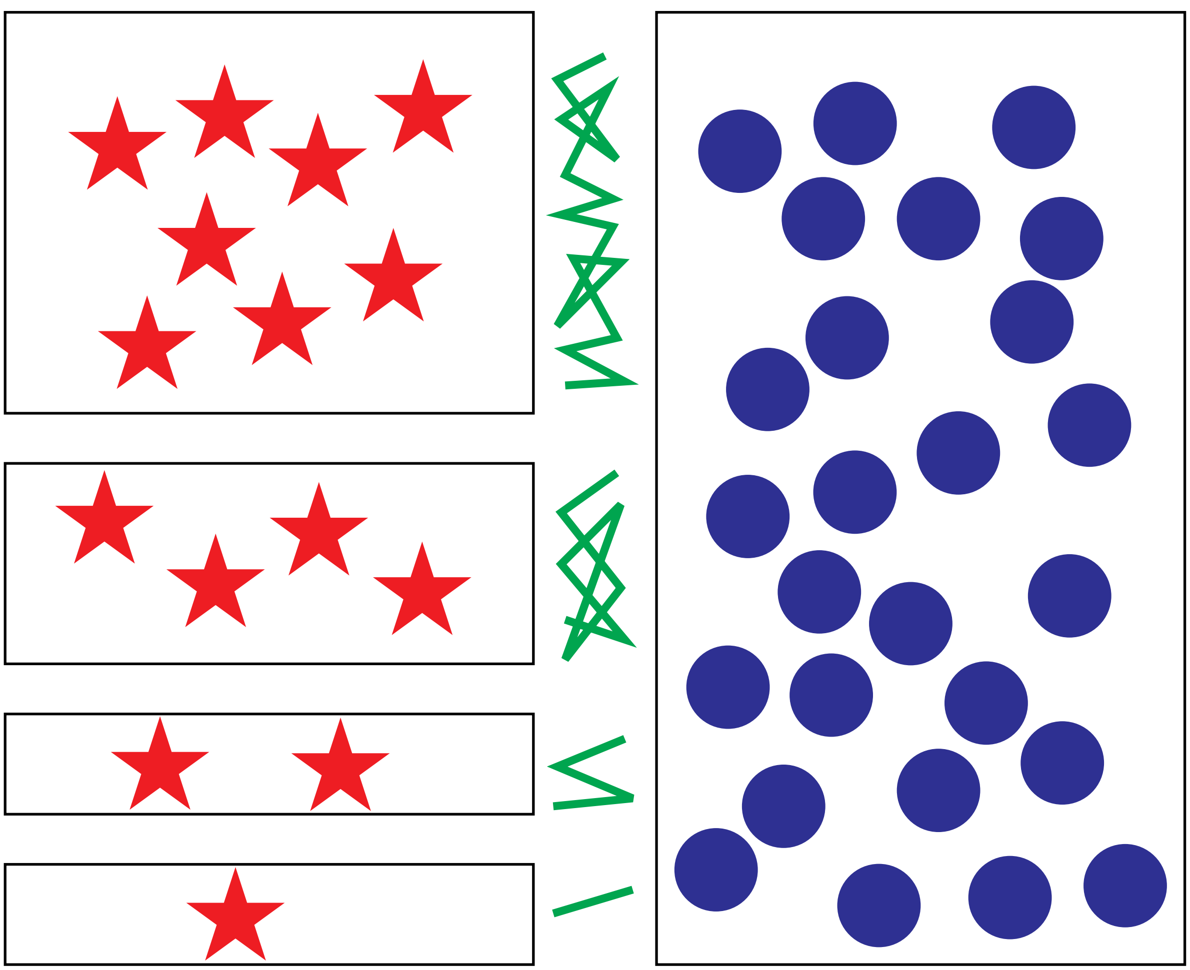
Lemma: if *f(s,t)* is minimized, then either *s* chooses *t* or *t* chooses *s*

Overall Data Structure

Divide S into powers of two

For each subset, form conga line with T

(similarly, divide T and form conga lines)



Data structure insertions

To insert an object:

Make new singleton subset

Regroup subsets into distinct powers of two

Recompute conga lines

Analysis:

Each time object is involved in a recomputation, subset size doubles

So at most $\log n$ recomputations

Total time per insertion: $O(n \log n)$.

Data structure deletions

To remove an object:

> Remove it from $O(\log n)$ conga lines (breaking each line in two)
>
> Treat neighbors at broken ends of lines as if they were newly inserted objects

Analysis:

> Each deletion causes $O(logn)$ insertions
>
> Total time per deletion: $O(n \log^2 n)$

Conclusions

Data structure for maintaining function minima

As fast as priority-queue approach

As space-efficient as brute force approach

Many applications