Asynchronous Parallel Pattern Search

PATTY HOUGH
Sandia National Laboratory

TAMMY KOLDA
Sandia National Laboratory

VIRGINIA TORCZON
College of William & Mary

Problem

$$\min_{x \in \mathbb{R}^n} f(x)$$

where $f : \mathbb{R}^n \to \mathbb{R}$.

Assume

- f is nonlinear,
- f is continuously differentiable,
- f is computationally expensive to evaluate,
- ∇f is unavailable, and
- approximations to ∇f are not reliable.

Pattern Search Methods Only Need Function Values — No Derivatives

- **No Derivatives:** Pattern search methods do not require derivative/gradient/sensitivity information.
- **Noisy Data:** Pattern search methods tend to do well even when the data is "noisy".
- **"Embarrassingly" Parallel:** Pattern search methods are easily amenable to parallelization, even when the cost function is serial.

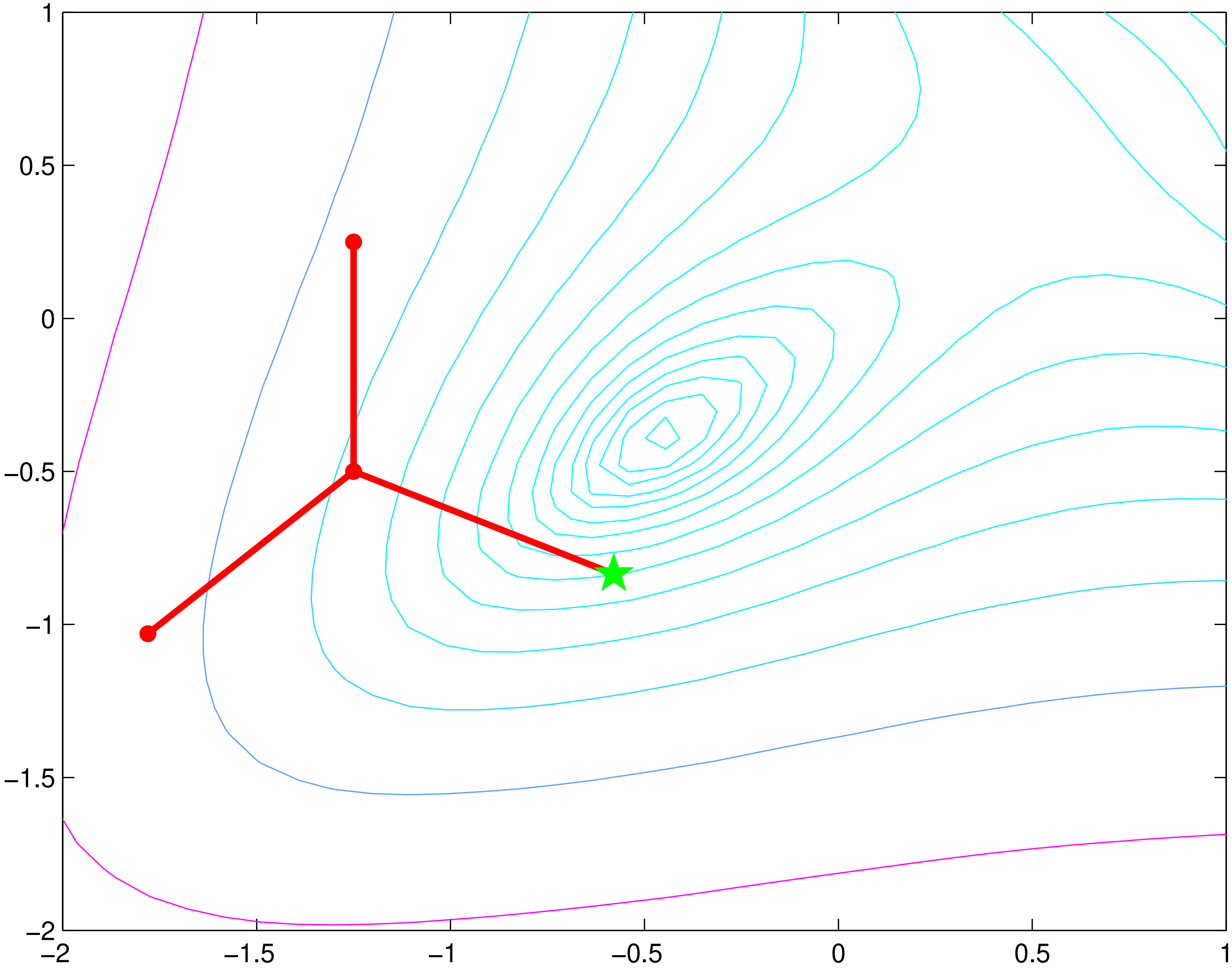
Why Revisit the Parallelization of Pattern Search?

- The problems of interest are defined by computationally expensive computer simulations of complex processes.
- The platforms for high-performance computing have become loosely-coupled networks of commercial off-the-shelf processors.
- The analysis for pattern search suggest a whole new range of algorithmic options to investigate.

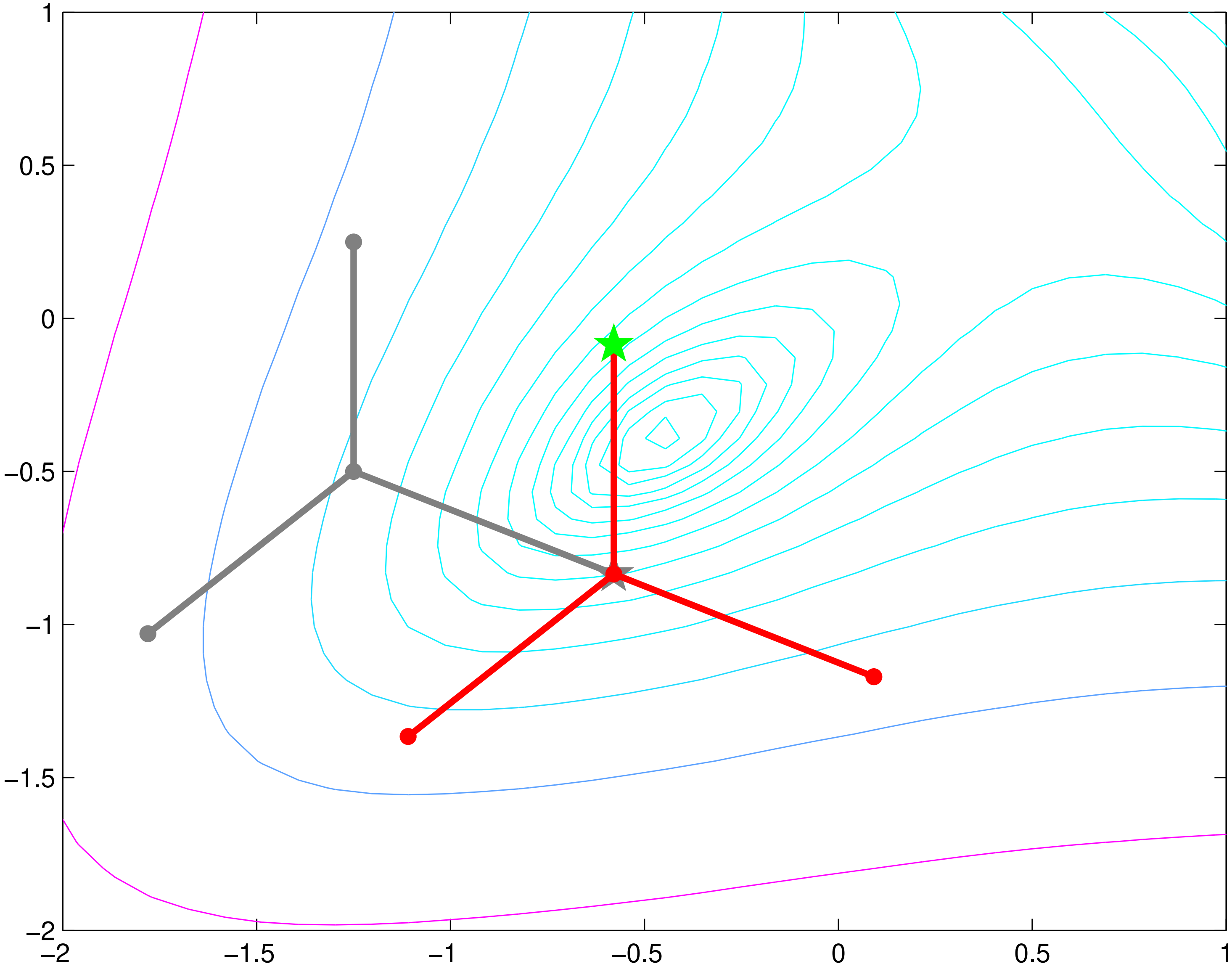
Our Goals

- Versatile concurrent strategies that allow us to balance the computational load across all available processors.
- Fault-tolerance that can be incorporated with almost no additional overhead.

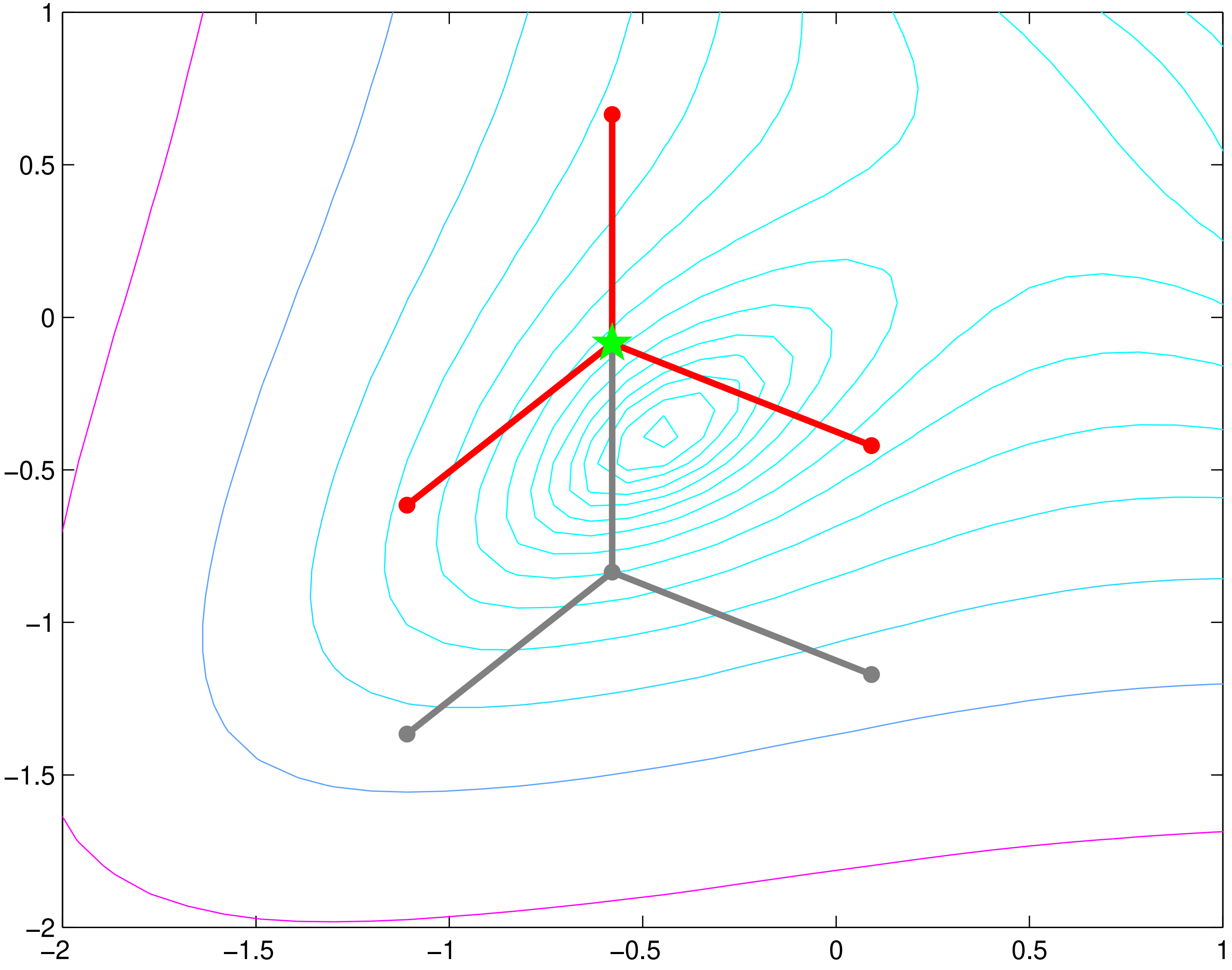
Parallel Pattern Search



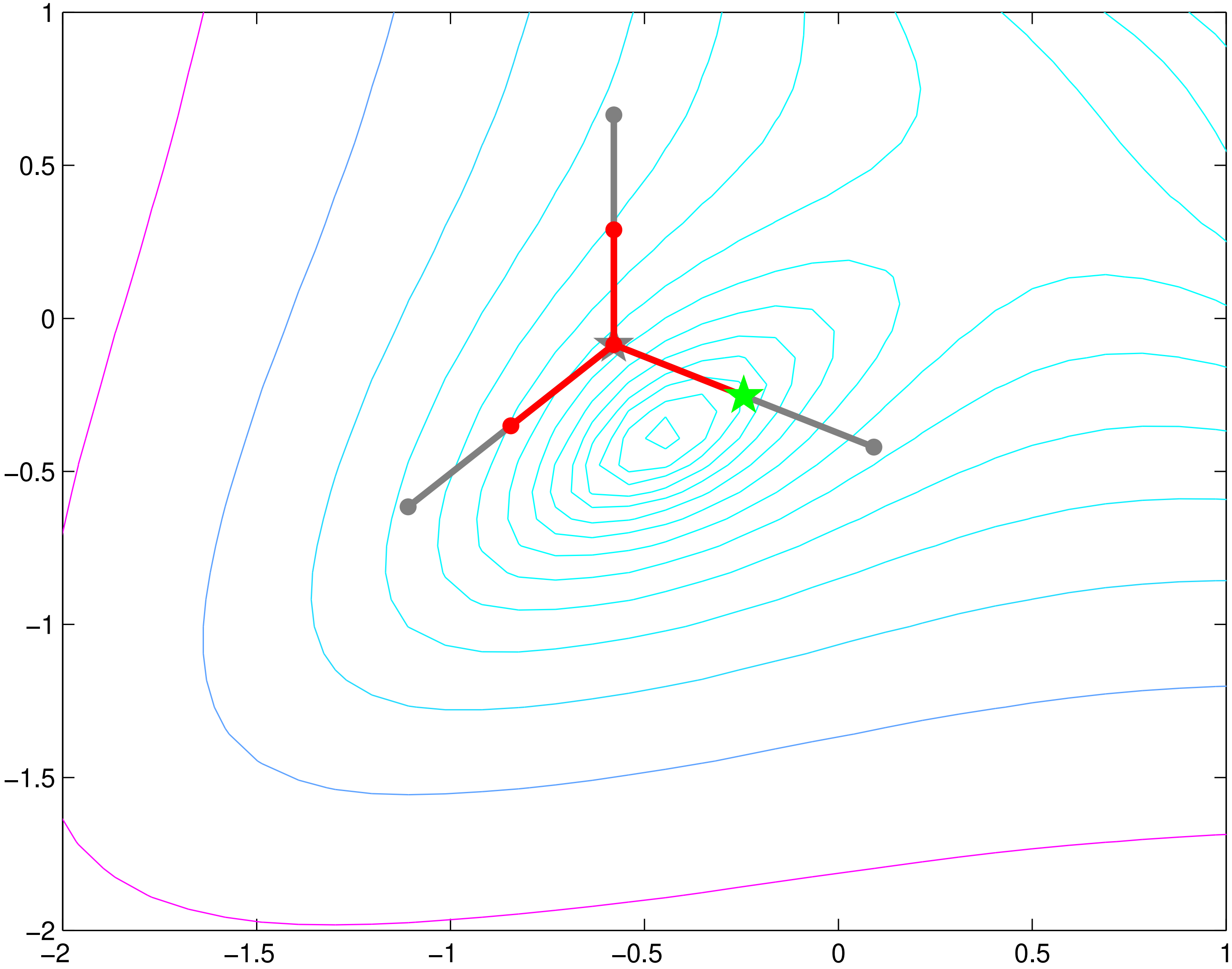
Parallel Pattern Search



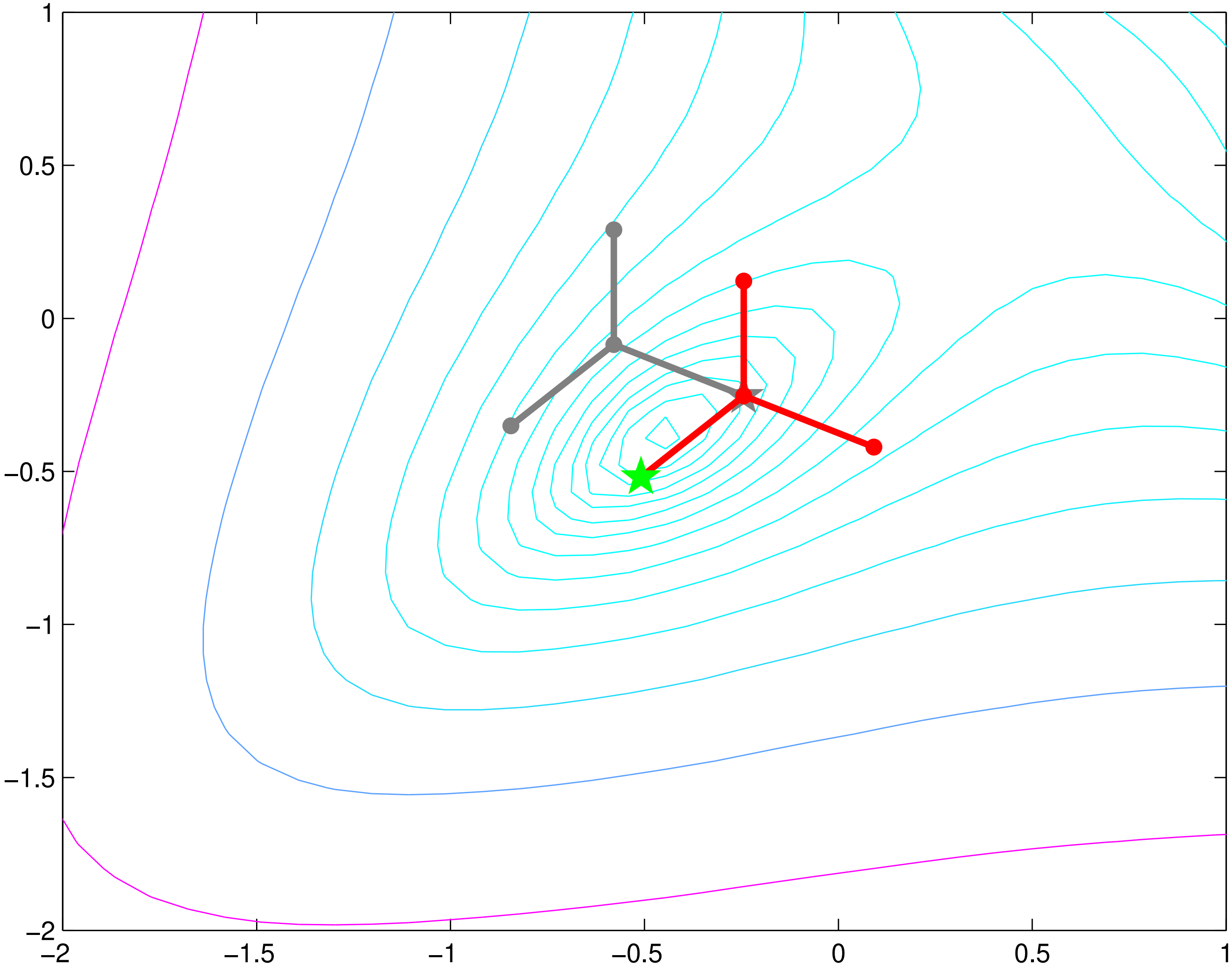
Parallel Pattern Search



Parallel Pattern Search



Parallel Pattern Search



Basic Ingredients for Parallel Pattern Search

Initialization:

- Choose a suitable set of search directions.
- Create one process responsible for each direction.

Search (from the perspective of a single process):

- evaluate f;
- synchronize (global reduction) to find the best trial point across all processes
- update local information based on results from the global reduction
- repeat until convergence.

PPS is Unsatisfactory from the Perspective of Distributed Computing

- PPS may waste computational resources
- PPS is not fault-tolerant

It is the synchronization step (the global reduction) contained in every iteration of PPS that introduces these bottlenecks.

Why PPS Might Waste Computational Resources

Ignores:

- variations in the time required to compute f on homogeneous processors,
- possible heterogeneity of processors, and
- differences in the number of processes per host processor.

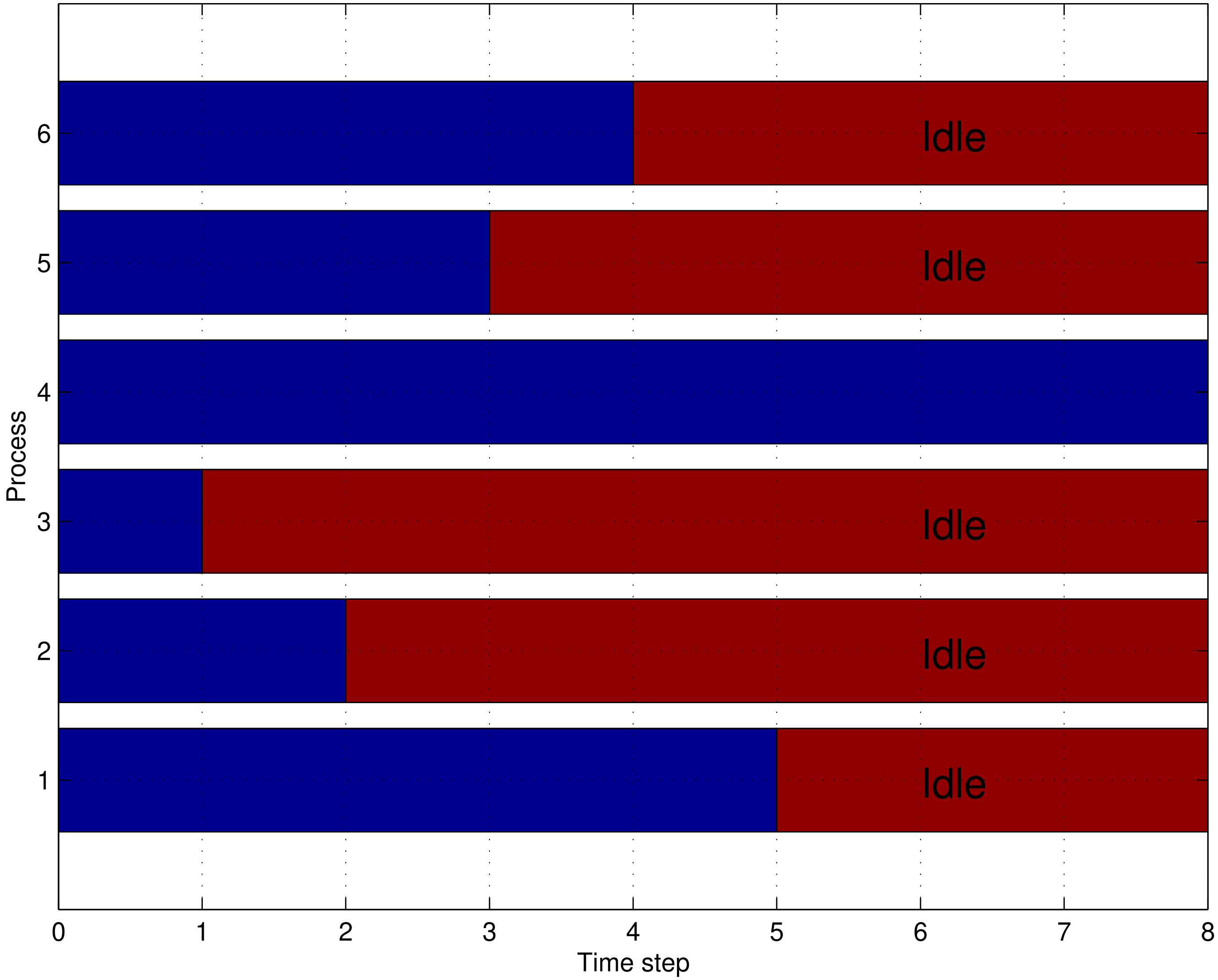
$\Rightarrow$ idle time at every iteration

Further:

If a process fails, then, effectively, the number of time steps required to receive a message from that process becomes infinite.

$\Rightarrow$ idle time on the active processes becomes infinite

An Illustration of the Consequences



Our Idea

Since these variations are usually too difficult — if not impossible — to predict, use flexibility of pattern search to devise:

- adaptive,
- event-driven,
- fault-tolerant,
- peer-to-peer

pattern search algorithms.

$\Rightarrow$ Asynchronous Parallel Pattern Search

This is in Contrast to the Standard Approach to Load Balancing

- create a pool of tasks
  (master/slaves or manager/workers paradigm)
- use generic job scheduling strategies

Also in Constrast to the Standard Approach to Fault Tolerance

- checkpoint computation/state at routine intervals
  (involves repeated writing to disk)
- restart failed computation from last checkpoint

Basic Strategy for Asynchronous Parallel Pattern Search

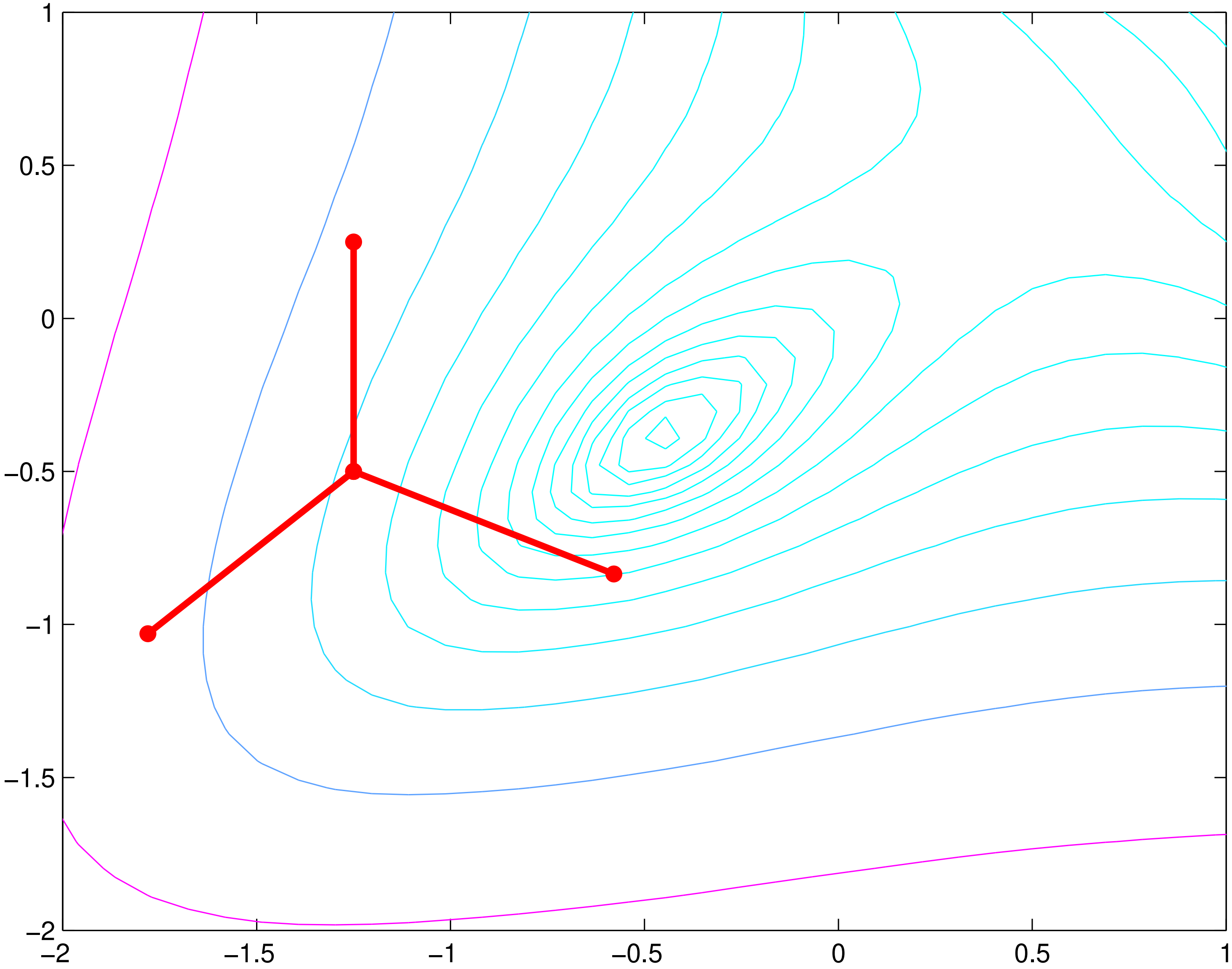
Initialization:

- Choose a positive spanning set $\mathcal{D} = \{d_1, \ldots, d_p\}$ such that $\forall\ d_i \in \mathcal{D},\ d_i \in \mathbb{Q}^n$.
- Create one process responsible for each search direction $d_i \in \mathcal{D}$.

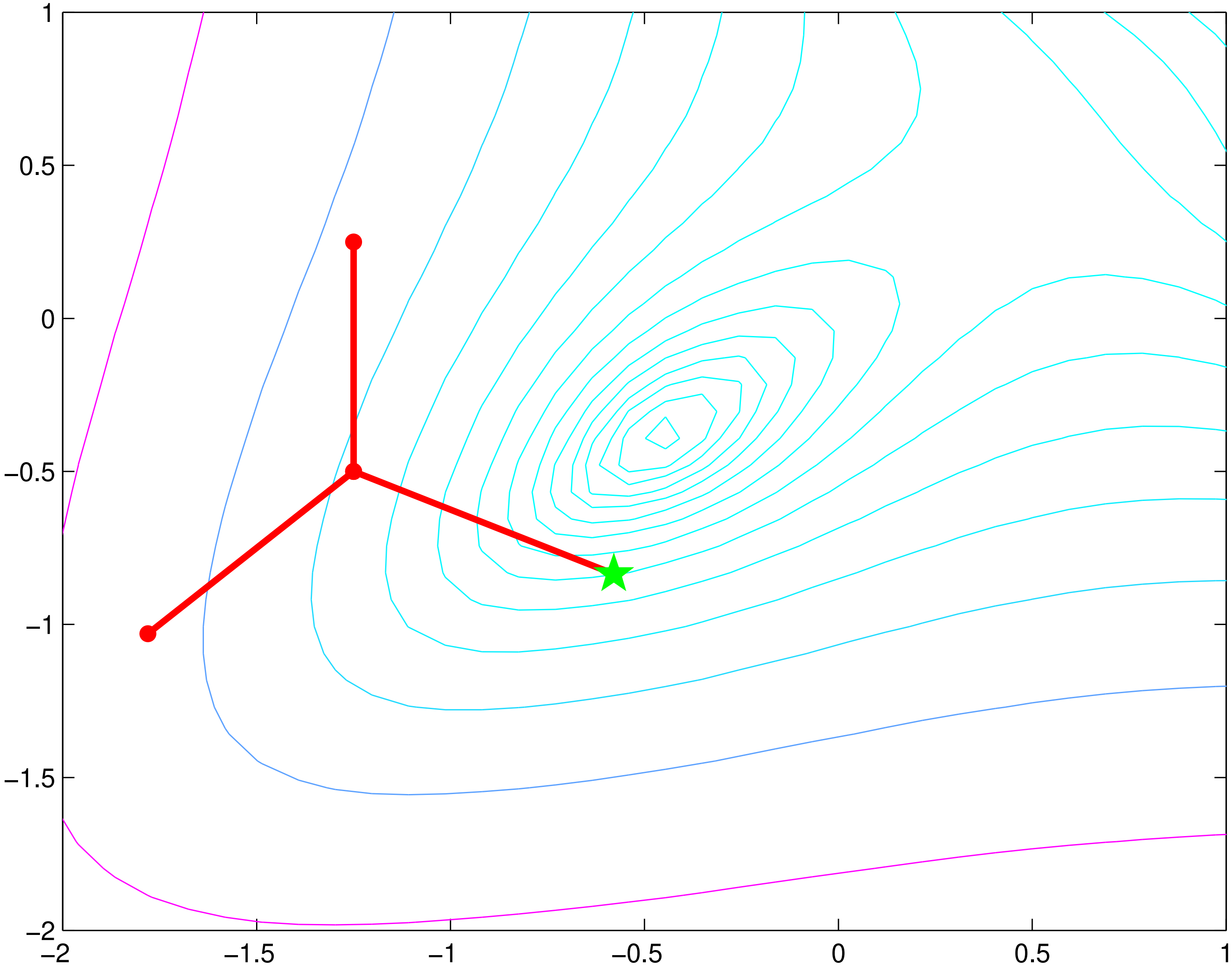
Search:

- evaluate $f(x_{best} + \Delta_{best}\, d_i)$;
- if $f(x_{best} + \Delta_{best}\, d_i) < f(x_{best})$, then broadcast (non-blocking) result to other processes;
- update local copies of x_{best} and Δ_{best}, based on current local information; and
- repeat until convergence.

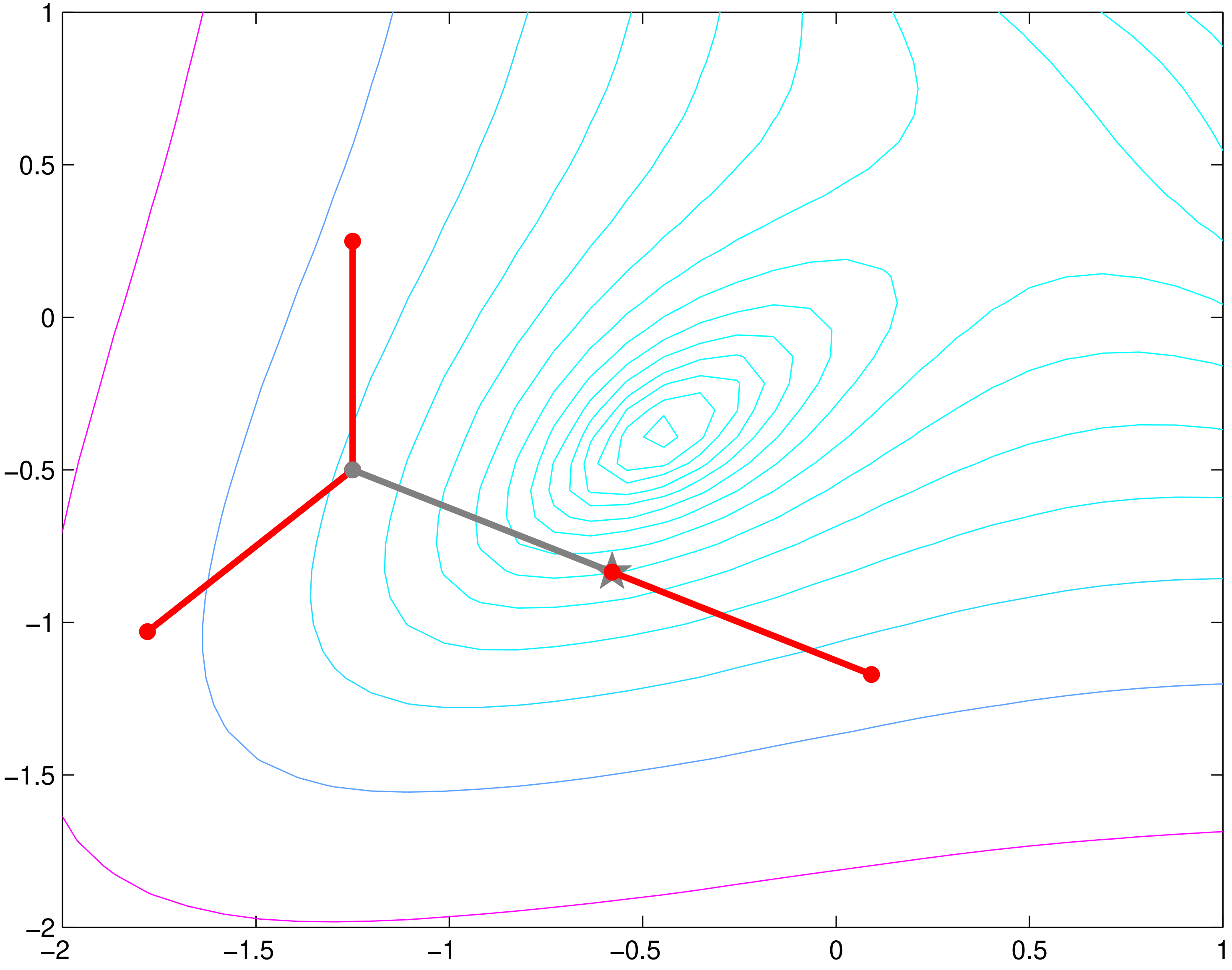
Asynchronous Parallel Pattern Search



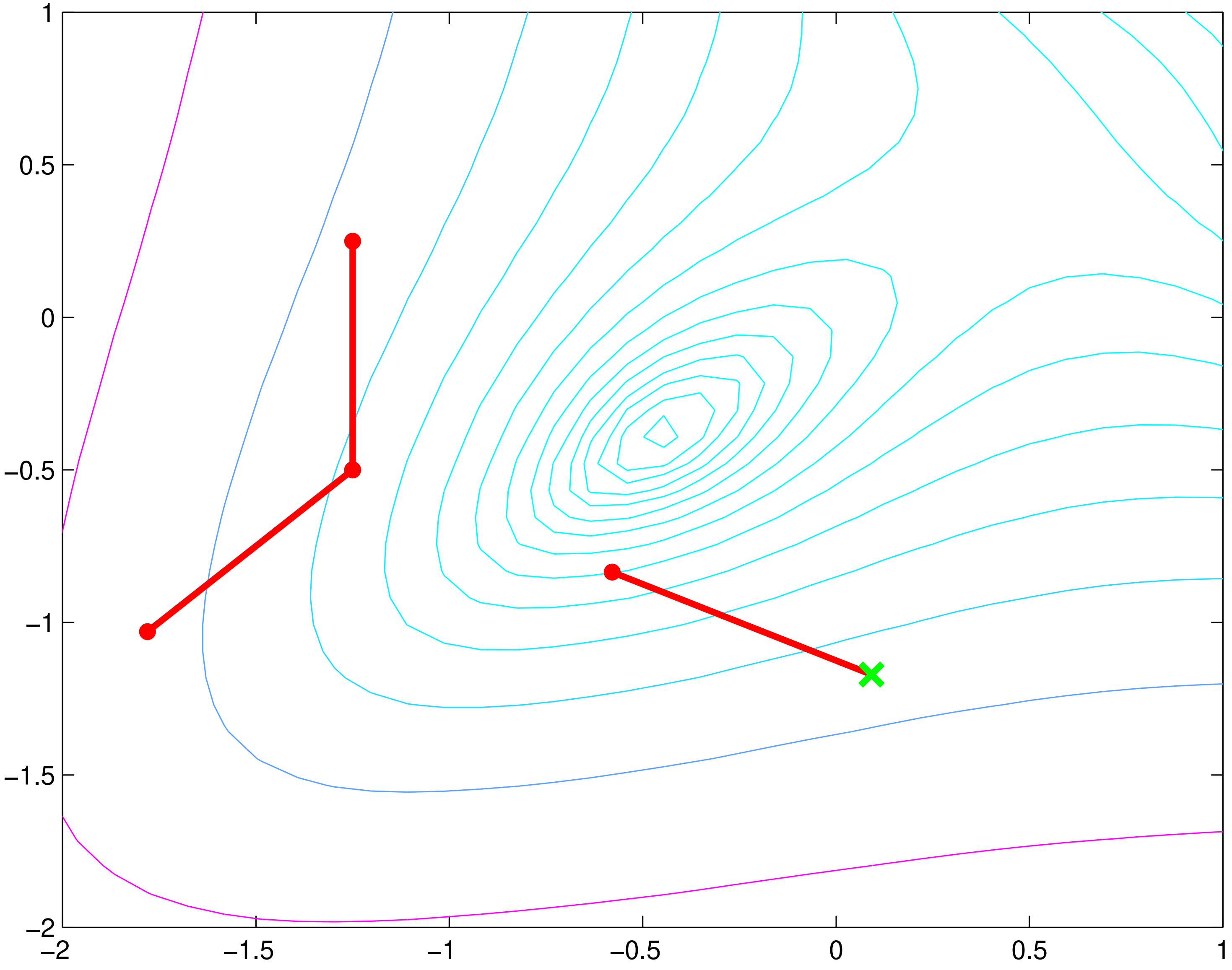
Asynchronous Parallel Pattern Search



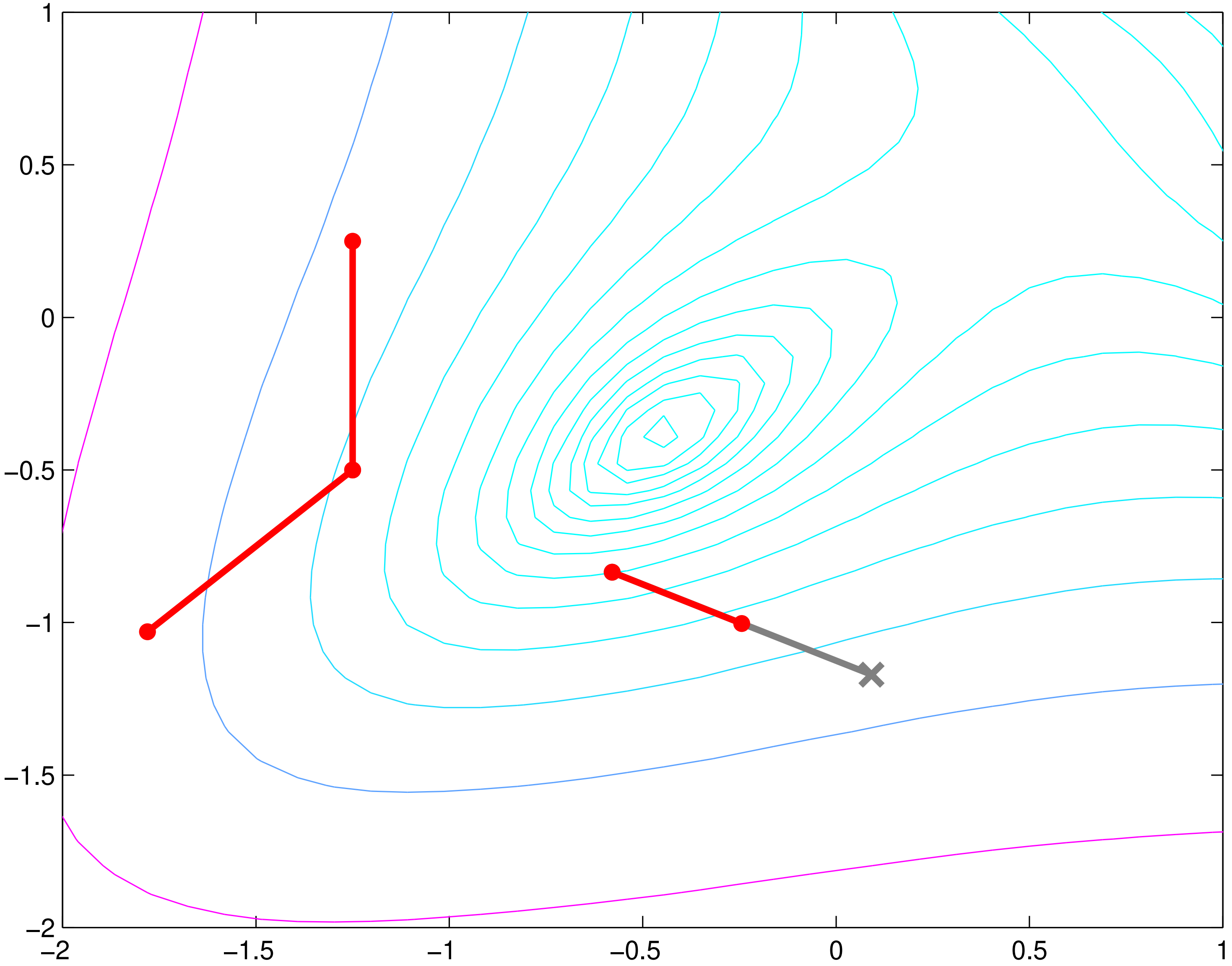
Asynchronous Parallel Pattern Search



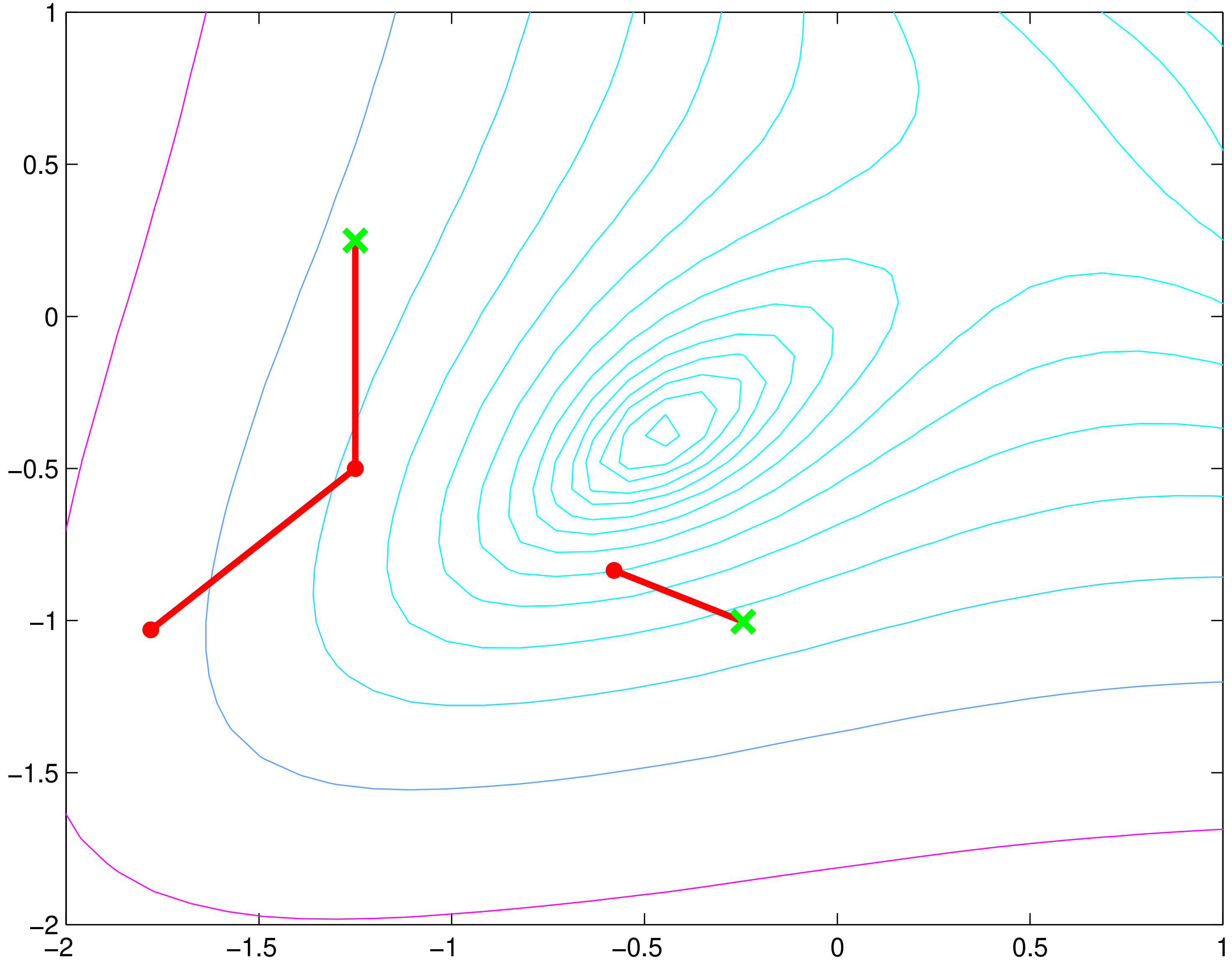
Asynchronous Parallel Pattern Search



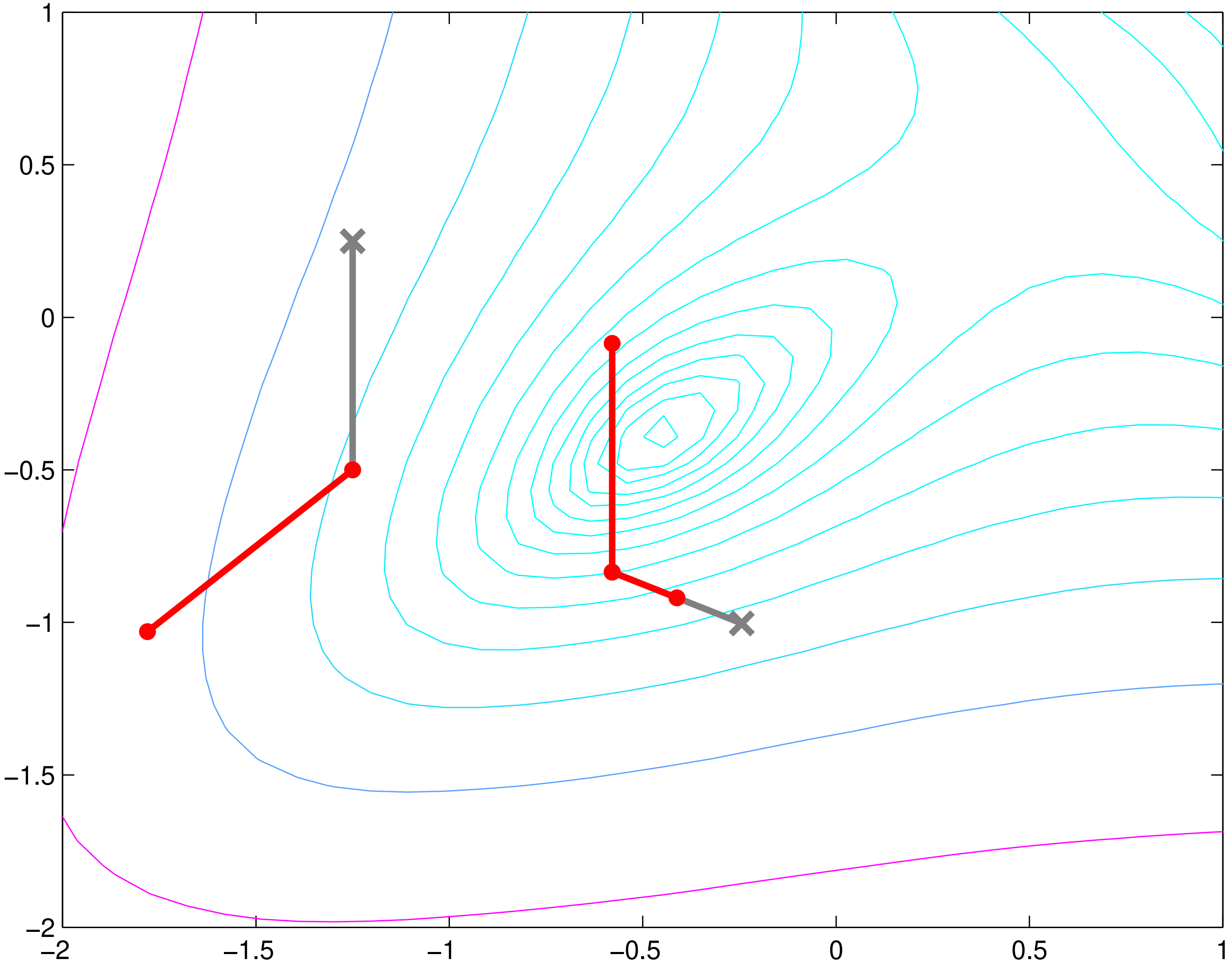
Asynchronous Parallel Pattern Search



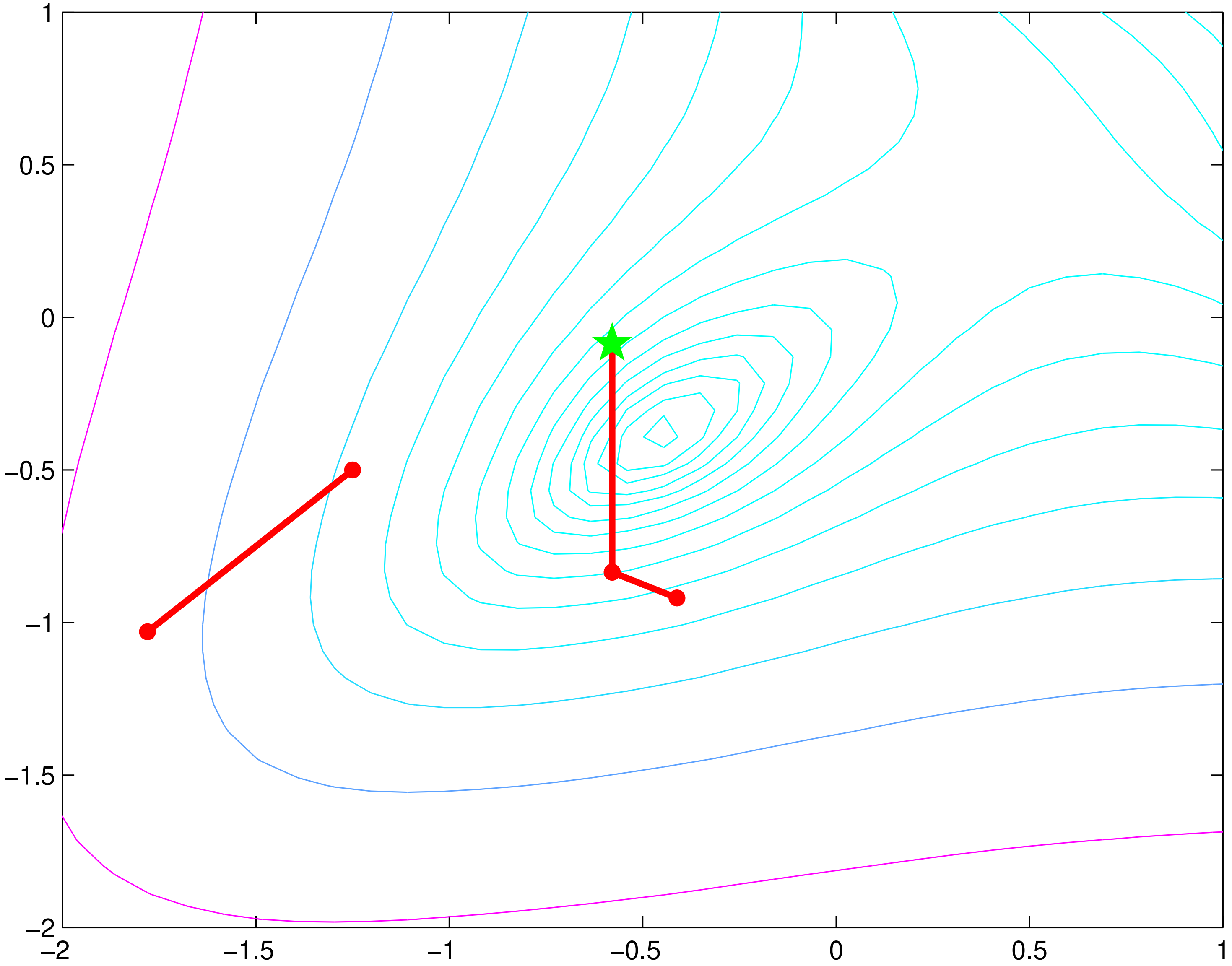
Asynchronous Parallel Pattern Search



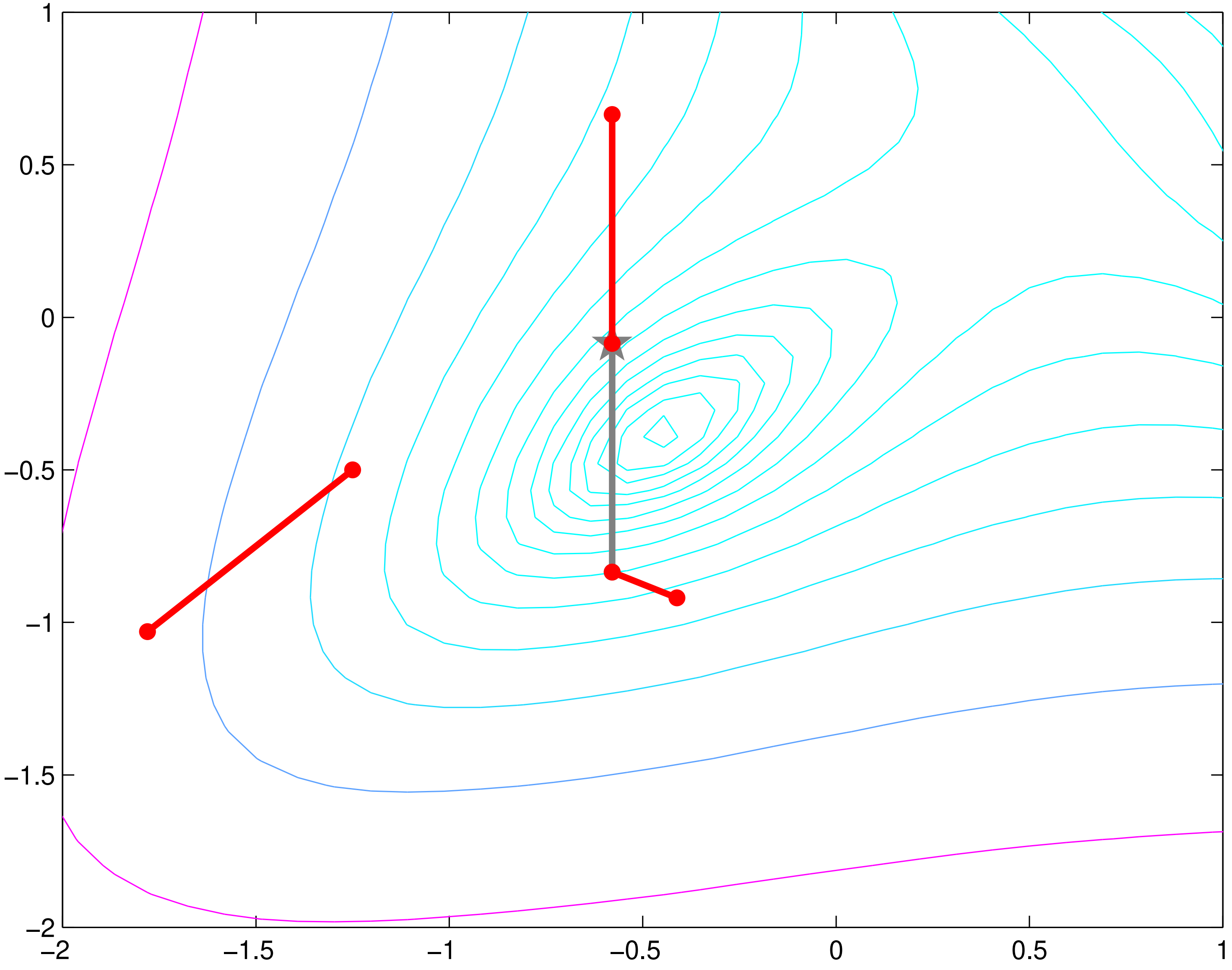
Asynchronous Parallel Pattern Search



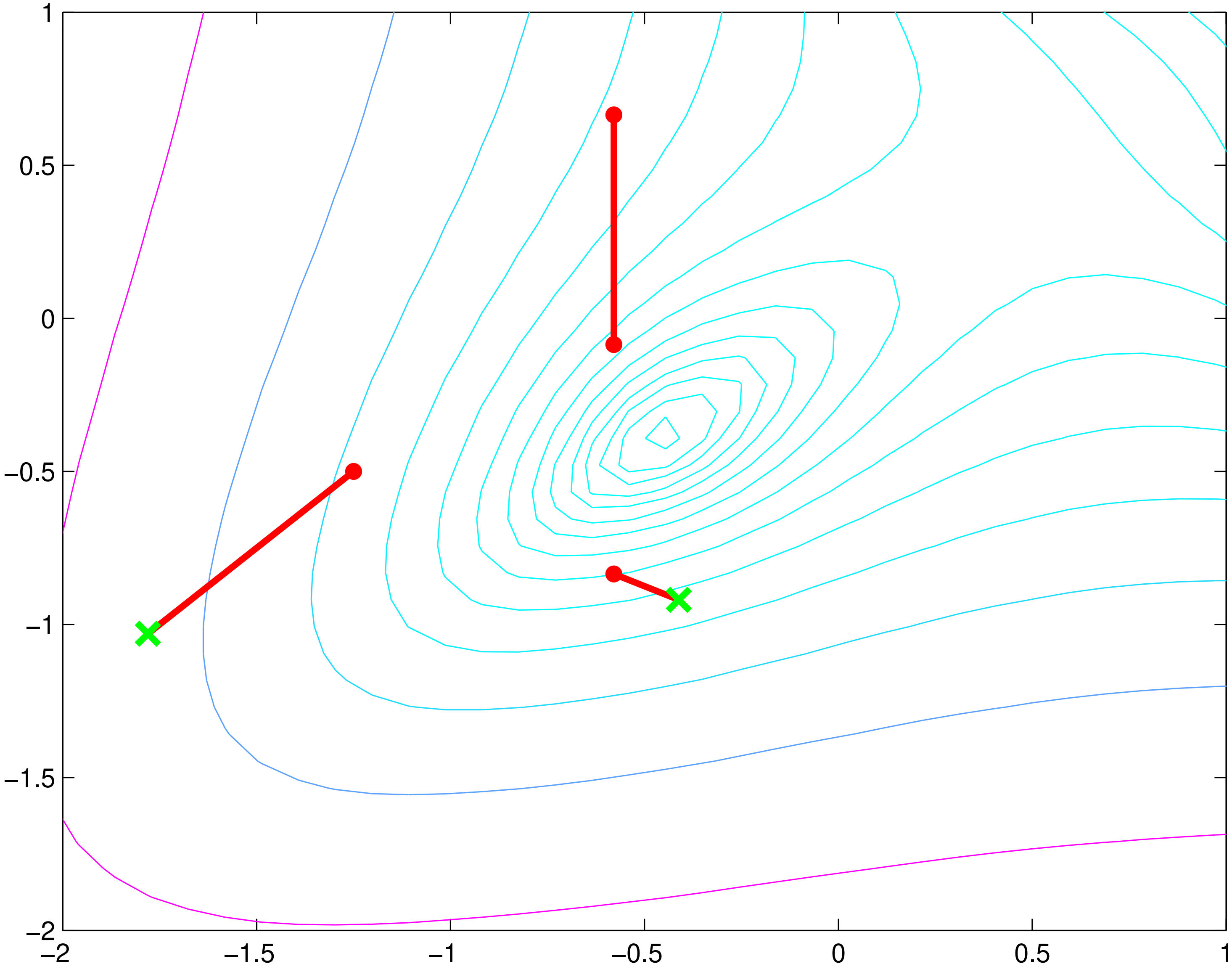
Asynchronous Parallel Pattern Search



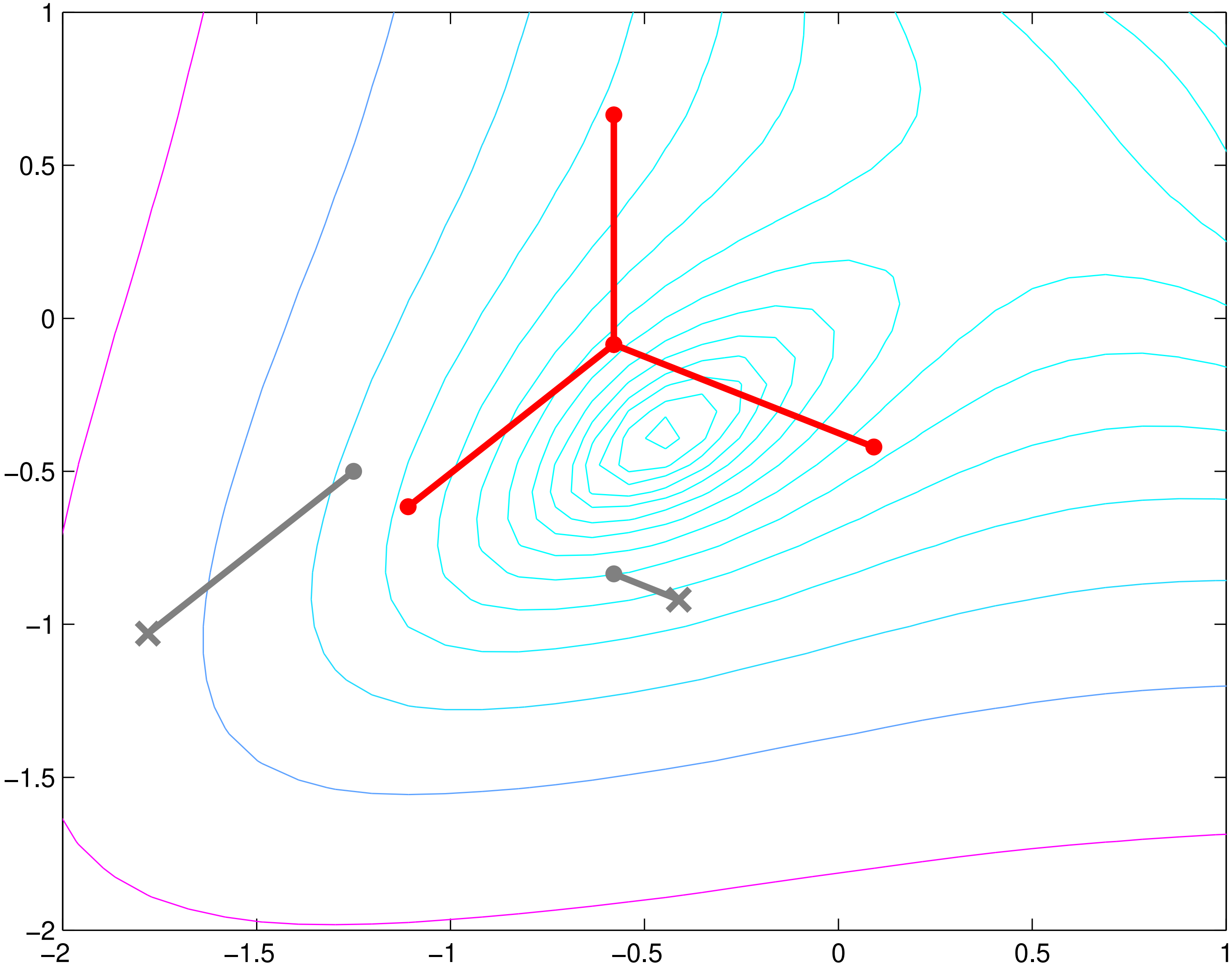
Asynchronous Parallel Pattern Search



Asynchronous Parallel Pattern Search



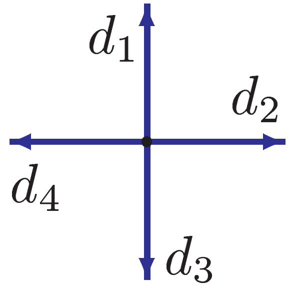
Asynchronous Parallel Pattern Search

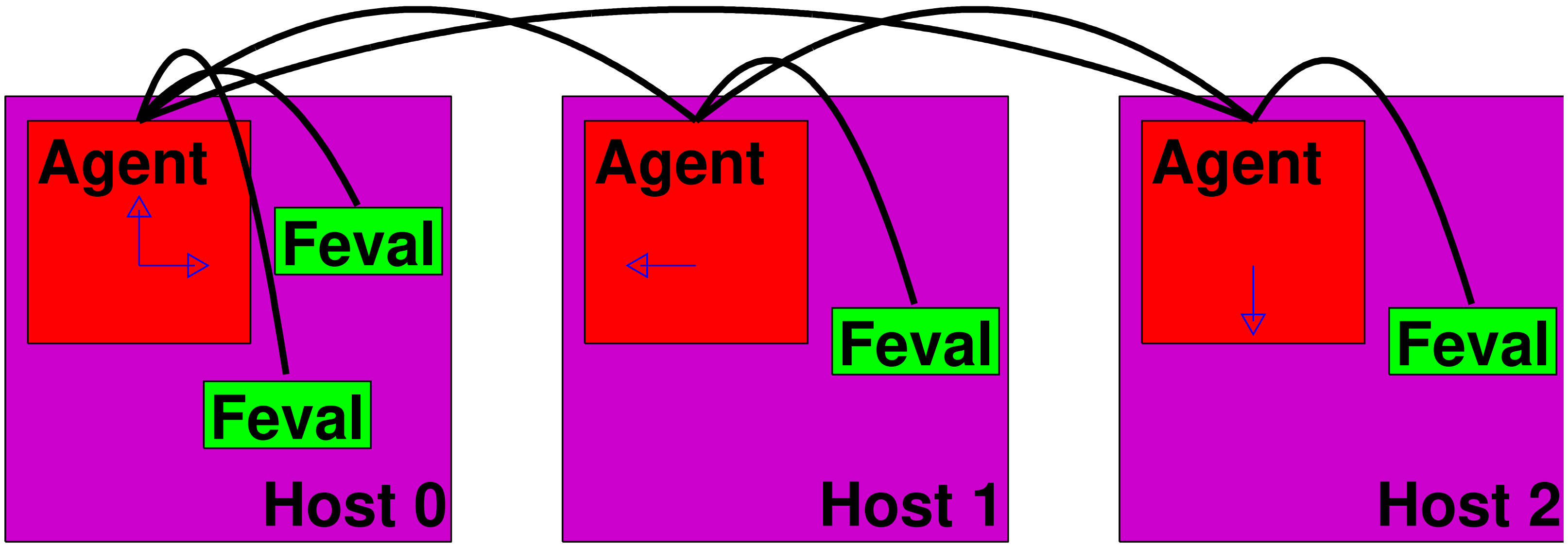


APPS Distributes Control via Agents

- The work of the pattern search is divided among *agents*, one per host.
- Each agent is in charge of some subset of the *search pattern*.
- Agents make algorithmic decisions based only on available information — *there is no synchronization*.

2-D Search Pattern





APPS Is Event-Driven

Rather than adhering to the notion of a fixed “iteration”, APPS agents respond to the type of message they have received:

- return from evaluation of the objective function
  (received from the agent’s function evaluation processes),
- new best point
  (received from another agent),
- single direction convergence
  (received from another agent), and
- shutdown
  (received from the agent acting as *temporary master*).

APPS Is Fault-Tolerant

Agents respond to process/processor failures detected within the system:

- failure of function evaluation process,
- failure of another APPS agent, and
- failure of a host processor.

Agents respond in a way that ensures the overall integrity of the search (i.e., global convergence to a stationary point of f [Kolda/Torczon, 2001]).

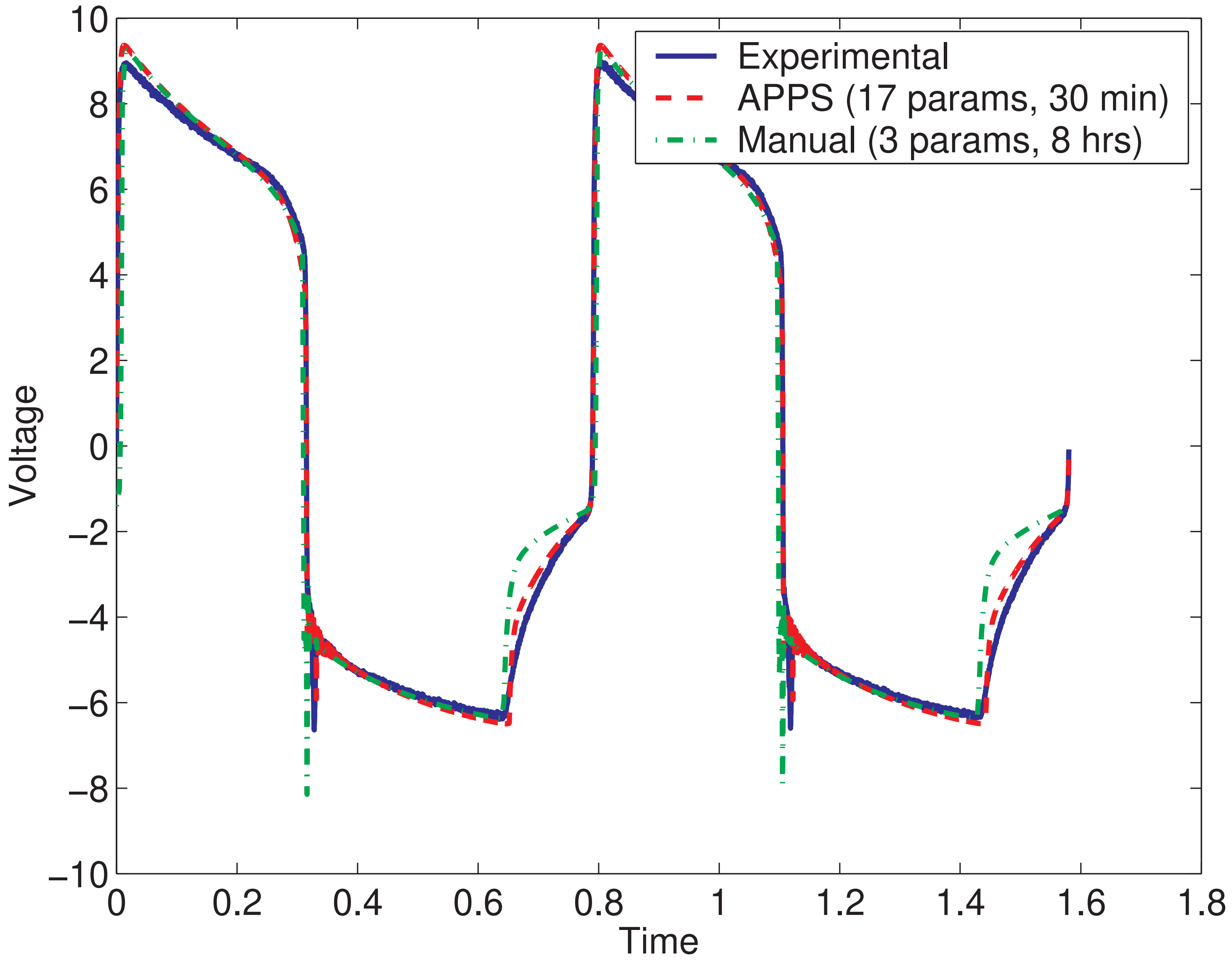
Engineering Example: Determining the Characteristics of a Circuit

- **Work by:** Tammy Kolda and Ken Marx (Sandia)
- **Variables:** inductances, capacitances, diode saturation currents, transistor gains, leakage inductances, and transformer core parameters
- **Simulation Code:** SPICE3

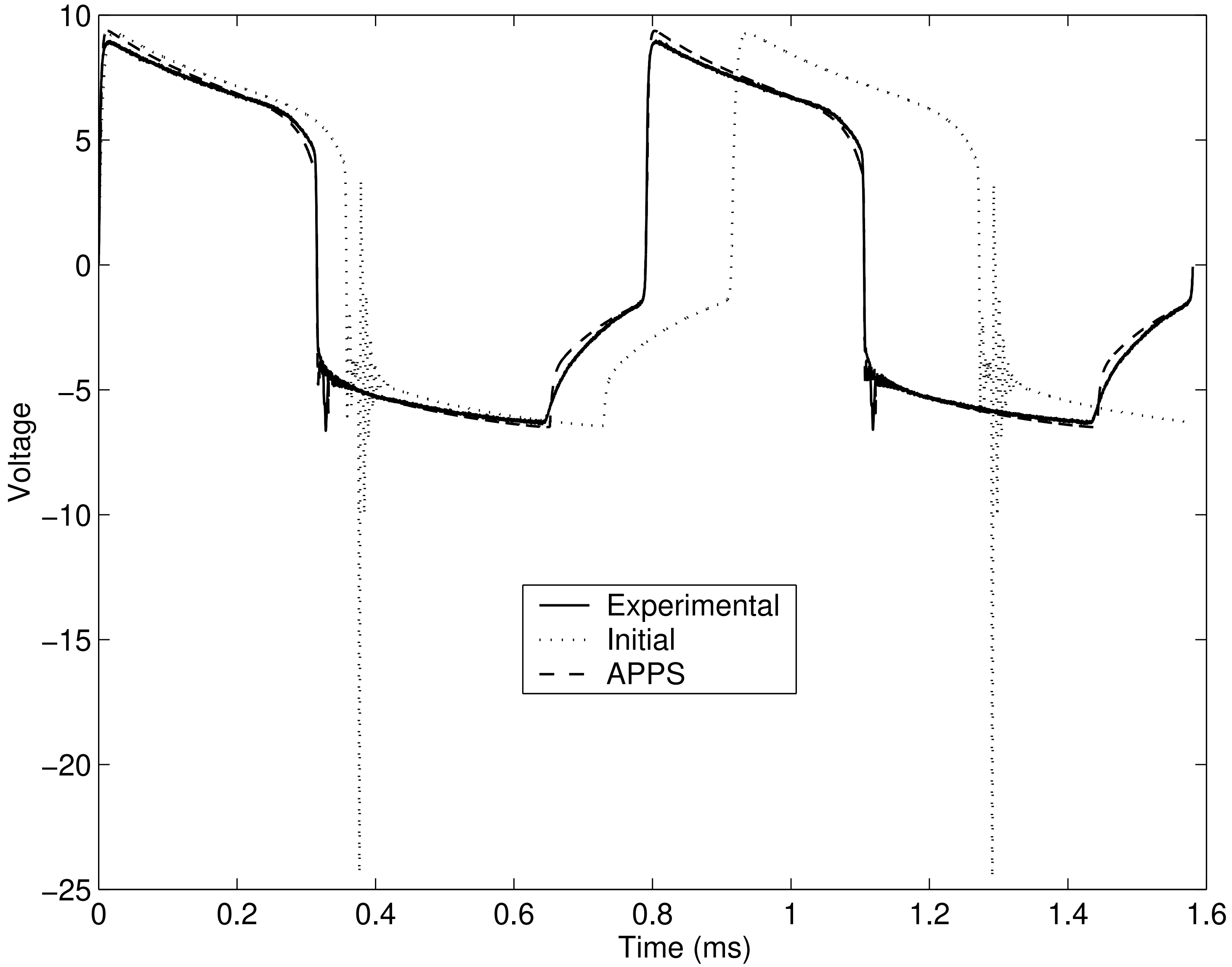
$$f(x) = \sum_{t=1}^{N} \left(V_t^{\mathrm{SIM}}(x) - V_t^{\mathrm{EXP}} \right)^2 ,$$

$$\begin{aligned} x &= \text{17 unknown characteristics} \\ V_t^{\mathrm{SIM}}(x) &= \text{Simulation voltage at time } t \\ V_t^{\mathrm{EXP}} &= \text{Experimental voltage at time } t \\ N &= \text{Number of time steps (2700)} \end{aligned}$$

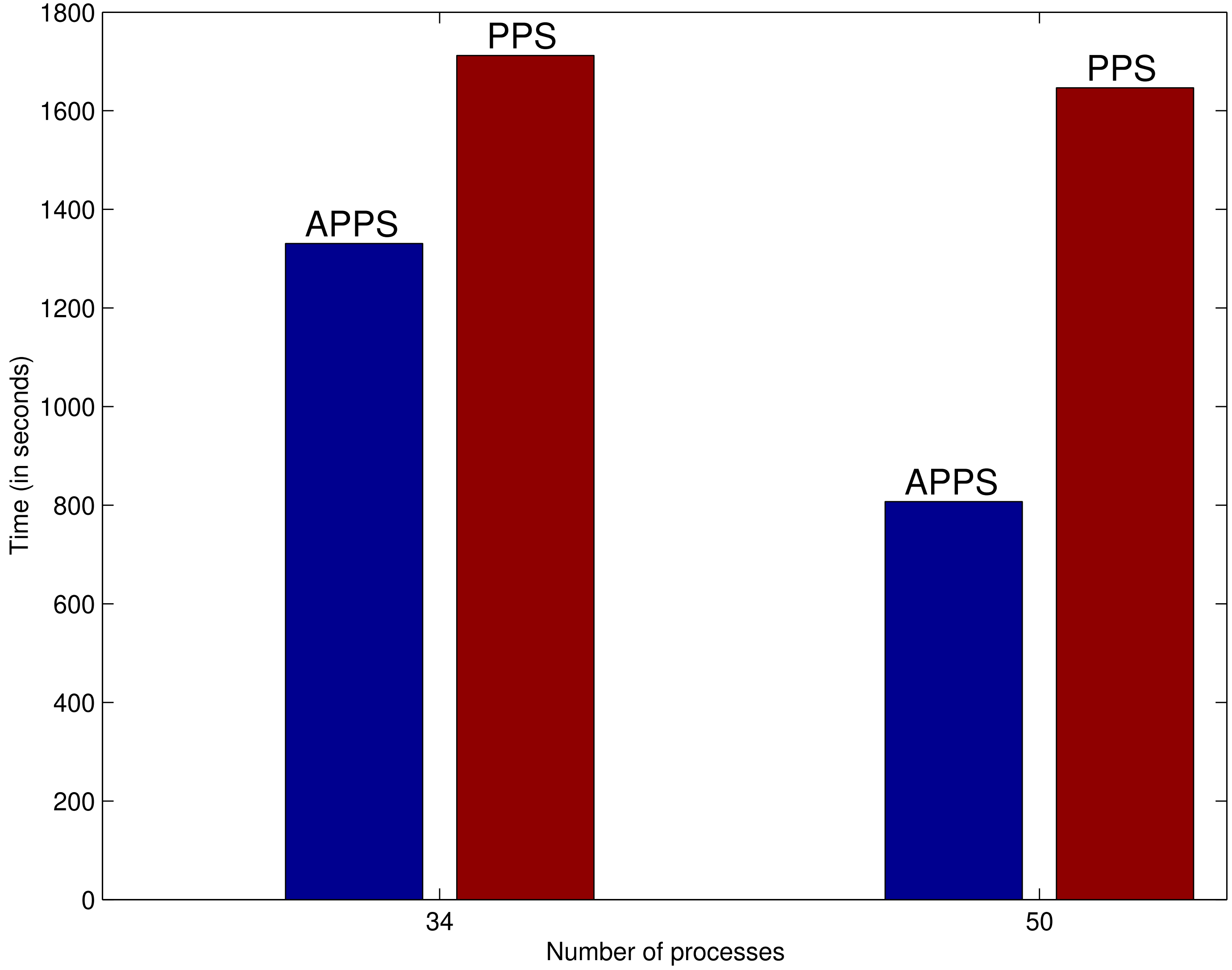
APPS Solution is Faster and More Accurate than Hand-Tuning



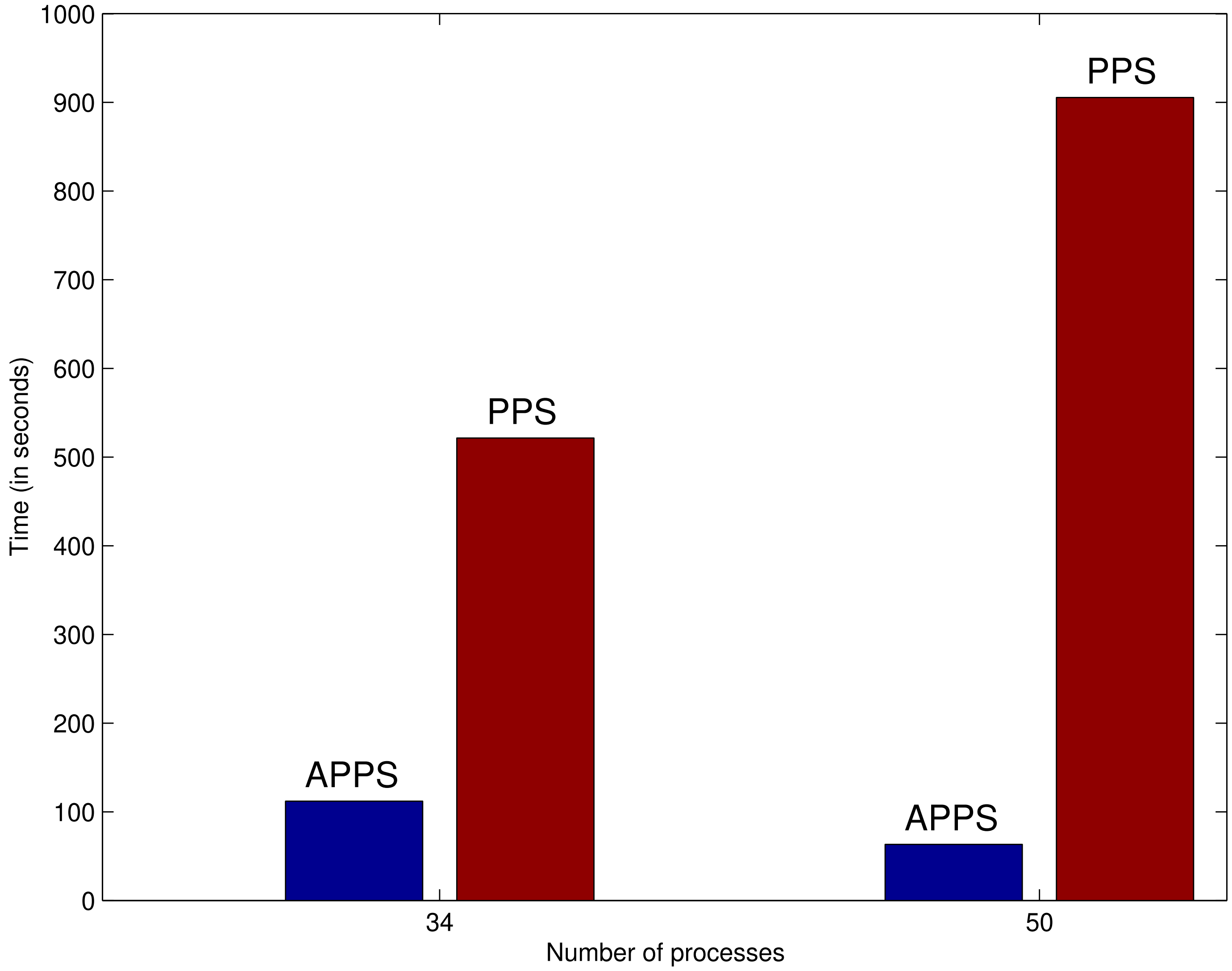
APPS Solution Shows Significant Improvement over Initial Parameters



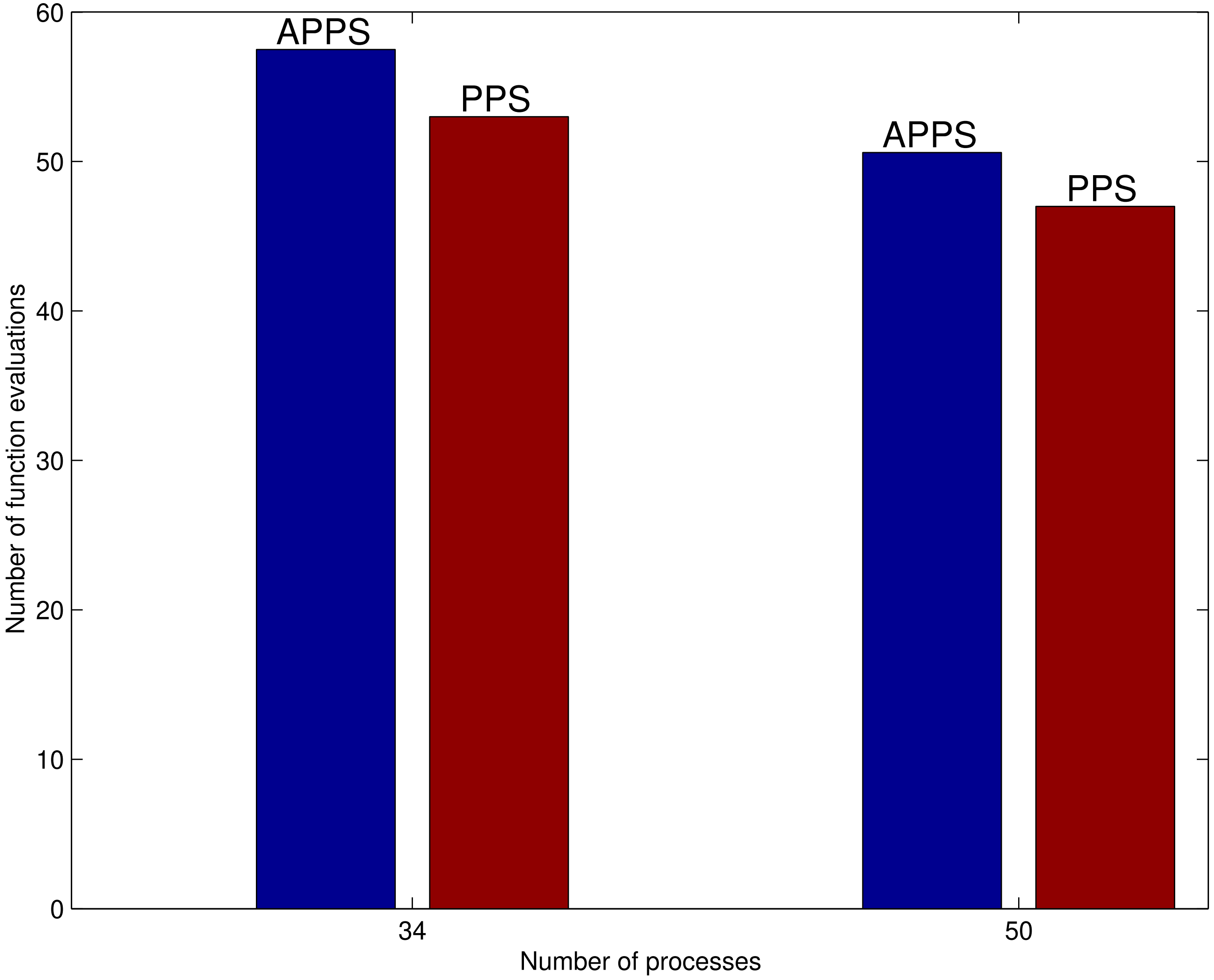
Total Execution Time for APPS versus PPS on the SPICE Problem



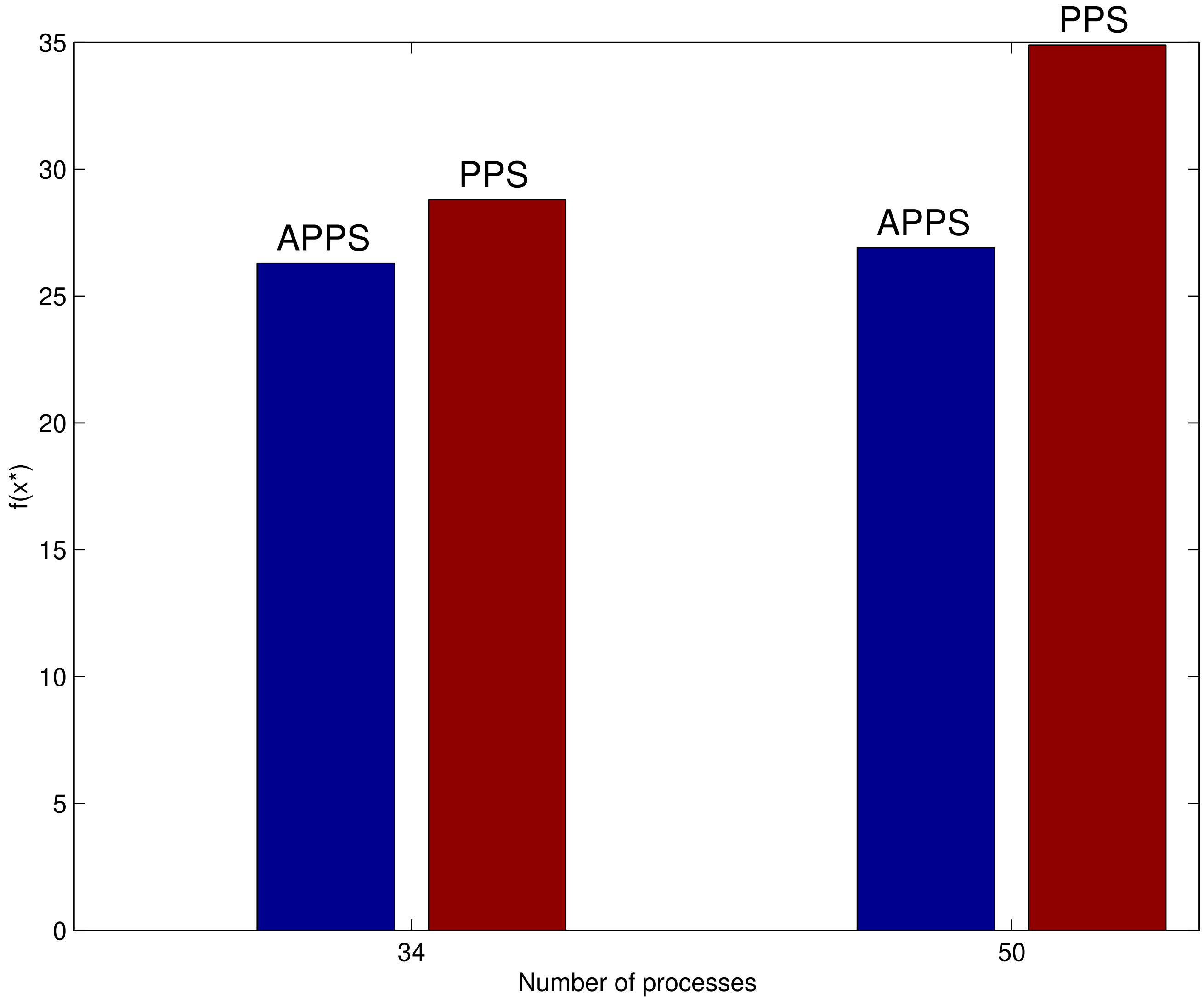
Average Idle Time per Process for APPS versus PPS on the SPICE Problem



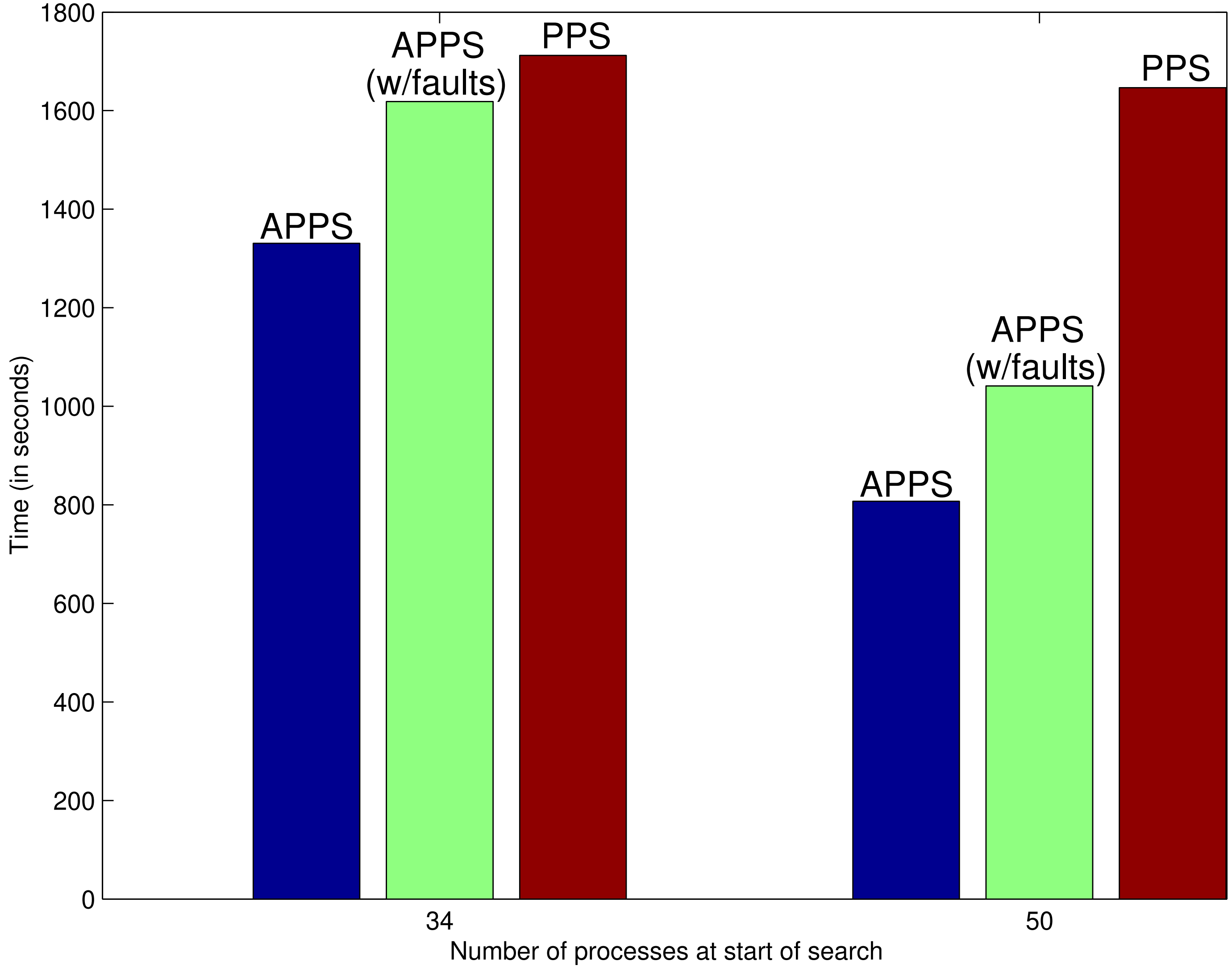
Average Number of Function Evaluations per Process for APPS versus PPS on the SPICE Problem



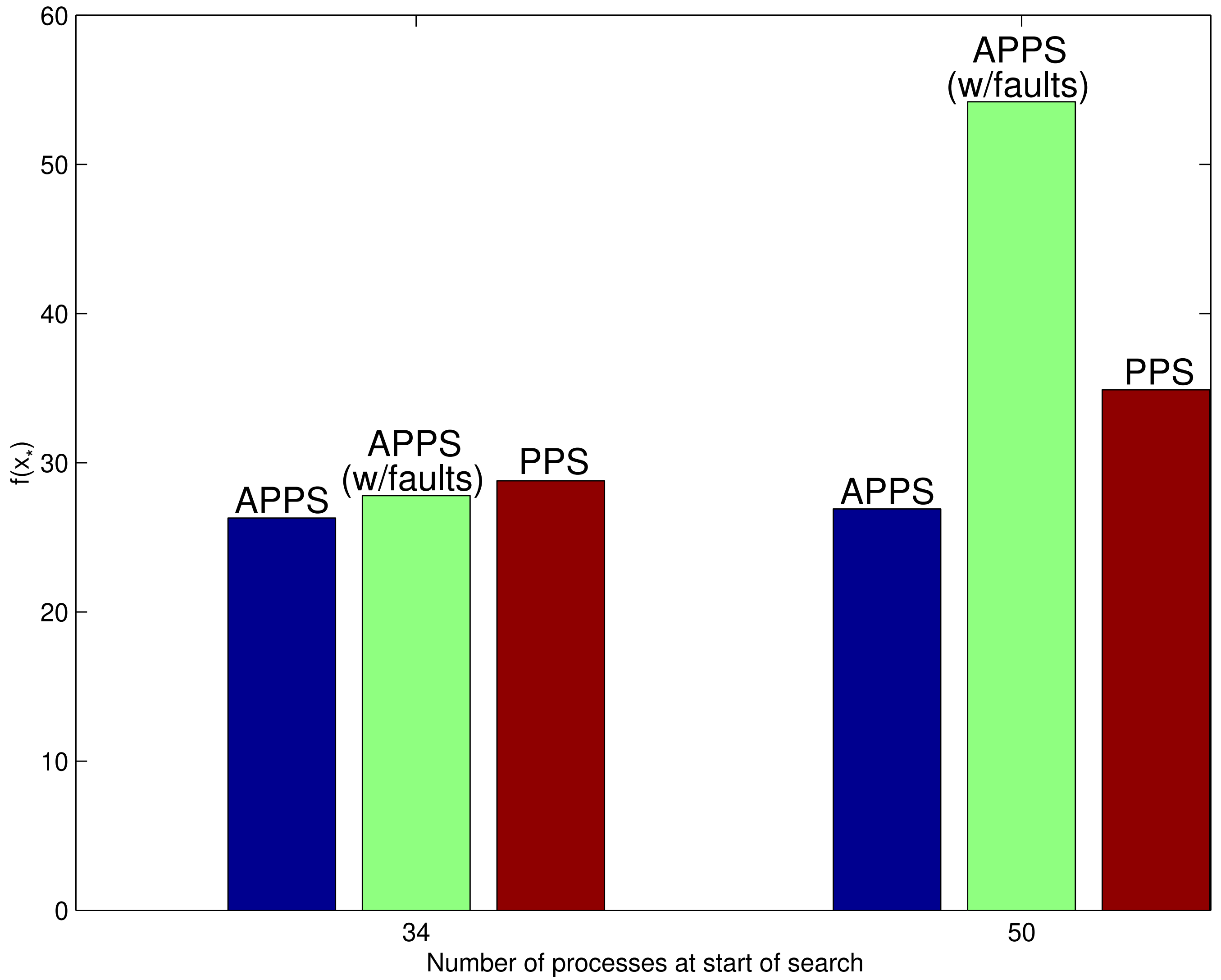
Best Value of f Found by APPS versus PPS for the SPICE Problem



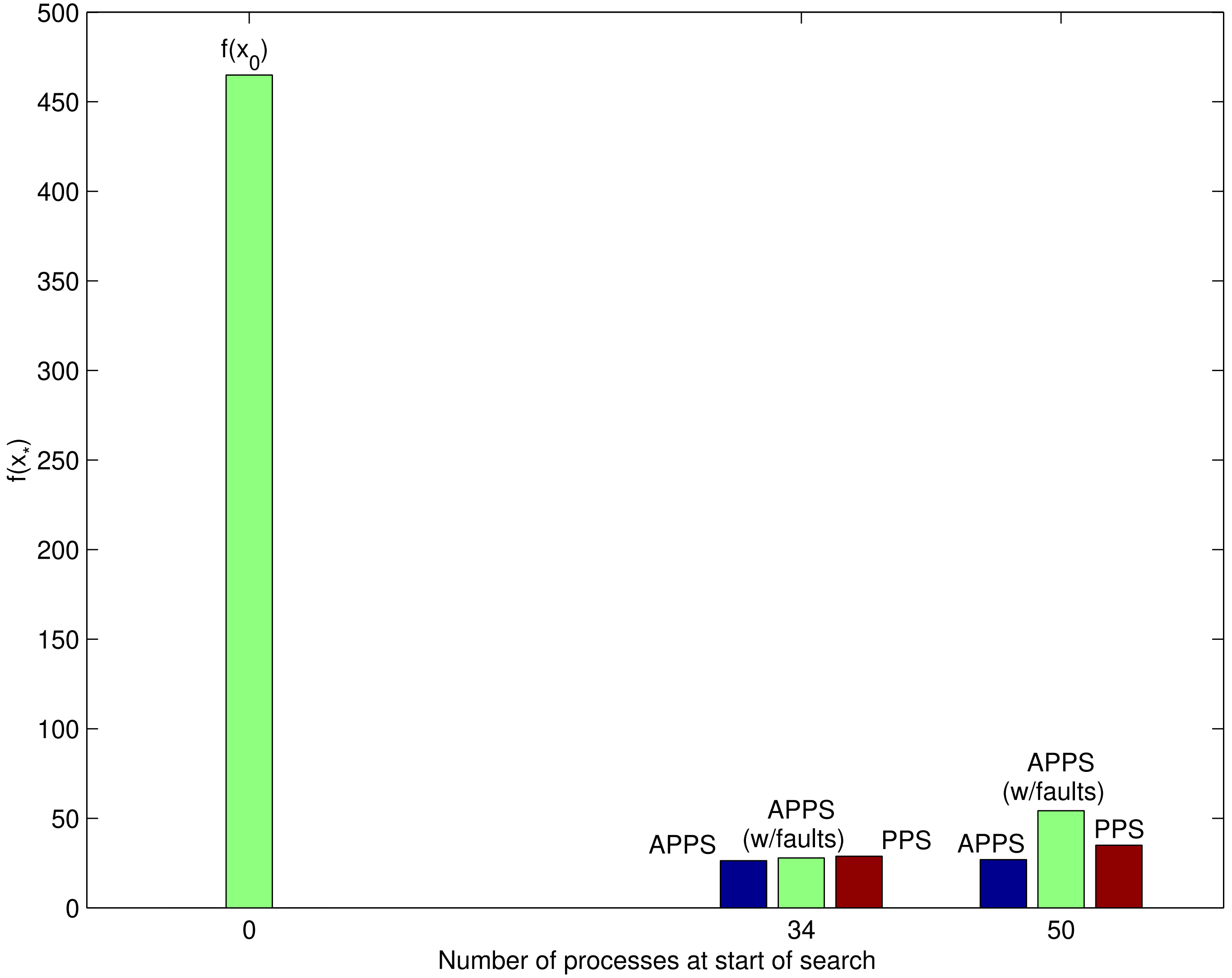
Total Execution Time for PPS, APPS, and APPS with A Failure Approximately Every 30 Seconds



Best Value of f Found by PPS, APPS, and APPS with a Failure Approximately Every 30 Seconds



To Put the Final Values of f for the SPICE Problem in Perspective



To Get Started With APPSPACK, Contact

Tammy Kolda

tgkolda@sandia.gov

APPS/APPSPACK Collaborators

- Patty Hough (Sandia)
- Virginia Torczon (William & Mary)
- Alton Patrick (2000 Sandia Summer Intern)
- Sarah Brown (2000 Sandia Summer Intern)
- Danny Dunlavy (2001 Sandia Summer Intern)

For More Information on APPSPACK

http://csmr.ca.sandia.gov/projects/apps.html

http://csmr.ca.sandia.gov/~tgkolda/

APPSPACK is freely available under the terms of the GNU Lesser General Public License.

For Trial Runs of APPSPACK

Check out the NEOS Server

http://www-neos.mcs.anl.gov/neos/

For More Information on Related Papers

http://www.cs.wm.edu/~va/research/

http://csmr.ca.sandia.gov/~tgkolda/

In Particular

- *Asynchronous parallel pattern search for nonlinear optimization*, P. D. Hough, T. G. Kolda, and V. J. Torczon, SIAM Journal on Scientific Computing, Volume 23, Number 1, pages 134-156, 2001.
- *On the convergence of asynchronous parallel pattern search*, Tamara G. Kolda and Virginia J. Torczon. To be submitted to SIAM Journal on Optimization, Fall 2001.

Where Next?

Constraints!

We want to develop asynchronous parallel algorithms that guarantee global convergence to a constrained stationary point (i.e., a Karush–Kuhn–Tucker point).

Pattern search analysis tells us that robust algorithms are possible. The challenge is to make them effective in a high-performance computing environment.

The art lies in ensuring that $\mathcal{D}$ contains the right directions to prevent premature convergence to a nonstationary point.